[Introduction]

Discrete Structures (CSc 511)

Samujjwal Bhandari
Central Department of Computer Science and Information Technology (CDCSIT)
Tribhuvan University, Kirtipur,
Kathmandu, Nepal.

Introduction

Discrete Mathematics deals with discrete objects. Discrete objects are those objects that can be counted and are not connected for e.g. houses, trees, desks, integers, etc. So dealing with these discrete objects requires different concepts like counting techniques, knowledge of different discrete structures that are needed to understand what exactly discrete structure is like sets, relations, graphs, etc. we start our quest with foundation and go in depth later.

Logic

Logic is a language for reasoning. Since logic can helps us to reason the mathematical models it needs some rules associated with logic so that we can apply those rules for mathematical reasoning. There are lots of applications of logic in the field of computer science for e.g. designing circuits, programming, program verifications, etc.

Propositions and Propositional Calculus

Proposition is a fundamental concept in logic. Proposition is a declarative sentence that is either true or false, but not both. See the examples below:

2 + 2 = 5. (False), 7 - 1 = 6. (True)

It is hot today. (If it is hot then true)

Kathmandu is the capital of Nepal. (True)

All the above examples are either true or false.

Try to analyze the sentences below:

x > 5, Come here, Who are you?, 3 + 4

The above sentences are not propositions since we cannot say whether they are true or false.

Propositions are denoted conventionally by using small letters like p, q, r, s …. The truth value of proposition is denoted by **T** for true proposition and **F** for false proposition.

Reminder: p, q ,r ,s … are not actual propositions but they are propositional variables i.e. place holders for propositions.

The logic that deals with propositions is called propositional logic or propositional calculus.

Logical Operators/Connectives

Logical operators are used to construct mathematical statements having one or more propositions by combining the propositions. The combined proposition is called compound Proposition. The truth table is used to get the relationship between truth values of propositions. Here we present the logical operators along with their behavior in truth table:

Negation (not)

Given a proposition p, negation operator (¬) is used to get negation of p denoted by ¬p called "not p".

Example: Negation of the proposition " I love birds" is " I do not love birds" if the sentence I love birds is denoted by p then its negation is denoted by ¬p.

Truth table

p	¬p
T	**F**
F	**T**

Conjunction (and)

Given two propositions p and q, the proposition "p and q" denoted by p∧q is the proposition that is true whenever both the propositions p and q are true, false otherwise. The proposition that is obtained by the use of "and" operator is also called conjunction of p and q.

Example: If we have propositions p = "Ram is intelligent" and q = "Ram is diligent" the conjunction of p and q is Ram is intelligent and diligent. This proposition is true only when Ram is intelligent and he is diligent also, false otherwise.

Truth Table

p	q	p∧q
T	**T**	**T**
T	**F**	**F**
F	**T**	**F**
F	**F**	**F**

Disjunction (or)

Given two propositions p and q, the proposition "p or q" denoted by p∨q is the proposition that is false whenever both the propositions p and q are false, true otherwise. The proposition that is obtained by the use of "or" operator is also called disjunction of p and q.

Example: If we have propositions p = "Ram is intelligent" and q = "Ram is diligent" the disjunction of p and q is Ram is intelligent or he is diligent. This proposition is false only when Ram is not intelligent and not diligent, true otherwise.

Truth Table

p	q	p∨q
T	**T**	**T**
T	**F**	**T**
F	**T**	**T**
F	**F**	**F**

Exclusive or (Xor)

Given two propositions p and q, the proposition exclusive or of p and q denoted by p ⊕ q is the proposition that is true whenever only one of the propositions p and q is true, false otherwise. As opposed to the disjunction above which is inclusive the general meaning of the English sentence can be used to know whether the "or" used is inclusive or exclusive.

Example: If we have propositions p = "Ram drinks coffee in the morning" and q = "Ram drinks tea in the morning" the exclusive or of p and q is Ram drinks coffee or tea in the morning.

Truth Table

p	q	p ⊕ q
T	**T**	**F**
T	**F**	**T**
F	**T**	**T**
F	**F**	**F**

Implication

Given two propositions p and q, the proposition implication p → q is the proposition that is false when p is true and q is false, true otherwise. Here p is called "hypothesis" or "antecedent" or "premise" and q is called "conclusion" or "consequence".

We come across the implication in many places in mathematical reasoning and we use different terminologies to express p → q like: "if p, then q", "q is consequence of p", "p is sufficient for q", "q if p" "q is necessary for p", "q follows from p", "if p, q", "p implies q", "p only if q","q whenever p","q provides p"

Example: p = "today is Sunday" q = "it is hot" then the implication can be "if today is Sunday then it is hot today" or "today is Sunday only if it is hot today".

Truth Table

p	q	p → q
T	**T**	**F**
T	**F**	**F**
F	**T**	**T**
F	**F**	**T**

Contrapositive, Inverse and Converse

Some of the related implications formed from p → q are:

Converse: q → p (if it is hot today then today is Sunday).

Inverse: ¬p → ¬q (if today is not Sunday then it is not hot today).

Contrapositive: ¬q → ¬p (if it is not hot then today is not Sunday).

(Is contrapositive same as p → q? verify!!!).

Biconditional

Given propositions p and q, the biconditional p ↔ q is a proposition that is when p and q have same truth values. Alternatively p ↔ q is true whenever both q → p and q → p are true. Some of the terminologies used for biconditional are:

"p if and only if q" "if p then q, and conversely" "p is necessary and sufficient for q"

Example: For propositions given above in implication, "today is Sunday if and only if it is hot today".

Truth Table

p	q	$p \leftrightarrow q$
T	**T**	**T**
T	**F**	**F**
F	**T**	**F**
F	**F**	**T**

More Examples logical connectives:

1) Let, p = "it rained last night"

q = "the sprinkles came on last night"

r = "the lawn was wet this morning"

Translate the following into English $\neg p$, $r \wedge \neg p$ and $\neg r \vee p \vee q$.

$\neg p$ = " it didn't rain last night"

$r \wedge \neg p$ = "the lawn was wet this morning and it didn't rain last night"

$\neg r \vee p \vee q$ = "either the lawn was not wet this morning or it rained last night or the sprinkles came on last night"

2) Let p, q and r be the propositions with truth values **T**, **F**, **T** respectively. Evaluate the following:

$\neg r \vee \neg(p \wedge q)$, $\neg(p \vee q) \wedge (\neg r \vee q)$

$\neg r \vee \neg(p \wedge q) = \mathbf{F} \vee \neg(\mathbf{T} \wedge \mathbf{F}) = \mathbf{F} \vee \mathbf{T} = \mathbf{T}$ (true)

$\neg(p \vee q) \wedge (\neg r \vee q) = \neg(\mathbf{T} \vee \mathbf{F}) \wedge (\mathbf{F} \vee \mathbf{F}) = \mathbf{F} \wedge \mathbf{F} = \mathbf{F}$ (false)

Note: To translate English sentences to the proposition symbolic form follow these steps:

Restate the given sentence into building block sentences.

Give the symbol to each sentence and substitute the symbols using connectives

For e.g. "if it is snowing then I will go to the beach"

Restate into "it is snowing" give it symbol p and "I will go to the beach" and give it symbol q then we can write it as $p \rightarrow q$.

Propositional Equivalences

Given two propositions that differ in their syntax we may get the exactly same semantic for both the proposition. If two propositions are semantically identical then we say those two propositions are "equivalent". Such constructs are very useful in mathematical reasoning where we can substitute such propositions to equivalent propositions to construct mathematical arguments.

Tautology and Contradiction

A compound proposition that is always true, no matter what the truth values of the atomic propositions that contain in it, is called a tautology. For e.g. $p \vee \neg p$ is always true (verify!!!).

A compound proposition that is always false is called contradiction. For e.g. $p \wedge \neg p$ is always false (verify!!!).

A compound proposition that is neither a tautology nor a contradiction is called a contingency.

Logical Equivalences

The compound propositions p and q are logically equivalent, denoted by $p \Leftrightarrow q$ or $p \equiv q$, if proposition $p \leftrightarrow q$ is a tautology.

Some important logical equivalences

$p \wedge \mathbf{T} \Leftrightarrow p$	Identity law
$p \vee \mathbf{F} \Leftrightarrow p$	Identity law
$p \wedge \mathbf{F} \Leftrightarrow \mathbf{F}$	Domination law
$p \vee \mathbf{T} \Leftrightarrow \mathbf{T}$	Domination law
$p \wedge p \Leftrightarrow p$	Idempotent law
$p \vee p \Leftrightarrow p$	Idempotent law
$\neg(\neg p) \Leftrightarrow p$	Double negation law
$p \wedge q \Leftrightarrow q \wedge p$	Commutative law
$p \vee q \Leftrightarrow q \vee p$	Commutative law
$(p \wedge q) \wedge r \Leftrightarrow p \wedge (q \wedge r)$	Associative law

$(p \vee q) \vee r \Leftrightarrow p \vee (q \vee r)$	Associative law
$p \wedge (q \vee r) \Leftrightarrow (p \wedge q) \vee (p \wedge r)$	Distributive law
$p \vee (q \wedge r) \Leftrightarrow (p \vee q) \wedge (p \vee r)$	Distributive law
$\neg(p \wedge q) \Leftrightarrow \neg p \vee \neg q$	De Morgan's law
$\neg(p \vee q) \Leftrightarrow \neg p \wedge \neg q$	De Morgan's law
$p \wedge \neg p \Leftrightarrow \mathbf{F}$	Trivial tautology
$p \vee \neg p \Leftrightarrow \mathbf{T}$	Trivial tautology
$p \rightarrow q \Leftrightarrow \neg p \vee q$	Implication
$p \leftrightarrow q \Leftrightarrow (p \rightarrow q) \wedge (q \rightarrow p)$	Equivalence
$(p \wedge q) \rightarrow r \Leftrightarrow p \rightarrow (q \rightarrow r)$	Exportation
$(p \rightarrow q) \wedge (p \rightarrow \neg q) \Leftrightarrow \neg p$	Absurdity
$p \rightarrow q \Leftrightarrow \neg q \rightarrow \neg p$	Contrapositive
$p \wedge (p \vee q) \Leftrightarrow p$	Absorption law
$p \vee (p \wedge q) \Leftrightarrow p$	Absorption law

Proving logical equivalence

a) Truth Table (for $(p \rightarrow q) \wedge (p \rightarrow \neg q) \Leftrightarrow \neg p$)

p	¬p	q	¬q	p→q	p→¬q	p→q∧p→¬q
T	**F**	**T**	**F**	**T**	**F**	**F**
T	**F**	**F**	**T**	**F**	**T**	**F**
F	**T**	**T**	**F**	**T**	**T**	**T**
F	**T**	**F**	**T**	**T**	**T**	**T**

b) Symbolic Derivation

Prove $(p \wedge \neg q) \rightarrow \neg (p \leftrightarrow r) \Leftrightarrow \neg p \vee q \vee \neg r$

Solution:

	$(p \wedge \neg q) \rightarrow \neg (p \leftrightarrow r)$	
$\equiv$	$\neg(p \wedge \neg q) \vee \neg (p \leftrightarrow r)$	[Implication]
$\equiv$	$(\neg p \vee q) \vee \neg (p \leftrightarrow r)$	[De Morgan's law]
$\equiv$	$(\neg p \vee q) \vee \neg[(p \rightarrow r) \wedge (r \rightarrow p)]$	[Biconditional equivalence]
$\equiv$	$(\neg p \vee q) \vee \neg[(\neg p \vee r) \wedge (\neg r \vee p)]$	[Implication]

≡ $(\neg p \vee q) \vee \neg[\{(\neg p \vee r) \wedge \neg r\} \vee \{(\neg p \vee r) \wedge p\}]$ [Distributive law]

≡ $(\neg p \vee q) \vee \neg[\{(\neg r \wedge \neg p) \vee (r \wedge \neg r)\} \vee \{(p \wedge \neg p) \vee (r \wedge p)\}]$ [Distributive law]

≡ $(\neg p \vee q) \vee \neg[\{(\neg r \wedge \neg p) \vee \mathbf{F}\} \vee \{\mathbf{F} \vee (r \wedge p)\}]$ [Trivial tautology]

≡ $(\neg p \vee q) \vee \neg[(\neg r \wedge \neg p) \vee (r \wedge p)]$ [Identity law]

≡ $(\neg p \vee q) \vee \neg(\neg(r \wedge q)) \wedge \neg(r \wedge p)$ [De Morgan's law]

≡ $(q \vee \neg p) \vee ((r \vee p) \wedge \neg(r \wedge p))$ [Commutative and Double negation]

≡ $q \vee (\neg p \vee ((r \vee p) \wedge \neg(r \wedge p)))$ [Associative law]

≡ $q \vee ((\neg p \vee (r \vee p)) \wedge (\neg p \vee \neg(r \wedge p)))$ [Distributive law]

≡ $q \vee (((\neg p \vee p) \vee r) \wedge (\neg p \vee \neg(r \wedge p)))$ [Associative and commutative laws]

≡ $q \vee ((\mathbf{T} \vee r) \wedge (\neg p \vee \neg(r \wedge p)))$ [Trivial tautology]

≡ $q \vee (\mathbf{T} \wedge (\neg p \vee \neg(r \wedge p)))$ [Domination law]

≡ $q \vee (\neg p \vee \neg(r \wedge p))$ [Identity law]

≡ $q \vee (\neg p \vee (\neg r \vee \neg p))$ [De Morgan's law]

≡ $q \vee ((\neg p \vee \neg p) \vee \neg r)$ [Commutative and Associative laws]

≡ $q \vee (\neg p \vee \neg r)$ [Idempotent law]

≡ $\neg p \vee q \vee \neg r$ [Associative and Commutative laws]

Proved.

Predicate

We studied propositional logic. Lets take a statement "x > 5" is this statement a proposition? The answer is no. Whenever the statements have variable(s) in them we cannot say those statements as a proposition. The question here is can we make such statements to propositions? The answer here is yes.

In the above statement there are two parts one is the variable part called "subject" and another is relation part ">5" called "predicate". We can denote the statement "x>5" by P(x) where P is predicate ">5" and x is the variable. We also call P as a propositional function where P(x) gives value of P at x. Once value is assigned to the propositional function then we can tell whether it is true or false i.e. a proposition.

For e.g. if we put the value of x as 3 and 7 then we can conclude that P(3) is false since 3 is not greater than 5 and p(7) is true since 7 is greater than 5.

We can also denote a statements with more than one variable using predicate like for the statement "x = y" we can write P(x,y) such that P is the relation "equals to" . Similarly the statements with higher number of variables can be expressed.

Remember: The logic involving predicates is called Predicate Logic or Predicate calculus similar to logic involving propositions is Propositional Logic or Propositional Calculus

Quantifiers

Quantifiers are the tools to make the propositional function a proposition. Construction of propositions from the predicates using quantifiers is called quantification. The variables that appear in the statement can take different possible values and all the possible values that the variable can take forms a domain called "Universe of Discourse" or "Universal set". We study two types of quantifier Universal quantifier and Existential quantifier.

Universal Quantifier

Universal quantifier, denoted by $\forall$, is used for universal quantification. The universal quantification of P(x), denoted by $\forall x$ P(x), is a proposition "P(x) is true for all the values of x in the universe of discourse".

We can represent the universal quantification by using the English language like:

"for all x P(x) holds" or "for every x P(x) holds" or "for each x P(x) holds".

Example:

Take universe of discourse a set of all students of CDCSIT.

P(x) represents x takes graphics class.

Here universal quantification is $\forall x$ P(x), i.e. "all students of CDCSIT take graphics class", is a proposition.

The universal quantification is conjunction of all the propositions that are obtained by assigning the value of the variable in the predicate. Going back to above example if universe of discourse is a set {ram, shyam, hari, sita} then the truth value of the universal quantification is given by P(ram) $\wedge$ P(shyam) $\wedge$ P(hari) $\wedge$ P(sita) i.e. it is true only if all the atomic propositions are true.

Existential Quantifier

Universal quantifier, denoted by ∃, is used for existential quantification. The existential quantification of P(x), denoted by ∃x P(x), is a proposition "P(x) is true for some values of x in the universe of discourse". The other forms of representation include "there exists x such that P(x) is true" or " P(x) is true for at least one x".

Example:

For the same problem given in universal quantification ∃x P(x) is a proposition is represent like " some students of CDCSIT take graphics class".

The existential quantification is the disjunction of all the propositions that are obtained by assigning the values of the variable from the universe of discourse. So the above example is equivalent to P(ram) ∨ P(shyam) ∨ P(hari) ∨ P(sita), where all the instances of variable are as in example of universal quantification. Here if at least one of the students takes graphics class then the existential quantification results true.

Free and Bound Variables

When the variable is assigned a value or it is quantified it is called bound variable. If the variable is not bounded then it is called free variable. A part of a logical expression that is quantified is given by the scope of the quantifier. We use parenthesis to give scope of the quantifier. For e.g. ∀x (P(x)) → q is not same as ∀x (P(x) → q)

Example:

P(x,y) has two free variables x and y.

P(2, y) has one bound variable 2 and one free variable y.

P(2,y) where y = 4, is bounding the variable y also.

∀x P(x) has a bound variable x.

∀x P(x,y) has one bound variable x and one free variable y.

Expression with no free variable is a proposition.

Expression with at least one free variable is a predicate only.

Order of Quantification

Order of quantification goes from the left to right. If we have a quantified proposition involving two variable (nested quantifier) then the order must be considered.

Example: Let L(x,y) denotes x loves y where universe of discourse for x, y is set of all people in the world. Translate ∀x∃yL(x,y), ∃y∀x L(x,y), ∃x∀y L(x,y), ∀y∃xL(x,y), ∀x∀yL(x,y) and ∃x∃yL(x,y) into English.

Solution:

∀x∃yL(x,y) : [for all x there is some y such that x loves y i.e. everybody loves someone. This is false when there is someone who doesn't love any one]

∃y∀x L(x,y): [for some y all x love y i.e. there is a people who is loved by everyone. This is false when there is no person who is loved.]

∃x∀y L(x,y): [There is some x such that x loves all y i.e. there is someone who loves all the people. This is false when all people do not love some people]

∀y∃xL(x,y): [for all y there is x who loves y i.e. everyone has someone who loves them. When this is false? (try yourself)]

∀x∀yL(x,y): [for all x, x loves all the y i.e. everybody loves everyone. When this is false ? (try yourself)]

∃x∃yL(x,y): [There is some x such that he loves some y i.e. someone loves somebody. When this is false? (try yourself)]

Negation of Quantifies Expression

Let P(x) denotes x is lovely, universe of discourse for x is girls in Kathmandu. Then,

∀x P(x) is every girl in Kathmandu is lovely. If we want to negate it the meaning would be like there is a girl in Kathmandu who is not lovely i.e. ∃x¬P(x).

∃xP(x) is at least a girl in Kathmandu is lovely. The opposite for this (negation) would be no girls in Kathmandu are lovely. i.e. ∀x ¬P(x).

The negation of the nested quantifier can be done by successively negating the quantifier using the above negation rule for single quantifier for e.g. ¬ (∀y∃xP(x,y)) is

∃y(¬∃xP(x,y)) = ∃y∀x¬ P(x,y).

Translating the Sentences into Logical Expression

Example 1

Translate "not every integer is even" where the universe of discourse is set of integers.

Solution:

Let E(x) denotes x is even. $\neg\forall xE(x)$

Example 2

Translate "if a person is female and is a parent, then this person is someone's mother" into logical expression, Universe of discourse is set of all people.

Solution:

Let F(x) denotes x is female, P(x) denotes x is a parent and M(x,y) denotes x is a mother of y. then the logical expression for above sentence is $\forall x\exists y\ (F(x) \wedge P(x) \rightarrow M(x,y))$

Example 3

Translate "everyone has exactly one best friend" into logical expression where universe of discourse is set of all people.

Solution:

Let B(x,y) denotes y is best friend of x then $\forall x\exists y(B(x,y) \wedge \forall z\ (B(x,z) \rightarrow (y = z))$ is the solution.

Sets and Set Operations

Set is a very important concept in mathematics. In computer science also we deal with set in most of the case. For e.g. if we are dealing with relations in database then they are sets ordered collection of elements, similarly we can view graph as a set. Set is a collection of zero or more objects (or elements or members), the elements need not be ordered. If we denote set by S and some element from the set by e then we say "e belongs to S" or "S contains e" or in symbol we can write $e \in S$. for e.g. V = {a, e, i, o, u} is a set of vowels and $i \in V$, if some object doesn't belong to the set we write it as " does not belong to" i.e. say $x \notin V$; C = {b, c, d, f, g, h, j, k, l, m, n, p, q, r, s, t, v, w, y, z) is a set of consonants.

Representations of Sets

Some of the ways of representing a set are:

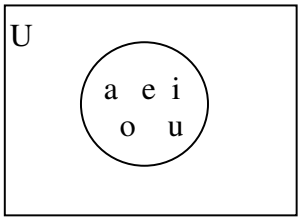
Listing of elements: Set of vowels is {a, e, i, o, u}.

Set builder form: here propertied of the members of the set is described for e.g. R = {x | x is a real number}

Recursive formula: the elements of set are defined using the previous element of the set

that is known. For e.g. set of natural numbers can be represented as $N = \{x_n = x_{n-1} + 1$, where $x_0 = 0\}$.

Venn diagram: graphical representation of set. For e.g. set of vowel as given above can be represented as:



In the above diagram the circle represents the set of vowels where as the enclosed rectangle represents the "universe of discourse" or "universe".

Some Definitions

Subset: Let A and B be two sets. Then A is said to be subset of B if every elements of A is an element of B. A is said to be proper subset of B if A is subset of B and there is at least one element in B that is not in A. Symbolically subset is represented as $A \subseteq B$ to denote that A is subset of B and $A \subset B$ to denote that A is proper subset of B.

Some subsets related properties

$A \subseteq A$; If $A \subseteq B$ and $B \subseteq C$ then $A \subseteq C$; If $A \subseteq B$ and $B \subset C$ then $A \subset C$; If $A \subseteq B$ and $A \not\subset C$ then $B \not\subset C$, where $\not\subset$ means "is not contained in". As subset is defined above **superset** can be defined in similar manner where in the above definition B is the superset of A denoted by $B \supseteq A$ for superset and $B \supset A$ for proper superset.

Equal Sets: Two sets A and B are equal if and only if they contain exactly same elements. In other words if $A \subseteq B$ and $B \subseteq A$ then $A = B$.

For e.g. $A = \{1, 2, 3, 4, 5\}$ and $B = \{2, 5, 4, 1, 3\}$ are equal sets.

Empty set: The set that contains no element is called empty set and denoted by $\varnothing$. It is also called null set. We have $\varnothing = \{\}$ but $\varnothing \neq \{\varnothing\}$.

Cardinality: For the set S, if there are exactly *n distinct* elements in S where n is a number then we say that cardinality of the set S is n denoted by $|S|$.

For e.g. $|\varnothing| = 0$; $|\{a, b, b, c, a\}| = 3$; $|\{\{a, b\}, \{a, b, c\}\}| = 3$

If $n \in N$ then the set is finite otherwise, it is infinite.

Power Set: Given a set S, power set denoted by P(S) is the set that contains all the subsets of the set S. Symbolically we can write $P(S) = \{x \mid x \subseteq S\}$. For e.g. power set for the set {2, 3} is {∅, {2}, {3}, {2,3}}. The number of elements in the power set of set having n elements is $2^{|n|}$. *Remember: ∅ is member of all power set.*

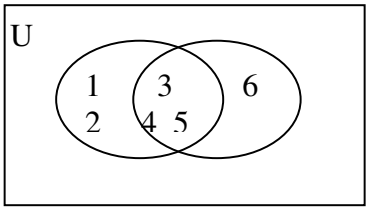
Set Operations

Union Operator: Given two sets A and B, the union of set A and set B is the set that contains those elements that are either in A or in B, or in both, denoted by $A \cup B$. Symbolically, we write union of A and B as: $A \cup B = \{x \mid x \in A \vee x \in B\}$.

Example:

$\{2, 3\} \cup \{a, b, c\} = \{2, 3, a, b, c\}$.

$\{1, 2, 3, 4, 5\} \cup \{3, 4, 5, 6, 7\} = \{1, 2, 3, 4, 5, 6, 7\}$. This can be shown in Venn diagram as

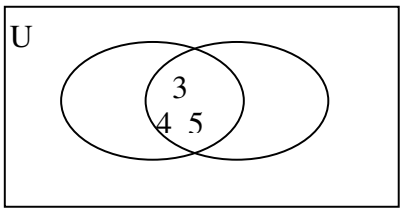


Intersection Operator: Given two sets A and B, the intersection of set A and set B is the set that contains those elements that are in both A and B, denoted by $A \cap B$. Symbolically, we write intersection of A and B as: $A \cap B = \{x \mid x \in A \wedge x \in B\}$.

Example:

$\{2, 3\} \cap \{a, b, c\} = \{\ \}$.

$\{1, 2, 3, 4, 5\} \cap \{3, 4, 5, 6, 7\} = \{3, 4, 5\}$. This can be shown in Venn diagram as



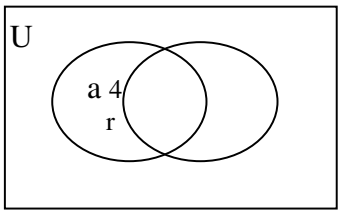
Disjoint Set: Two sets are disjoint if the intersection of two sets is null set.

Set Difference: Given two sets A and B, The difference of A and B is the set that contains all the elements that are in A but not in B, denoted by A –B. This difference is also called complement of B with respect to A. Symbolically we write difference of A

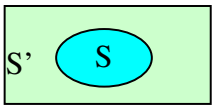
and B as $A - B = \{x \mid x \in A \wedge x \notin B\}$.

Examples:

A = {2, 4, a, r} and B = {1, 2, a, s, t} then A – B = {4, r}. Venn diagram is given below:



Complement: Complement of set S is denoted by U – S or S', where U is the universal set, is the of difference of universal set U and set S. Symbolically complement is written as $S' = \{x \mid x \notin S\}$.



In the Venn diagram shown above, the outer part from the oval is complement of S.

Set Identities

The set identities that we learn here is similar to that of propositional logic.

$A \cap U = A$	Identity law
$A \cup \varnothing = A$	Identity law
$A \cap \varnothing = \varnothing$	Domination law
$A \cup U = U$	Domination law
$A \cap A = A$	Idempotent law
$A \cup A = A$	Idempotent law
$(A')' = A$	Complementation law
$A \cap B = B \cap A$	Commutative law
$A \cup B = B \cup A$	Commutative law
$(A \cap B) \cap C = A \cap (B \cap C)$	Associative law
$(A \cup B) \cup C = A \cup (B \cup C)$	Associative law
$A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$	Distributive law
$A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$	Distributive law
$(A \cap B)' = A' \cup B'$	De Morgan's law
$(A \cup B)' = A' \cap B'$	De Morgan's law

Generalized Union and Intersection

Since union and intersection of the sets holds associativity we can combine the sets with same operators in any order. The union of a collection of sets is the set that contains those elements that are members of at least one set in the collection. The notation we use for this operation is $A_1 \cup A_2 \cup \ldots \cup A_n = \bigcup_{i=1}^{n} A_i$. The intersection of the collection of the sets is the set that contains those elements that are in all the sets in the collection. We represent generalized intersection as $A_1 \cap A_2 \cap \ldots \cap A_n = \bigcap_{i=1}^{n} A_i$.

Proving Set Identities

Using Mutual Subsets: Show $A \subseteq B$ and $B \subseteq A$ to show $A = B$.

Example: Show $A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$.

Step1: Assume $x \in A \cap (B \cup C)$ and try to show $x \in (A \cap B) \cup (A \cap C)$.

Then we know, $x \in A$ and $x \in B$ or $x \in C$, or both.

Since $x \in A$ and $x \in B$, $x \in (A \cap B)$. Hence $x \in (A \cap B) \cup (A \cap C)$.

Since $x \in A$ and $x \in C$, $x \in (A \cap C)$. Hence $x \in (A \cap B) \cup (A \cap C)$.

Therefore, $A \cap (B \cup C) \subseteq (A \cap B) \cup (A \cap C)$.

Step 2: Assume $x \in (A \cap B) \cup (A \cap C)$ and try to show $x \in A \cap (B \cup C)$.

Then we know $x \in (A \cap B)$, that is $x \in A$ and $x \in B$, or $x \in (A \cap C)$, that is $x \in A$ and $x \in C$, or both. In any case we have $x \in A$ as true.

Since $x \in B$, $x \in (B \cup C)$. So $x \in A \cap (B \cup C)$.

Since $x \in C$, $x \in (B \cup C)$. So $x \in A \cap (B \cup C)$.

Therefore, $(A \cap B) \cup (A \cap C) \subseteq A \cap (B \cup C)$.

From step 1 and step 2 $A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$.

Using Logical Equivalences: represent the set to the set builder form and apply logical equivalence transformations.

Example: Show $(A \cup B)' = A' \cap B'$.

We can denote $(A \cup B)' = \{x \mid x \notin A \cup B\} = \{x \mid \neg(x \in A \cup B)\}$

$= \{x \mid \neg(x \in A \vee x \in B)\} = \{x \mid x \notin A \wedge x \notin B)\}$

$= \{x \mid x \in A' \wedge x \in B')\}$ $= \{x \mid x \in A' \cap B'\}$

$= A' \cap B'$

Membership Table: Like truth table you have studied earlier for propositional logic. Use 1 to denote set membership and 0 otherwise.

Example: Show $A - (A - B) = A \cap B$.

A	B	A - B	A - (A - B)	A ∩ B
1	1	0	1	1
1	0	1	0	0
0	1	0	0	0
0	0	0	0	0

Cartesian Products

Sets are unordered collections of objects but sometime we need ordered collections. Such ordered collections can be obtained from ordered n- tuples.

Ordered n- tuple: The ordered n tuple $(a_1, a_2, \ldots, a_n)$ is the ordered collection where a_1 is the first element, a_2 is the second element and so on …. Two ordered n – tuples are equal if and only if they have each corresponding pair of their elements is equal i.e. $(a_1, a_2, \ldots, a_n) = (b_1, b_2, \ldots, b_n)$ if and only if $a_1 = b_1, a_2 = b_2, \ldots, a_n = b_n$.

Cartesian Product: Given sets A and B. The Cartesian product of A and B is the set of all ordered pairs (a, b) where $a \in A$ and $b \in B$. The Cartesian product of A and B is denoted by $A \times B$. Symbolically, we can write it as $A \times B = \{(a, b) \mid a \in A \wedge b \in B\}$

Example: Let $A = \{a, b, c\}$ and $B = \{1, 2, 3\}$ then

$A \times B = \{(a, 1), (a, 2), (a, 3), (b, 1), (b, 2), (b, 3), (c, 1), (c, 2), (c, 3)\}$

Remember: $A \times B \neq B \times A$ unless $A = \varnothing$ or $B = \varnothing$ or $A = B$ (verify!!!)

Similarly Cartesian product of more than two sets can be defined as, Cartesian product of $A_1, A_2, \ldots, A_n$ is the set of ordered n-tuples $(a_1, a_2, \ldots, a_n)$ where each $a_i \in A_i$, for i =1, 2, …, n. It is denoted by $A_1 \times A_2 \times \ldots \times A_n$.

Symbolically, $A_1 \times A_2 \times \ldots \times A_n = \{(a1, a_2, \ldots, a_n) \mid a_i \in A_i$, for i =1, 2, …, n$\}$

Example: $A = \{1, 2\}$, $B = \{2\}$ and $C = \{a, b, c\}$ then

$A \times B \times C = \{(1, 2, a), (1, 2, b), (1, 2, c), (2, 2, a), (2, 2, b), (2, 2, c)\}$

Relations (Intro)

Binary Relation: Given sets A and B, a binary relation from A to B is a subset of $A \times B$ i.e. a binary relation from A to B is a set R of ordered pairs where first element of each ordered pair belongs to set A and the second element belongs to the set B. the notation aRb is used to denote that $(a, b) \in R$ and we say a is related to b by R.

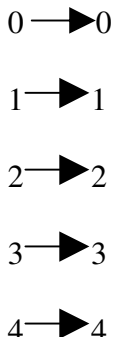
Example:

$A = \{1, 2, 3\}$ $B = \{a, b\}$ then relations from the above two sets can be

$R = \{(1, a), (2, a)\}$ $S = \{(1,a), (1, b), (2, a), (2, b)\}$ and others are also possible.

Relations on a Set: A relation on the set A is the relation from A to A i.e. it is the subset of $A \times A$.

Example: Let $A = \{0, 1, 2, 3, 4\}$ the ordered pairs that are in the relation $R = \{(a, b) \mid a = b\}$ is given by $R = \{(0, 0), (1, 1), (2, 2), (3, 3), (4, 4)\}$, We can view this relation as



R	0	1	2	3	4
0	#				
1		#			
2			#		
3				#	
4					#

Properties of Relations

Reflexive: A relation R on a set A is called reflexive if $(a, a) \in R$ for every element $a \in A$. For e.g. relation $\leq$ on set of integers is reflexive.

Symmetric: A relation R on set A is called symmetric if $(a, b) \in R$ then $(b, a) \in R$. for $a, b \in A$. For e.g. relation = on set of integers is symmetric.

Transitive: A relation R on a set A is called transitive if $(a, b) \in R$ and $(b, c) \in R$ then $(a, c) \in R$, for $a, b, c \in A$. For e.g. relation $\leq$ on set of integers is transitive.

Antisymmetric: A relation R on a set A is called antisymmetric if $(a, b) \in R$ and $(b, a) \in R$ then $a = b$, for $a, b \in A$. For e.g. relation $\geq$ on set of integers is antisymmetric.

Asymmetric: A relation R on a set A is called asymmetric if (a, b) ∈ R then (b, a) ∉ R, for a, b ∈ A. For e.g. relation > on set of integers is asymmetric.

Irreflexive: A relation R on a set A is called irreflexive if for every a ∈ A, (a, a) ∉ R. For e.g. relation > on set of integers is irreflexive.

Combining Relations

Relation from A to B is a subset of A × B so any operations that are operable in sets are also operable in relations (see notes on sets for detail).

Composite Relation: Let R and S be the relations from A to B and B to C respectively. The composite relation of R and S is a set having ordered pairs (a, c), where a∈ A and c ∈ C, and for which there exists an element b ∈ B such that (a, b) ∈ R and (b, c) ∈ . The composite relation of R and S is denoted by SoR.

Example: Let R = {(a, 1), (a, 2), (b, 1)} and S = {(1, x), (2, y)} where A = {a, b}, B ={1, 2} and C = {x, y}. Then, SoR = {(a, x), (a, y), (b, x)}

On the basis of composite relation the powers of a relation R can be defined as, The powers R^n, n =1, 2, 3, …are inductively defined as $R^1 = R$ and $R^{n+1} = R^n oR$.

Note: More on the relations will be covered later

Functions

Sometimes we assign each element form a set to the elements of other set that may be the same as the first. For e.g. each worker working on the factory to the set of the works. This kind of assignment gives rise to the function.

Function: given two A and B**,** A function f from A to B is an assignment of unique element of B to each element of A. if b is the unique element of B assigned by the function f to the element a of A the we write f(a) = b. A function from A to B is written as f: A → B. Given a function f: A → B where f(a) = b then we define following terms:

Domain: Set A is the domain of function f.

Codomain: Set B is the codomain of function f.

Image: b is the image of a.

Pre-image: a is the pre-image of b.

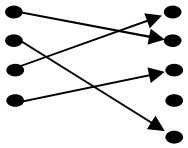
Range: Set of all images of elements of A is range.

Types of Functions

One – to – One (Injective) Function: A function f is one-to-one, if and only if f(x) = f(y) implies x = y for all x and y in the domain of f.

Example: f(x) = x2 from set of integers to the set of integers is not an injection because f(-1) = f(1) = 1 does not imply –1 = 1.

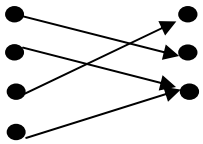
The pictorial representation of one-to-one functions looks like:



Onto (Surjective) Function: A function f is surjective or onto if and only if for every element b ∈ B there is an element a ∈ A such that f(a) = b.

Example: The function f(x) = x +1 from the set of integers to the set of integers is onto because for every integer b there is an integer a such that f(a) = b, where each a = b-1.

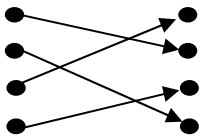
The pictorial representation of surjection is as below:



One-to-One Correspondence (Bijective Function): A function is bijection if it is both onto and one-to-one.

Example: The function f(x) = x +1 from the set of integers to the set of integers is bijection since it is one-to-one (How?) and onto (see above).

The pictorial representation of bijection is as below:



Inverse Function: given a bijective function f: A → B, the inverse of function f is denoted by f^{-1} assigns each element of B to the unique element of A such that f(a) = b. so we can write f^{-1} (b) = a when f(a) = b.

Example: The function f(x) = x +1 from the set of integers to the set of integers is bijection (see above) hence we can have inverse of it and it is denoted as f^{-1} (x) = x-1.

Growth of Functions

Complexity analysis of an algorithm is very hard if we try to analyze exact. we know that the complexity (worst, best, or average) of an algorithm is the mathematical function of the size of the input. So if we analyze the algorithm in terms of bound (upper and lower) then it would be easier i.e. understanding the growth of the function is easier. For this purpose we need the concept of asymptotic notations.

Big Oh (O) notation

When we have only asymptotic upper bound then we use O notation. A function f(x) = O(g(x)) (read as f(x) is big oh of g(x)) iff there exists two positive constants c and x_0 such that for all x >= x_0, 0 <= f(x) <= c*g(x)

The above relation says that g(x) is an upper bound of f(x)

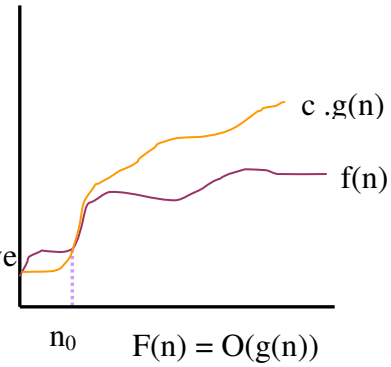
Some properties:

Transitivity : f(x) = O(g(x)) & g(x) = O(h(x)) $\Rightarrow$ f(x) = O(h(x))

Reflexivity: f(x) = O(f(x))

O(1) is used to denote constants.

For all values of n >= n_0, plot shows clearly that f(n) lies below or on the curve of c*g(n)



Examples

1. f(n) = $3n^2 + 4n + 7$

   g(n) = n^2 , then prove that f(n) = O(g(n)).

   Proof: let us choose c and n_0 values as 14 and 1 respectively then we can have f(n) <= c*g(n), n>=n0 as

   $3n^2 + 4n + 7 <= 14*n^2$ for all n >= 1

   the above inequality is trivially true

   hence f(n) = O(g(n))

2. Prove that n log (n^3) is O($\sqrt{n^3}$)).

Proof: we have n log (n^3) = 3n log n

again, $\sqrt{n^3}$ = n $\sqrt{n}$,

if we can prove log n = O($\sqrt{n}$) then problem is solved

because n log n = n O($\sqrt{n}$) that gives the question again.

We can remember the fact that log ^{a} n is O (n^b) for all a,b>0.

In our problem a = 1 and b = ½ ,

hence log n = O($\sqrt{n}$).

So by knowing log n = O($\sqrt{n}$) we proved that

n log (n^3) = O($\sqrt{n^3}$)).

3. Is 2^{n+1} =O(2^n) ?

Is 2^{2n} = O(2^n) ?

Big Omega (Ω) notation

Big omega notation gives asymptotic lower bound. A function f(x) = Ω(g(x)) (read as f(x) is big omega of g(x)) iff there exists two positive constants c and x_0 such that for all x >= x_0,

0 <= c*g(x) <= f(x).

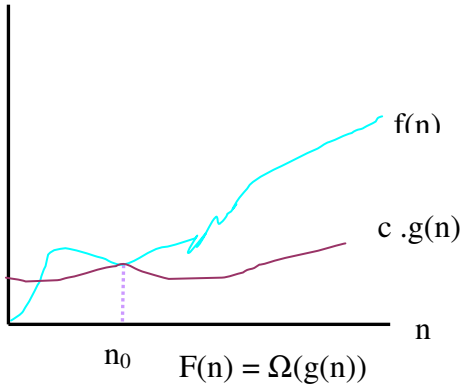
The above relation says that g(x) is an lower bound of f(x).

some properties:

Transitivity : f(x) = O(g(x)) & g(x) = O(h(x)) ⇒ f(x) = O(h(x))

Reflexivity: f(x) = O(f(x))

For all values of n >= n_0, plot shows clearly that f(n) lies above or on the curve of c*g(n).



Examples

1. f(n) = $3n^2 + 4n + 7$

g(n) = n^2 , then prove that f(n) = Ω(g(n)).

Proof: let us choose c and n_0 values as 1 and 1, respectively then we can have

f(n) >= c*g(n), n>=n0 as

$3n^2 + 4n + 7 >= 1*n^2$ for all n >= 1

the above inequality is trivially true

hence f(n) = Ω(g(n))

Big Theta (Θ) notation

When we need asymptotically tight bound then we use notation. A function f(x) = (g(x)) (read as f(x) is big theta of g(x)) iff there exists three positive constants c_1, c_2 and x_0 such that for all x >= x_0, 0 <= c_1*g(x) <= f(x) <= c_2*g(x)

The above relation says that f(x) is order of g(x)

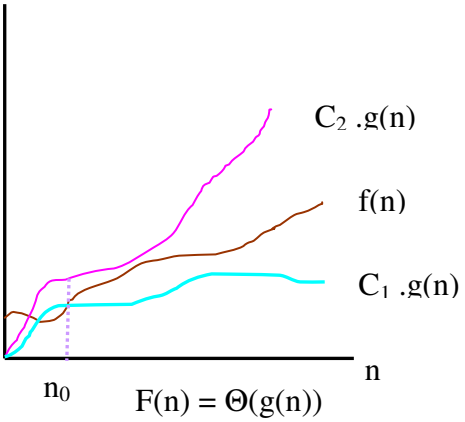
some properties:

Transitivity : f(x) = Θ (g(x)) & g(x) = Θ (h(x)) ⇒ f(x) = Θ (h(x))

Reflexivity: f(x) = Θ (f(x))

Symmetry: f(x) = Θ g(x) iff g(x) = Θ f(x)

For all values of n >= n_0, plot shows clearly that f(n) lies between c_1* g(n)and c_2*g(n).



Examples

1. f(n) = $3n^2 + 4n + 7$

   g(n) = n^2 , then prove that f(n) = (g(n)).

   Proof: let us choose c1, c2 and n_0 values as 14, 1 and 1 respectively then we can have,

   f(n) <= c1*g(n), n>=n0 as $3n^2 + 4n + 7 <= 14*n^2$, and

   f(n) >= c2*g(n), n>=n0 as $3n^2 + 4n + 7 >= 1*n^2$

   for all n >= 1(in both cases).

   So c2*g(n) <= f(n) <= c1*g(n) is trivial.

   Hence f(n) = Θ (g(n)).

2. Show $(n + a)^b = \Theta(n^b)$, for any real constants a and b, where b>0.

Here, using Binomial theorem for expanding $(n + a)^b$, we get ,

$C(b,0)n^b + C(b,1)n^{b-1}a + \ldots + C(b,b-1)na^{b-1} + C(b,b)a^b$

we can obtain some constants such that $(n + a)^b <= c_1*(n^b)$, for all $n >= n_0$

and

$(n + a)^b >= c_2*(n^b)$, for all $n >= n_0$, here we may take $c_1 = 2^b$ $c_2 = 1$ $n_0 = |a|$,

since $1 *(n^b) <= (n + a)^b <= 2^b *(n^b)$.

Hence the problem is solved.

Why $c_1 = 2^b$? since $\Sigma C(n,k) = 2^k$, where k=0 to n.

Little Oh (o) notation

Little oh (o) notation is used to denote the upper bound that is not asymptotically tight. A function f(x) = o(g(x)) (read as f(x) is little oh of g(x)) iff for any positive constant c there exists positive constant x_0 such that for all $x >= x_0$,

$0 <= f(x) < c*g(x)$

for example $4x^4$ is $O(x^4)$ but not $o(x^4)$.

Alternatively f(x) is little oh of g(x) if $\lim_{x->\infty} \frac{f(x)}{g(x)} = 0$

Note: transitivity is satisfied.

Little Omega (ω) notation

Little omega (o) notation is used to denote the lower bound that is not asymptotically tight. A function f(x) = ω(g(x)) (read as f(x) is little omega of g(x)) iff for any positive constant c there exists positive constant x_0 such that for all $x >= x_0$,

$0 <= c*g(x) < f(x)$.

for example $x^3/7$ is $\omega(x^2)$ but not $\omega(x^3)$.

Alternatively f(x) is little omega of g(x) if $\lim_{x->\infty} \frac{f(x)}{g(x)} = \infty$

Note: transitivity is satisfied.

[Algorithms & Complexity]

Discrete Structures (CSc 511)

Samujjwal Bhandari
Central Department of Computer Science and Information Technology (CDCSIT)
Tribhuvan University, Kirtipur,
Kathmandu, Nepal.

Algorithm

An algorithm is a finite set of instructions executed in finite time for problem solving or performing computation. To represent an algorithm we use English language however for the simplicity we use pseudo code that can represent an algorithm in clear manner like in English language and gives the implementation view as in the programming languages.

Example:

Write an algorithm for finding factorial of the given number.

Algorithm:

```
fact(n)
{
fact = 1;
for(i = 1; i<=n; i++)
        fact = fact*i;
return fact;
}
```

See the above algorithm is for getting the factorial of n. The algorithm first assigns the value 1 to the variable fact and then until the n is reached the fact is assigned a value fact * i where value of i is from 1 to n. At last the value fact is returned.

Algorithm Properties

Input/Output: An algorithm has input or set of inputs values from the set that has possible input values and for each set of inputs an algorithm produces the solution of the problem that are in the set of output values.

Definiteness: Each step must be clear and unambiguous.

Correctness: An algorithm produced output must be correct for each set of input values.

Finiteness: The algorithm must terminate after finite amount of time for every possible set of input values.

Effectiveness: Each step must be executable in finite time.

Generality: The devised algorithm must be capable of solving the problem of similar kind for all possible inputs.

The algorithm provided above i.e fact(n) has all the properties defined above. Input for an algorithm is any positive integers and the output is the factorial of the given number. Each step is clear since assignments, finite loop, and the arithmetic operations and jump statements are unambiguous. Correctness requires rigorous explanation and you will read this later at this point just see that each input gives its corresponding factorial value. When the return statement is reached the algorithm terminates and each step terminates in finite time all other steps other than loop are simple and they need no explanation. In case of loop when the value of i reaches n+1 then the loop terminates. An algorithm is general since it can work out for every positive integer.

Complexity of Algorithms

When an algorithm is designed it must be analyzed for its efficiency. The efficiency of an algorithm is measured in terms of complexity. The complexity of algorithms is mentioned in terms of resource needed by the algorithm. We generally consider two kinds of resources used by an algorithm time and space. The measure of time required by an algorithm to run is given by **time complexity** and the measure of space (computer memory) required by an algorithm is given by **space complexity**. Here in this course we generally discuss time complexity of an algorithm that is given by the number of operations needed by an algorithm for given set of inputs. Since actual time required may vary from computers to computers we use number of operations required to measure the time complexity.

Example (Time complexity):

In the above algorithm of finding factorial we can find time complexity by enumerating the number of operations required for the algorithm to obtain the factorial of n. you can see that the assignment of the variable fact is done one time, the for loop executes for n+1 time and the statement inside the for loop executes for n times and the last operation that is due to the return statement executes single time. The total time in terms of number of

operations required can be written as

T(n) = 1+ (n+1) + n + 1 = 2n + 3

You have already read growth of functions so denoting time complexity in terms of big oh (O) notation for the above algorithm is O(n) i.e. linear time complexity.

Example Binary Search Algorithm:

The binary search problem says that given the input sequence in some order (ascending or descending in this example we take ascending order) we must be able to find whether the given element is in the set or not. The algorithm for binary search problem is given below:

Algorithm:

```
binsearch(key // this is the value to be searched, list[] // this is the list of the elements in ascending order)
{
left = 0;
right = n // here n is the size of the array list
while left < right
{
mid = floor((left + right)/2);
if(key > list[mid]
        left = mid + 1;
else
        right = mid;
}
if( key = = list[left])
        return left;
else
        return –1;
}
```

Analysis: in the above algorithm the statements left = 0 and right = n both take a unit time if (key > list[left] … else return –1 takes 2 steps (one condition check and next statement run). So the portion of while will dominate the running time. If the number of iterations of while loop is obtained then our problem is solved. We can observe that inside the while loop each time the problem is halved (more or less). So if we consider the size of the input n be the power of 2 for simplicity i.e. say $2^k = n$. so at he first execution of loop search is to be done for the size of 2^{k-1}, at the second iteration search space is reduced to 2^{k-2}…. Until the search space has only one element this violates the condition left < right (verify!!!). so we can write the above steps as

T(n) = 1 + 1 + k(steps inside the loop) + 2

=k(3) + 4 = 3k + 4 = O(k) (how?)

Since we have assumed $2^k = n$, k = logn

so the complexity is O(logn). If n is not a power of 2 then we can have some numbers which is power of 2 and greater than n so that the asymptotic behavior is same since the number of operations only differ by some constant.

Some commonly used functions in Algorithms Complexity

f(x) = O(1)	constant
f(x) = C*log x	logarithmic
f(x) = C*x	linear
f(x) = C*x log x	linearithmic
$f(x) = C*x^2$	quadratic
$f(x) = C* x^3$	cubic
$f(x) = C* x^k$	polynomial in k
$1.f(x) = C* k^x$	exponential in k

Order of growth

$O(1) < C*log\ x < C*x < C*x\ log\ x < C*x^2 < C* x^3 < C* x^k (k > 3) < C* k^x$ (k > 1)

Value of k should be in increasing order.

Some Definitions

Tractable: A problem that is solvable by algorithm with polynomial worst-case complexity.

Intractable: A problem that cannot be solved using worst-case polynomial time algorithm. Many problems in use are thought to be intractable but due to the simple nature of the input instances most of the time the average case behavior of an algorithm is used. Some time instead of exact solution approximate solutions are considered.

Unsolvable: The problem with no existence of algorithms to solve them. For e.g. halting problem proposed by Alan Turing.

Class P: Tractable problems.

Class NP: the problems for which the solution can be checked in polynomial time (The solution must exist already).

NP-complete problems: this is the class of problems where if any one of the problem is solved in worst case polynomial time then all other of that class can be solved in worst case polynomial time.

Divisibility

If a and b are integers where $a \neq 0$, we say a divides b if there is an integer c such that $b = ac$. When a divides b then we say a is a factor of b and the b is a multiple of a. notational representation $a|b$ is for a divides b. for e.g. $4|12$ means 4 divides 12 where $a = 4$, $b = 12$ and $c = 3$.

Theorem 1:

Let a,b, and c be integers. Then

1.if $a|b$ and $a|c$, then $a|(b+c)$;

2.if $a|b$, then $a|bc$ for all integers c;

3.if $a|b$ and $b|c$, then $a|c$.

Proof:

1.given that a|b and a|c, so by the definition of divisibility we can say that there are integers p and q such that b = ap and c = aq. From this we can write,

b+c = ap + aq

i.e. b+c = a(p+q)

So form this we can say that a divides b+c.

2.given that a|b, by the definition of divisibility we can say there is an integer p such that b = ap so for any integer c we can write,

bc = apc this means a divides bc since pc is an integer too.

3.given that a|b and b|c, by the definition of divisibility we have integers p and q such that b = ap and c = bq

i.e c = apq

Since pq is an integer we conclude that a divides c.

Proved.

Lemma 1: If a, b, and c are positive integers such that gcd(a,b) = 1 and a|bc, then a|c.

Proof:

gcd(a,b) can be written as linear combination with integer coefficient of a and b as sa + tb (this is a theorem). Since gcd(a,b) = 1, we have sa + tb =1.

Multiplying both side by c,

sac + tbc = c.

From theorem 1 (2) above a|tbc since we have a|bc. We have a|sac (trivial). Since a|sac and a|tbc, we have a|c fro theorem 1 (1).

Proved.

Primes

A positive integer grater than 1 and divisible by only 1 or itself is called prime. If the positive integer is not a prime then it is a composite number. For e.g. 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, and 37.

Lemma 2:

If p is a prime and $p|a_1, a_2, \ldots, a_n$ where each a_i is an integer, then $p|a_i$, for some i.

Proof:

Basis Step: for n =1, we have $p|a_1$ so $p|a_i$ is true for i =1 (this is trivial case).

Induction Hypothesis: Assume that lemma is true for n.

Inductive Step: for n+1 we have $p|a_1, a_2, \ldots, a_n, a_{n+1}$.we have $\gcd(p, a_i) = 1$ or p for all i = 1, 2, …., n. if $\gcd(p, a_i) = 1$, then from the lemma 1 above $p|a_{n+1}$. If $\gcd(p, a_i) = p$, then $p|a_1, a_2, \ldots, a_n$. (this is induction hypothesis) so $p|a_i$, for some $i \leq n$. Hence the proof.

Lemma 3:

Prime factorization of a positive integer in nondecreasing order of primes is unique.

Proof:

Suppose that n, a positive integer, can be written as product of prime numbers in two different ways $n = p_1, p_2, \ldots, p_r$ and $n = q_1, q_2, \ldots, q_s$, each p_i and p_j are primes such that $p_1 \leq p_2 \leq \ldots \leq p_r$ and $n = q_1 \leq q_2 \leq \ldots \leq q_s$.

Removing all the common primes from the two factorization we have

$p_{i1}, p_{i2}, \ldots, p_{iu} = q_{j1}, q_{j2}, \ldots, q_{jv}$,

Here no prime occurs on both sides and u, v are positive integers. From the lemma 2 $p_{i1}|q_{jk}$, for some k. But this is impossible since no prime divides another prime. Hence there can be at most one factorization of n into primes in nondecreasing order.

Lemma 4:

Every positive integer can be written as the product of primes. (Here, a product can have zero, one, or more than one prime factor.)

Proof:

Let P(n) be the proposition that positive integer can be written as the products of primes.

Basis Step: p(1) is true since 1 is the product of no prime. Similarly P(2) is true, since 2 can be written as the product of one prime i.e. 2 itself.

Induction Hypothesis: Assume that P(k) is true for all positive integers k where $k \leq n$.

Inductive Step: if we can prove that p(n+1) is true then our proof is done. Here we have two cases. If n+1 is a prime number then it can be written as product of prime number and that is itself. If n+1 is a composite number then we can write it as product of two positive integers. Let those positive integers be p and q where $2 \leq p \leq q < n+1$, then from the induction hypothesis both a and b has prime factorization. Hence n + 1 has prime factorization that is combination of prime factorization of a and prime factorization of b. This proofs that every integer has a prime factorization.

Theorem 2: The Fundamental Theorem of Arithmetic

Every positive integer can be written uniquely as the product of primes, where the prime factors are written in order of increasing size. (Here, a product can have zero, one, or more than one prime factor.)

Proof:

From lemma 3 and lemma 4 above, we complete the proof.

Division Algorithm

Let a be an integer and d a positive integer. Then there are unique integers q and r, with $0 \leq r < d$, such that $a = dq + r$. here a is called dividend, d is called divisor, q is called quotient, and r is called remainder. For e.g. 305(dividend) = 10(divisor)*30(quotient) + 5(remainder).

GCD and LCM

Let a and b be integers, not both zero. The largest integer d such that d|a and d|b is called the **greatest common divisor (gcd)** of a and b. we denote gcd of a and b by gcd(a,b). a

and b are relatively prime if gcd(a,b) = 1 for e.g. 3 and 5 are relatively prime (how?). Similarly the integers a_1, a_2, …, a_n are pairwise relatively prime if gcd(a_i, a_j) = 1 whenever $1 \leq i < j \leq n$. The **least common multiple (lcm)** of the positive integers a and b is the smallest positive integer that is divisible by both a and b. It is denoted by lcm(a,b).

Modular Arithmetic

If a is an integer and m is a positive integer then a **mod** m is the remainder when a is divided by m. so from the definition of the remainder that a **mod** m is the integer r such that a = qm + r and $0 \leq r < m$. for e.g. 24 mod 5 = 4 since 24 = 4*5 + 4.

If a and b are integers and m is a positive integer, then a is congruent to b modulo m if m divides a-b. We use the notation $a \equiv b \pmod{m}$ to indicate that a is congruent to b modulo m. for e.g. 23 is congruent to 11 modulo 2 since 23 –11 = 12 is divisible by 2.

[See the applications of congruences on hashing, pseudorandom numbers and cryptology.]

Euclidean Algorithm

Finding gcd(a,b) using the prime factorization is inefficient. The algorithm called Euclidean Algorithm is an algorithm that finds the gcd(a,b) in efficient manner. The successive division is used to reduce the problem if finding gcd(a,b) to the same problem of smaller integers, until one of the integers is zero. This algorithm used the result from the following lemma.

Lemma 5: Let a = bq + r, where a, b, q, and r are integers. Then gcd(a,b) = gcd(b,r)

Proof:

For the expression gcd(a,b) = gcd(b,r) to be true the greatest common divisor of a,b and b.r must be same. Let d be the divisor of both a and b then we can say that a – bq =r is divided by d (from theorem 1 above). Hence any common divisor of a and b are common divisor of b and r. Similarly if d is the divisor of b and r then d also divides bq + r =a (how?). Hence, any common divisor of b and r divides a and b also. Hence, the proof.

Algorithm:

```
gcd(a,b) // a and b are positive integers
{
x = a;
y = b;
while (y!=0)
{
r = x mod y;
x = y;
y = r;
}
}
```

Analysis:

In the later chapter we will see the Lame's Theorem, this results the complexity of above algorithm as O(log b) (this is actually based on number of division) assuming a ≥ b.

Matrix Multiplication

In this section just the algorithm and its complexity is presented for the matrix multiplication problem.

Algorithm:

Input(s): two matrices A and B, where A is m × p and B is p × n.

Output: the matrix C = AB such that C is m × n.

```
matmul(A,B){
for(i = 0; i<m; i++)
        for(j = 0; j<n; j++){
                C[i][j] = 0;
                for(k = 0; k<p; k++)
                        C[i][j] = C[i][j] + A[i][k]B[k][j];
}}
```

Analysis:

The above algorithms clearly gives its complexity as O(mnp). For simplicity lets take matrices A and B is of size q × q then (m = n = p) = q so, complexity of an above algorithm is $O(q^3)$.

Self Study

Read Chapter 2 all from the book (you may omit the section [2.5 from (4^{th} edition) or 2.6 (5^{th} edition)] but it will be good to see it also since it covers something about cryptography).

[Mathematical Reasoning]

Discrete Structures (CSc 511)

Samujjwal Bhandari

Central Department of Computer Science and Information Technology (CDCSIT)

Tribhuvan University, Kirtipur,

Kathmandu, Nepal.

Mathematical Reasoning

Any of the mathematical statement must be supported by arguments that make it correct. For this we need to know different techniques and rules that can be applied in the mathematical statements such that we can prove the correctness of the given mathematical statement. This method of understanding the correctness by sequence of statements forming an argument is a proof of the statement. A theorem is a mathematical statement that can be shown to be true. Well founded proof is the steps of mathematical statements that present an argument that makes the theorem true. By proving some mathematical problem we mean that we solve that problem. For this purpose valid steps are required such that as mentioned above those steps aid on solving the problems. Problem solving or proving is not just a science so there is no hard and fast rule that is applied in problem solving. However there are some guiding methods that help us to solve different kinds of problems. Here still we must note that the problem solving is not just a science, so hard work and art is needed.

Rules of Inference

To draw conclusion from the given premise we must be able to apply some well defined steps that helps reaching the conclusion. These steps of reaching the conclusion are provided by the rules of inference. Here some of the rules of inferences are given below:

Rule 1: Modus Ponens (or Law of Detachment)

Whenever two propositions p and $p \rightarrow q$ are both true then we confirm that q is true. We write this rule as

$$\frac{\begin{array}{l} p \\ p \rightarrow q \end{array}}{\therefore q}$$, This rule is valid rule of inference because the implication $[p \wedge (p \rightarrow q)] \rightarrow q$ is a tautology.

Example:

Ram is hard working and if Ram is hard working, then he is intelligent. By modus ponens

(verify!!!), this logically infers Ram is intelligent.

Rule 2: Hypothetical Syllogism (Transitive Rule)

Whenever two propositions p→ q and q → r are both true then we confirm that implication p → q is true. We write this rule as

$$\frac{\begin{array}{l} p \to q \\ q \to r \end{array}}{\therefore\ p \to r}$$, This rule is valid rule of inference because the implication [(p→ q) ∧ (q→ r)] → (p→ r) is a tautology.

This rule can be extended to larger numbers of implications as

$$\frac{\begin{array}{l} p \to q \\ q \to r \\ r \to s \end{array}}{\therefore\ p \to s}$$

Example:

If today is Sunday, then today is rainy day and if today is rainy day, then it is wet today. By transitivity rule (verify!!!), this logically infers It is wet today.

In similar fashion we can define the following rules.

Rule 3: Addition

Due to the tautology p→ (p ∨ q), rule $\frac{p}{\therefore\ p \vee q}$ is a valid rule of inference.

Rule 4: Simplification

Due to the tautology (p ∧ q) → p, rule $\frac{p \wedge q}{\therefore\ p}$ is a valid rule of inference.

Rule 5: Conjunction

Due to the tautology [(p) ∧ (q)]→ (p ∧ q), rule $\frac{\begin{array}{l} p \\ q \end{array}}{\therefore\ p \wedge q}$ is a valid rule of inference.

Rule 6: Modes Tollens

Due to the tautology $[\neg q \wedge (p \rightarrow q)] \rightarrow \neg p$, rule $\dfrac{\begin{array}{l}\neg q \\ p \rightarrow q\end{array}}{\therefore \neg p}$ is a valid rule of inference.

Rule 7: Disjunctive Syllogism

Due to the tautology $[(p \vee q) \wedge \neg p] \rightarrow q$, rule $\dfrac{\begin{array}{l}p \vee q \\ \neg p\end{array}}{\therefore q}$ is a valid rule of inference.

Rule 8: Constructive Dilemma

Due to the tautology $[(p \rightarrow q) \wedge (r \rightarrow s) \wedge (p \vee r)] \rightarrow (q \vee s)$, rule $\dfrac{\begin{array}{l}(p \rightarrow q) \wedge (r \rightarrow s) \\ p \vee r\end{array}}{\therefore q \vee s}$ is a valid rule of inference.

Rule 9: Destructive Dilemma

Due to the tautology $[(p \rightarrow q) \wedge (r \rightarrow s) \wedge (\neg q \vee \neg s)] \rightarrow (\neg p \vee \neg r)$, rule $\dfrac{\begin{array}{l}(p \rightarrow q) \wedge (r \rightarrow s) \\ \neg p \vee \neg s\end{array}}{\therefore \neg p \vee \neg r}$ is a valid rule of inference.

Rule 10: Resolution

Due to the tautology $[(p \vee q) \wedge (\neg p \vee r)] \rightarrow (q \vee r)$, rule $\dfrac{\begin{array}{l}p \vee q \\ \neg p \vee r\end{array}}{\therefore q \vee r}$ is a valid rule of inference.

Valid Arguments

An argument is called valid if all hypotheses are true and the conclusion is also true. We can conclude that the implication $(p_1 \wedge p_2 \wedge \ldots \wedge p_n) \rightarrow q$ is tautology. If all the propositions in the valid argument are true then the conclusion is true.

Sometime valid argument can lead to incorrect conclusion if one or more of the false premises are used in the argument. For e.g. If CDCSIT is at Kirtipur, then Lagankhel is at Kirtipur. CDCSIT is at Kirtipur. Consequently, Lagankhel is at Kirtipur.

The above argument is a valid argument using the rule modus ponens. However the conclusion of the argument is false since the proposition at the hypothesis "Lagankhel is at Kirtipur" is false that means conclusion may be false here.

Example 1: Construct an argument using rules of inference to show that the hypotheses "If it does not rain or if it is not foggy, then the sailing race will be held and the life saving demonstration will go on," " If the sailing race is held, then the trophy will be awarded,' and "The trophy was not awarded" imply the conclusion " It rained".

Solution:

Let p = "It rains", q = "It is foggy", r = "the sailing race is held", s = "Life saving demonstration is done", and t = " Trophy is awarded".

Then we have to show that the argument

$[((\neg p \vee \neg q) \rightarrow (r \wedge s)) \wedge (r \rightarrow t) \wedge \neg t] \rightarrow p$ is valid.

[1]	$(r \rightarrow t)$	[Hypothesis]
[2]	$\neg t$	[Hypothesis]
[3]	$\neg r$	[Modus Tollens using steps 1 and 2]
[4]	$((\neg p \vee \neg q) \rightarrow (r \wedge s))$	[Hypothesis]
[5]	$\neg(\neg p \vee \neg q) \vee (r \wedge s)$	[Implication of Step 4]
[6]	$(p \wedge q) \vee (r \wedge s)$	[De Morgan's Law in Step 5]
[7]	$p \vee (r \wedge s)$	[Simplification using step 6]
[8]	$p \vee r$	[Simplification using step 7]

Here our original premises changes to $(p \vee r) \wedge \neg r$ [from step 8 and 3]

[9]	$r \vee p$	[Commutative law in step 8]
[10]	$\neg r \vee p$	[Addition using step 3]
[11]	$p \vee p$	[Resolution using steps 9 and 10]
[12]	p	[Idempotent law]

Hence argument is valid. With conclusion "It rained".

Example 2: For the set of premises "If I play hockey, then I am sore the next day." "I use the whirlpool if I am sore." " I did not use the whirlpool". What relevant conclusion can be drawn? Explain the rules of inference used to draw the conclusion.

Solution:

Let p = "I play hockey", q = " I am sore", r = "I use the whirlpool"

Then the above premises are

a) $p \rightarrow q$

b) $q \rightarrow r$

c) $\neg r$

Using hypothetical syllogism in premises a and b we have $p \rightarrow r$ i.e. "if I play hockey, then I use whirlpool"

Using the modus tollens in premise c and inferred proposition $p \rightarrow r$ we conclude $\neg p$ is true i.e. p is false. p is false means " I did not play hockey".

Fallacies

The fallacies are arguments that are convincing but not correct. So fallacies produce faulty inferences. So fallacies are contingencies rather than tautologies. Here we talk different fallacies that we may encounter.

Fallacy of affirming the conclusion (consequence)

This kind of fallacy has the form $\begin{array}{l} q \\ p \rightarrow q \\ \hline \therefore p \end{array}$ i.e. $p \wedge (p \rightarrow q) \rightarrow q$. This is not a tautology hence it is a fallacy.

Example:

If economy of Nepal is poor, then the education system in Nepal will be poor. The education system in Nepal is poor. Therefore, Economy of Nepal is poor.

In this argument above the conclusion can be false even if both the propositions "If economy of Nepal is poor, then the education system in Nepal will be poor" and "The education system in Nepal is poor" are true. Denoting with symbols we may write $(p \rightarrow q)$

for first proposition and then the second proposition becomes q. this takes the form $q \wedge (p \rightarrow q) \rightarrow q$, which is not a tautology. Since the education system may not depend on the economy of the country.

Fallacy of denying the hypothesis

This kind of fallacy has the form $\frac{p \rightarrow q}{\therefore \neg q}$ i.e. $\neg p \wedge (p \rightarrow q) \rightarrow \neg q$. This is not a tautology hence it is a fallacy.

Example:

If today is Sunday, then it rains today. Today is not Sunday. Therefore, it does not rain today. This argument is not true since even if today is not Sunday and it is raining today then the first premise is true and second premise is also true but not the conclusion.

The non sequitur fallacy

Non sequitur mean "does not follow". Generally all logical errors are the cases of non sequitur fallacy. For e.g. $\frac{p}{\therefore q}$, if p is true and q is false then what happens?

Example:

I am a teacher therefore Ram is a doctor. (how is this valid? No, it is not i.e. if Ram is not a doctor then what?).

Begging the Question (Circular Reasoning)

If the statement that is used for proof is equivalent to the statement that is being proved then it is called circular reasoning.

Example:

The square root of 2 is irrational since it is not rational.

Man is mortal because man dies.

Ram is black because he is black.

Rules of Inference for Quantified Statements

There is a need of other rules to prove assertions that contain open propositions and quantifiers. Some of the rules are:

Universal Instantiation

If the proposition of the form ∀xP(x) is supposed to be true then the universal quantifier can be dropped out to get P(c) is true for arbitrary c in the universe of discourse. This can be written as

$$\frac{\forall x P(x)}{\therefore P(c), \text{ for all } c}$$

Example:

In universe of discourse of all man every man is mortal implies ram is mortal where ram is a man.

Universal Generalization

If all the instances of c makes P(c) true, then ∀xP(x) is true. This can be written as

$\frac{P(c), \text{ for all } c}{\therefore \forall x P(x)}$, Here the chosen c must be arbitrary, not a specific element from the universe of discourse. This rule is seldom explicitly used.

Existential Instantiation

If the proposition of the form ∃xP(x) is supposed to be true then the there is an element c in the universe of discourse such that P(c) is true. This can be written as

$\frac{\exists x P(x)}{\therefore P(c), \text{ for some } c}$, Here the element c is not arbitrary, it must be specific such that P(x) is true. We generally find difficulty in finding such c.

Existential Generalization

If at least a element c from the universe of discourse makes P(c) true, then ∃xP(x) is true. This can be written as $\frac{P(c), \text{ for some } c}{\therefore \exists x P(x)}$

Inference with quantified statements

Example 1:

Explain which rules of inference are used for the argument "Linda, a student in the class, owns a red convertible. Everyone who owns a red convertible has gotten at least one speeding ticket. Therefore, someone in this class has gotten a speeding ticket."

Solution:

Let S(x) denotes x is a student in a class, R(x) denotes x owns red convertible and T(x,y) denotes x has gotten y numbers of speeding tickets. Where x is a set of people, Then

S(Linda) , R(Linda), $\forall x(R(x) \rightarrow \exists yT(x,y))$ are the premises and $\exists x(S(x) \wedge T(x,1))$ is the conclusion.

$R(Linda) \rightarrow \exists yT(Linda,y)$ is true using universal instantiation. Since R(Linda) is true using modes ponens $\exists yT(Linda,y)$ is true. The number 1 is the least number of tickets that can be there. So using existential instantiation T(Linda,1). Since both S(Linda) and T(Linda,1) are true by using conjunction $S(Linda) \wedge T(Linda,1)$. Hence By using existential generalization $\exists x(S(x) \wedge T(x,1))$ is true.

Example 2:

Prove or disprove the validity of the argument " every living thing is a plant or an animal", "Hari's dog is alive and it is not a plant", "All animals have heart", Hence "Hari's dog has a heart".

Solution:

Let P(x) be x is a plant, A(x) be x is an animal, L(x) be x is alive, H(x) be x has heart and d be Hari's dog.

[1]	$\forall x(P(x) \vee A(x))$	[Hypothesis]
[2]	$L(d) \wedge \neg P(d)$	[Hypothesis]
[3]	$\forall x(A(x) \rightarrow H(x))$	[Hypothesis]
[4]	$P(d) \vee A(d)$	[Universal instantiation from 1]
[5]	$\neg P(d)$	[from 2 Simplification]
[6]	A(d)	[Disjunctive Syllogism form 4 and 5]

[7] $A(d) \rightarrow H(d)$ [Universal instantiation from 3]

[8] $H(d)$ [modus ponens from 6 and 7]

Hence Hari's dog has a heart, so the above argument is valid.

Proving Theorems

In this part we see Methods of Proof of an Implication. We present different methods here but it is not true that all the methods of proof are given here.

Direct Proofs

We prove the implication $p \rightarrow q$, where we start assuming that the hypothesis i.e. p is true and using information already available (rules of inferences, theorems, etc.), if q becomes true, then the argument becomes valid. This is direct proof.

Example:

If a and b are odd integers, then a + b is an even integer.

Proof:

We know the fact that if a number is even then we can represent it as 2k, where k is an integer and if the number is odd then it can be written as 2l + 1, where l is an integer. Assume that a = 2k + 1 and b = 2l + 1, for some integers k and m. then a + b = 2k + 1 + 2l + 1 = 2(k + l + 1), here (k + l + 1) is an integer. Hence a + b is even integer.

Indirect Proofs

We have $p \rightarrow q \equiv \neg q \rightarrow \neg p$ i.e. contrapositive of implication is equivalent to the implication. This is the base for indirect proof. We prove the implication $p \rightarrow q$ by assuming that the conclusion is false and using the known facts we show that the hypothesis is also false.

Example:

If the product of two integers a and b is even, then either a is even or b is even.

Proof:

Suppose both a and b are odd, then we have a = 2k + 1 and b = 2l + 1.

So ab = (2k +1)(2l + 1) = 4kl +2k +2l +1 = 2(2kl + k + l) + 1, i.e. ab is an odd number. Hence both a and b being odd implies ab is also odd. This is indirect proof.

Trivial and Vacuous Proofs

If it is possible to show that q is correct regardless of truth value of p then we can say that implication $p \rightarrow q$ is true. This is **trivial proof**. If we can show that p is false then the implication $p \rightarrow q$ is true. This is **vacuous proof**.

Example:

If x is an integer, then 3 is an odd integer. (Trivial)

If a black is white, then pink is blue. (Vacuous)

Proofs by Contradiction

The steps in proof of implication $p \rightarrow q$ by contradiction are:

Assume $p \wedge \neg q$ is true.

Try to so that the above assumption is false

When the assumption is found to be false then implication $p \rightarrow q$ is true since $p \rightarrow q$ is equivalent to $\neg p \vee q$ and negation of $\neg p \vee q$ is $p \wedge \neg q$ (By De Morgan's Law), so if our assumption is false then its negation is true.

Alternately,

Contradict the statement and show that this leads to the false conclusion; if this is true then the contradicted statement must be false (since $\neg p \rightarrow \mathbf{F}$ is true only if $\neg p$ is false), hence the statement is true.

Example:

If a^2 is an even number, then a is an even number.

Proof:

Assume that a^2 is an even number and a is an odd number. Since a is an odd number we have $a = 2k + 1$, for some integer k. so $a^2 = (2k + 1)^2 = 4k^2 + 4k + 1 = 2(k^2 + k) + 1$, here $k^2 + k$ is some integer, say l, then $a^2 = 2l + 1$ i.e. a^2 is an odd number This contradicts our assumption that is a^2 even. Hence the proof.

Proof by Cases

The implication of the form $(p_1 \vee p_2 \vee \ldots \vee p_n) \rightarrow q$ can be prove by using the tautology $(p_1 \vee p_2 \vee \ldots \vee p_n) \rightarrow q \leftrightarrow [(p_1 \rightarrow q) \wedge (p_2 \rightarrow q) \wedge \ldots \wedge (p_n \rightarrow q)]$, i.e. we can show every implication $(p_i \rightarrow q)$ true for i =1, 2, …, n.

Example:

If $|x| > 3$, then $x^2 > 9$, where x is a real number.

Proof:

Here we have to consider two cases -x >3 and x > 3 since |x|, is an absolute value of x, is x when $x \geq 0$ and –x when $x \leq 0$. If $–x > 3$, then $x^2 > 9$. Similarly, if $x > 3$, then $x^2 > 9$.

Proof of Equivalence

We can prove the equivalence i.e. $p \leftrightarrow q$ by showing $p \rightarrow q$ and $q \rightarrow p$ both.

Example:

Prove that if n is a positive integer, then n is even if and only if 7n + 4 is even.

Proof:

Assume n is even then we have an integer k such that n = 2k, so 7n + 4 = 7*2k +4 = 2(7k +2) here 7k + 4 is an integer so that 7n + 4 = 2l, where l =7k + 2 i.e. 7n + 4 is even. By direct proof it is proved that if n is even, then 7n + 4 is even.

Assume n is odd then we have an integer m such that n = 2m + 1, then 7n + 4 = 14m +7 + 4 = 2(7m + 5) +1here since 7m + 5 is an integer 7n + 4 is an odd number by indirect proof it is proved that if 7n + 4 is even, then n is even.

Hence the proof.

Existence Proofs

A proof of a proposition of the form $\exists x P(x)$ is called an existence proof. There are different ways of proving a theorem of this type. Sometime some element a is found to show P(a) to be true, this is called **constructive existence proof**. In other method we do not provide a such that P(a) is true but prove that $\exists x P(x)$ is true in different way, this is called **nonconstructive existence proof**.

Example: Constructive

Prove that there are 100 consecutive positive integers that are not perfect squares.

Proof:

Lets consider 2500 this is a perfect square of 50, and take 2601 this is a perfect square of 51. in between 2601 and 2500 there are 100 consecutive positive integers. Hence the proof.

Example: Nonconstructive

Prove that there is a rational number x and an irrational number y such that x^y is irrational.

Proof:

Lets take x = 2 and y = $\sqrt{2}$ then $2^{\sqrt{2}}$ is either rational or irrational. If it is irrational we are done, if it is not irrational then it is rational. So take x = $2^{\sqrt{2}}$ and y = $\sqrt{2}/4$ then we have $(2^{\sqrt{2}})^{\sqrt{2}/4} = 2^{2/4} = \sqrt{2}$ (irrational). Hence there is a rational number x and irrational number y such that x^y is irrational.

Uniqueness Proofs

To prove the theorem that asserts the existence of unique element with particular property we must show that the element with this property exists and no other elements has this property. There are two parts in this uniqueness proof

Existence: here we show that the element with desire property exists

Uniqueness: we show that if $y \neq x$, then y does not have the desired property.

The above two steps can be proved if we prove the statement

$\exists x(P(x) \wedge \forall y(y \neq x \rightarrow \neg P(y)))$.

Example:

Show that if a, b, and c are real numbers and $a \neq 0$, then there is a unique solution of the equation ax + b = c.

Proof:

From the equation ax + b = c we get the solution as x = (c – b)/a (since $a \neq 0$, it is possible). This solution is unique because there is no other value for x than (c – b)/a (a real number).

Proofs By Counter Examples

To prove that the statement of the form $\forall x P(x)$ is false, we just need some value of x. So while proving for falsity we just look for counter example.

Example:

Prove or disprove the product of two irrational numbers is irrational.

Proof:

Here we instantly try to get the product of the irrational to try it. Lets take both the number for product be $\sqrt{2}$ then we have $\sqrt{2}*\sqrt{2} = 2$ (not rational). Hence by counter example it is shown that the product of two irrational numbers is not necessarily irrational.

Proof Strategies

As mentioned before finding proof is not just a science but an art too, there are no exact rules and strategies for proving the statement. Few strategies that will be very helpful in proving the statement are presented here.

Forward and Backward Reasoning

Remember the proof strategy utilized by direct proof method. In this method we prove the implication $p \rightarrow q$ starting from p and using known theorems and axioms we come out with q. This type of reasoning is called forward reasoning. Sometimes it is difficult to prove in above way, particularly when the conclusion is complex one. To prove such statements we find p with the help of property that $p \rightarrow q$. this is called backward reasoning or find some property such that q is true, then show that we can come up to the property using p working from the property itself.

Example: Backward Reasoning

For integer a, b, c, d and positive integer n, prove that if $a \equiv b \pmod{n}$, and $c \equiv d \pmod{n}$, then $a + c \equiv b + d \pmod{n}$.

Proof:

We can prove $a + c \equiv b + d \pmod{n}$ if we can show that $(a + c) – (b + d) = n.k$, for some integer k (recall definition of congruence modulo).but $(a + c) – (b + d) = (a - b) + (c - d)$.

We know a – b = n.l, and c –d =n.m, for some integers l and m (see hypothesis). So we can see that (a + c) – (b + d) = n.l + n.m = n(l + m). Here l +m is also an integer, say k, then we have (a + c) – (b + d) = n.k. we have shown that for some integer k, (a + c) – (b + d) = n.k, hence a + c ≡ b + d (modn).

Using Proof by Cases

When there is no clear way to begin proof but you can sense that the information from different cases moves forward to the proof, you generally use proof by cases (see above on the section proofs by cases for example).

Techniques from Existing Proofs

It is generally very easy to prove if existing proofs results or ideas from those proofs are applied. You will see a lot of use of this strategy through out the course.

Using Counter Examples

Given a conjecture if you think it is to be wrong then you can just give the example that defy the statement given.

Mathematical Induction

In mathematics there are two ways of arriving at result deductive and inductive. In deductive reasoning based upon the assumption that some statements are premises and axioms, we deduce the other statements on the basis of valid inference. In the inductive reasoning through the experiments and observations we come up with the conjecture for a general rule and try to verify truth of the conjecture. One of the important reasoning that considers positive integers is mathematical induction.

Principle of Mathematical Induction

Let P(n) be a statement that may be true or false for all positive integers n. To prove P(n) is true for all n ≥ 1 we can prove the following steps.

P(1) is true.

For all $k \geq 1$, P(k) implies P(k+1).

Generalizing the above proof method instead of 1 take n_0 such that the n_0 is the basis for induction then we have the steps to be proved are:

Basis Step: Show $P(n_0)$ is true.

Inductive Hypothesis: Assume P(k) is true for k = n.

Inductive Step: Show that the P(k+1) is true on the basis of Inductive Hypothesis.

Expressing in terms of rule of inference, this proof technique can be written as

$[P(n_0) \wedge \forall k(P(k) \rightarrow P(k+1))] \rightarrow \forall n P(n)$.

Example 1:

Prove that $2 - 2.7 + 2.7^2 - \ldots + 2(-7)^n = (1 - (-7)^{n+1})/4$ whenever n is a nonnegative integer.

Proof:

Let P(n) be $2\sum_{i=0}^{n}(-7)^i = (1-(-7)^{n+1})/4$, then

Basis Step: $2.(-7)^0 = 2$ and

$(1 - (-7)^{0+1})/4 = (1+7)/4 = 2$,

so P(0) is true.

Inductive Hypothesis: Assume that P(n) is true.

Inductive Step: if P(n+1) is true then prove is done. So P(n+1) is $2\sum_{i=0}^{n+1}(-7)^i =$

$2\sum_{i=0}^{n}(-7)^i + 2.(-7)^{n+1}$ so Using the assumption from the induction hypothesis we have

$P(n+1) = (1 - (-7)^{n+1})/4 + 2(-7)^{n+1}$

$= (1 - (-7)^{n+1} + 8(-7)^{n+1})/4$

$= (1 + 7(-7)^{n+1})/4$

$= (1 - (-7)^{n+2})/4$.

Hence, P(n) is true for all nonnegative integers.

Example 2:

Prove that $1.1! + 2.2! + \ldots + n.n! = (n+1)! - 1$, whenever n is a positive integer.

Proof:

Let $P(n) = 1.1! + 2.2! + \ldots + n.n! = (n+1)! - 1$, then

Basis Step: for n = 1, we have $P(1) = 1.1! = 1$, Similarly $P(1) = (1+1)! - 1 = 2-1 = 1$

Hence P(1) is true.

Inductive Hypothesis: Assume that P(n) is true, i.e. $1.1! + 2.2! + \ldots + n.n! = (n+1)! - 1$.

Inductive Step: if we are able to prove that P(n+1) is true then we are done. So we have

$P(n+1) = 1.1! + 2.2! + \ldots + n.n! + (n+1)(n+1)!$

$= (n+1)! - 1 + (n+1)(n+1)!$ (using induction hypothesis)

$= (n+1)n! + (n+1)(n+1)! - 1 \quad = (n+1)(n! + (n+1)!) - 1$

$= (n+1)(n!\,(1+ (n+1)) - 1 = (n+1)n!\,(n+2) - 1$

$= (n + 2)! - 1$

P(n+1) is true

Hence P(n) is true for all positive integers.

Strong Induction (Second Principle of Mathematical Induction)

This method uses different inductive step than the first principle. Here we assume that P(k) is true for $k = n_0, n_0 + 1, \ldots, k$ and show that P(k+1) is true based on the assumption. The steps in this method are:

Basis Step: Show $P(n_0)$ is true.

Inductive Hypothesis (Strong): Assume P(k) is true for all $n_0 \le k \le n$.

Inductive Step: Show based on the assumption that P(k+1) is true.

Example 1:

Prove that 3 divides $n^3 + 2n$ whenever n is a nonnegative integer.

Proof:

Let $P(n) = n^3 + 2n$, then

Basis Step: For n = 0, we have $n^3 + 2n = 0$, this is divisible by 3 hence the statement is true for n = 0.

Inductive Hypothesis: assume that the $P(k) = k^3 + 2k$ is divisible by 3 for all nonnegative values for $k \leq n$.

Inductive Step: here we are going to show that p(k+1) true. We have

$P(k+1) = (k+1)^3 + 2(k+1) = k^3 + 3k^2 + 3k + 1 + 2k + 2$

$= k^3 + 2k + 3k^2 + 3k + 3$

$= 3l + 3k^2 + 3k + 3$ (since $k^3 + 2k$ is divisible by 3)

$= 3(l + k^2 + k + 1)$

Since both l and k are positive integers $(l + k^2 + k + 1)$ is also positive integer. Hence, P(k+1) is divisible by 3.

So by mathematical induction $n^3 + 2n$ is divisible by three for all nonnegative integers n.

Example 2:

Use mathematical induction to show that $1/(2n) \leq [1.3.5\ldots..(2n-1)]/(2.4\ldots..2n)$ whenever n is a positive integer.

Proof:

Let P(n) be $1/(2n) \leq [1.3.5\ldots..(2n-1)]/(2.4\ldots..2n)$

Basis Step: for n =1, we have $1/2n = 1 = [1.3.5\ldots..(2n-1)]/(2.4\ldots..2n)$, Since $1 \leq 1$, P(1) is true.

Inductive Hypothesis: Assume that P(k) is true for all positive $k \leq n$.

Inductive Step: Now to prove P(k+1) is true we have to show

$1/(2(k+1)) \leq [1.3.5\ldots..(2k-1)(2k+1)]/(2.4\ldots..2k.2(k+1))$ so we have,

$1/(2k) \cdot (2k+1)/(2(k+1)) \leq [1.3.5\ldots..(2k-1)(2k+1)]/(2.4\ldots..2k.2(k+1))$

[Above relation is true from inductive hypothesis]

$1/(2k).(2k+1)/(2(k+1))$

$= (2k+1)/(2k).1/(2(k+1))$

$= (1 + 1/(2n))1/(2(k+1))$

$= 1/(2(k+1)) + 1/(2(k+1))(2n)$

Here we have,

$1/(2(k+1)) \leq 1/(2(k+1)) + 1/(2(k+1))(2n)$

$\leq [1.3.5\ldots..(2k-1)(2k+1)]/(2.4\ldots..2k.2(k+1))$, hence proved.

Well Ordering Property

This property states, "Every nonempty set of nonnegative integers has a least element." Using this property we can verify the validity of proofs using mathematical induction.

Using mathematical induction we prove P(1) is true and P(n) → P(n+1) is true for all positive integers n. If the proof by mathematical induction is not valid then P(n) is true for all positive integers n would be false. Let the set of positive integers for which P(n) is false be T. then T is nonempty since there is at least one element in T such that P(n) is false. By the well ordering property, T has a least element, let the least element be k. we know that m cannot be 1 because we have already proved that P(1) is true. So k is a positive integer greater than 1 so k –1 is a positive integer, so we have P(k-1) must be true. Here k –1 is less than k i.e. k-1 is not in the set T. Since the implication P(k-1) → P(k) is also true, P(k) must be true. This contradicts the choice of k. Hence, P(n) must be true for all positive integers n.

Remember!!! You may prove wrongly if you do not take care

Prove $a^n = 1$ for all nonnegative integers n, whenever a is a nonzero real number.

Proof:

Basis Step: for n = 0, $a^0 = 1$ by the definition of a^0.

Inductive Hypothesis: assume that $a^k = 1$ for all nonnegative integers $k \leq n$.

Inductive Step: we have $a^{k+1} = a^k . a^k / a^{k-1} = 1.1/1 = 1$.

Hence proved.

Attention:

Whenever k + 1 = 1 the above proof fails because here k = 0 so that $a^{k+1} = a^k . a^k / a^{k-1} = a^0 . a^0 / a^{-1} = ?$. Here we cannot get the value for denominator from previously obtained value. So choosing base n = 0 does not produce correct result for n = 1 but mathematical induction says that P(0) → P(1) which is not true here. Similarly choosing base n =1 disprove the statement at basis step.

Recursive Definition

The process of defining the object in terms of itself is called recursion. Such a way of representation is given by recursive definition. For e.g. natural numbers can be defined in terms of itself as $N_n = N_{n-1} + 1$, for $N_0 = 0$ and n = 0, 1, 2, ….

Recursively Defined Functions, Sets and Structures

When we try to define a function recursively, where the domain of the function is set on nonnegative integers, we define such a function through two steps:

Basis Step: Specify the value of the function at base (base is generally 0 or 1). This is generally well known value of the function at the lowest value of integer.

Recursive Step: Specify the rule for finding the value of a function by using the value of a function already found i.e. at first base case is used and next result obtained from function definition that uses base case, and so on.

This kind of definition is also called **inductive definition**.

Similarly if you want to define sets or structures then the similar two steps above is used. You may also put exclusion rule in recursive step such that the elements of the sets are specified.

Example 1:

Give a recursive definition of a sequence $\{a_n\}$, n =1, 2, …, n if $a_n = 10^n$.

Solution:

Basis Step: $a_1 = 10^1 = 10$.

Recursive Step: $a_n = 10a_{n-1}$. This is the recursive definition required.

Example 2:

Give a recursive definition of the set of even positive integers.

Solution:

Let E be the set of even positive integers.

Basis Step: $2 \in E$

Recursive Step: If $a \in E$, then $a + 2 \in E$.

The above recursive definition gives a set of even positive integers.

Note: To prove that the recursive definition is correct we can use mathematical induction principle.

Example 4:

Show that $f_1^2 + f_2^2 + \ldots + f_n^2 = f_n f_{n+1}$, whenever n is a positive integer. Here f_i's are i^{th} fibonacci numbers (see book for more details on fibonacci numbers).

Proof:

Let P(n) be $f_1^2 + f_2^2 + \ldots + f_n^2 = f_n f_{n+1}$.

Basis Step: $P(1) = f_1^2 = 1^2 = 1.1 = f_1 f_2$. So P(1) is true.

Inductive Hypothesis: Assume that P(k) is true for all positive integers $k \leq n$.

Inductive Step: We have

$P(k+1) = f_1^2 + f_2^2 + \ldots + f_{k+1}^2$

$= f_k f_{k+1} + f_{k+1}^2$

$= f_{k+1}(f_k + f_{k+1})$

$= f_{k+1} f_{k+2}$. [$f_k + f_{k+1} = f_{k+2}$, this is fibonacci numbers property]

Hence P(k+1) is true.

So by mathematical induction P(n) is true for all positives integers n.

Theorem 1(Lame's Theorem): Let a and b be positive integers with $a \geq b$ then number of divisions used by the Euclidean algorithm to find gcd(a,b) is less than or equal to five times the number of decimal digits in b.

Proof:

Euclidean algorithm for finding gcd(a, b) with $a \geq b$ gives the following sequence of equations.

$r_0 = r_1 q_1 + r_2$. $0 \leq r_2 < r_1$. [Here $r_0 = a$ and $r_1 = b$]

$r_1 = r_2 q_2 + r_3$. $0 \leq r_3 < r_2$.

$\vdots$

$r_{n-2} = r_{n-1} q_{n-1} + r_n$. $0 \leq r_n < r_{n-1}$.

$r_{n-1} = r_n q_n$.

The gcd(a, b) = r_n and n divisions are used for finding it. All the quotients q_i, for i =1, 2,…n all at least 1. We have $q_n \geq 2$, since $r_n < r_{n-1}$. So we have

$r_n \geq 1 = f_2$.

$r_{n-1} \geq 2\ r_n \geq 2f_2 = f_3$.

$r_{n-2} \geq r_{n-1} + r_n \geq f_3 + f_2 = f_4$.

⋮

$r_2 \geq r_3 + r_4 \geq f_{n-1} + f_{n-2} = f_n$.

$b = r_1 \geq r_2 + r_3 \geq f_n + f_{n-1} = f_{n+1}$.

From the above relations we can conclude that if n divisions are used by the Euclidean algorithm to find gcd(a, b) with a ≥ b then $b \geq f_{n+1}$. We have $f_{n+1} > \phi^{n-1}$ for n > 2, where $\phi = (1 + \sqrt{5})/2$ (prove using mathematical induction). So we have $b > \phi^{n-1}$.

Now taking log on both sides, $\log_{10} b > (n-1)\log_{10}\phi$ but since $\log_{10}\phi \approx 0.208 > 1/5$, we have $\log_{10} b > (n-1)/5$ i.e. $(n-1) < 5 \log_{10} b$. If we know that the b has m decimal digits, then we have $b < 10^m$ and $\log_{10} b < m$. Here (n-1) < 5m, and since m is an integer, n ≤ 5m. This is the proof.

Structural Induction

While proving the recursively defined sets we use a form of mathematical induction called structural induction. This method consists two parts.

Basis Step: Show that the result holds for all elements specified in the basis step of the recursive definition to be in the set.

Recursive Step: Show that if the statement is true for each of the elements used to construct new elements in the recursive step of the definition, the result holds for these new elements.

Validity of Structural Induction

The validity of structural induction can be seen as the validity of the mathematical induction. If P(n) denotes the statement that is recursively defined, for all positive integers n. The basis step of the structural induction method correspondence to the basis

step of the mathematical induction method. We can see that the recursive step in the structural induction tells if P(k) is true it implies P(k+1), where P(k) is assumed already and the P(k+1) is derived in terms of P(k). Hence it follows the proofs by mathematical induction.

Example:

Recursive definition of the set of leaves and the set of internal vertices of a full binary tree can be defined as:

Basis Step: The root r is a leaf of the full binary tree with exactly one vertex r. This tree has no internal vertices.

Recursive Step: The set of leaves of the tree $T = T_1.T_2$ is the union of the set of the leaves of T_1 and the set of leaves of T_2. The internal vertices of T are the root r of T and the union of the set of internal vertices of T_1 and the set of internal vertices of T_2.

Use structural induction to show that l(T), the number of leaves of a full binary tree T, is 1 more than i(T), the number of internal vertices of T.

Proof:

Basis Step: Root r of a full binary tree has only one vertex and that is leaf of the tree. So l(T) = 1 and i(T) = 0, hence clearly T contains number of leaves 1 greater than number of internal vertices i.e. l(T) - i(T) = 1.

Recursive Step: Assume that T_1 and T_2 are trees holding the property $l(T_1) – i(T_1) = 1$ and $l(T_2) – i(T_2) = 1$. To complete the proof we must show that l(T) – i(T) = 1, where $T = T_1.T_2$. We know that $l(T) = l(T_1) + l(T_2)$ [form the recursive definition] and $i(T) = 1 + i(T_1) + i(T_2)$ [from the recursive definition]. So,

$$\begin{aligned} l(T) – i(T) &= l(T_1) + l(T_2) – 1 - i(T_1) - i(T_2). \\ &= l(T_1) - i(T_1) + l(T_2) - i(T_2) – 1. \\ &= 1 + 1 – 1 \text{ [from assumption above]} = 1 \end{aligned}$$

Hence the proof.

Recursive Algorithms

An algorithm is recursive it solves the problem by reducing the size of the same problem using smaller input size. In this section few recursive algorithms are presented.

Example 1:

Give the recursive algorithm for finding the sum of the first n positive integers.

Solution:

Input: A positive integer n.

Output: the sum of positive integers form 1 up to n.

PositiveInteger nsum(PositiveInteger n)

{

if(n = =1) then return 1;

else return nsum(n-1) + n;

}

Example 2:

Devise a recursive algorithm for finding x^n**mod**m whenever n, x, and, m are positive integers based on the fact that x^n**mod**m = (x^{n-1}**mod**m.x**mod**m) **mod**m.

Solution:

Input: Three positive integers n, x, and m.

Output: n^{th} x modulo m.

PositiveInteger nmod(n, x, m)

{

*if n = = 1 then return x**mod**m;*

*else return ((nmod(n-1, x, m).x**mod**m)**mod**m.*

}

Note: Try to understand difference between use of recursion and iteration

Self Studies

Read chapter 1 and 3 of your textbook such that you can cover all the read materials in the class.

[Combinatorics]

Discrete Structures (CSc 511)

Samujjwal Bhandari

Central Department of Computer Science and Information Technology (CDCSIT)

Tribhuvan University, Kirtipur,

Kathmandu, Nepal.

Elementary Combinatorics

Combinatorics is the study of arrangements or possible combination of objects. We come up with different situations where we need to identify the number of elements having similar features, number of steps required to solve the problem, amount of storage required, etc.

Basics of Counting

There are two basic counting principles that can be used to solve the counting problems. We define those two principles below:

Sum rule: The principle of disjunctive counting.

If the first task can be done in m ways and the second task can be done in n ways and if both the tasks cannot be done at a time, then there are m + n ways to do one of the task. We can generalize this rule as, if a set X is union of disjoint nonempty subsets S_1, S_2, …, S_n, then $|X| = |S_1| + |S_2| + \ldots + |S_n|$.

Remember: the set must be disjoint, for overlapping set we use different principle called inclusion exclusion principle (will be covered later).

Example 1:

In how many ways we can draw a heart or a diamond from an ordinary deck of playing cards?

Solution:

There are total 13 cards of heart and 13 card of diamond. So, by sum rule total number of ways of picking heart or diamond is 13 + 13 = 26.

Example 2:

How many ways we can get a sum of 4 or of 8 when two distinguishable dice (say one die is red and the other is white) are rolled?

Solution:

Since dice are distinguishable outcome (1, 3) is different form (3, 1) so to get 4 as sum we have the pairs (1, 3), (3, 1), (2, 2), so total of 3 ways. And similarly getting 8 can be from pairs (2, 6), (6, 2), (3, 5), (5, 3), (4, 4), so total 5 ways. Hence getting sum of 4 or 8 is 3 + 5 = 8.

Product Rule: Principle of sequential counting.

If a work can be done in m ways and another work can be done after the completion of first work in n ways, then there are m × n ways to do the task that consists both the work. Generalizing the rule, if S_1, S_2, …, S_n are non empty sets, then the number of elements in the Cartesian product $S_1 \times S_2 \times \ldots \times S_n$, is the product $\prod_{i=1}^{n} | S_i |$ i.e. $| S_1 \times S_2 \times \ldots \times S_n | = \prod_{i=1}^{n} | S_i |$.

Example 1:

An office building contains 27 floors and has 37 offices on each floor. How many offices are there are in the building?

Solution:

By the product rule there are 27.37 = 999 offices in the building.

Example 2:

How many different three-letter initials with none of the letters can be repeated can people have?

Solution:

Here the first letter can be chosen in 26 ways, since the first letter is assigned we can choose second letter in 25 ways and in the same manner we can choose third letter in 24 ways. So by product rule number of different three-letter initials are 26.25.24 = 15600.

More Examples on Basics:

Example 1:

How many strings are there of four lowercase letters that have the letter x in them?

Solution:

There are total 26.26.26.26 strings of four lowercase letters, by product rule. In the same way we can say that there are 25.25.25.25 strings of four lowercase letters without x, since without x there will be a set of 25 characters only. So there are total of 26.26.26.26 - 25.25.25.25 = 66351 four lowercase letter strings with x in them. This is true because we are decrementing total numbers of strings with the number of strings that do not contain x in them so at least one x will be in the strings.

Example 2:

How many functions are there from the set {1, 2, …, n}, where n is a positive integer, to the set {0, 1}.

Solution:

Each element from the set {1, 2, …, n}can map the set {0, 1} in 2 ways. Since there are n elements in the first set by the product rule number of possible functions are 2.2.2…..n^{th} term i.e. 2^n.

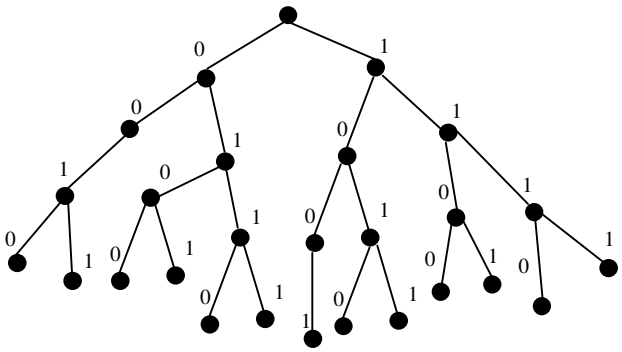
Tree Diagrams

We can use a tree diagram to solve the counting problem (don't worry we will study tree in detail later).

Example:

Use a tree diagram to find the number of bit strings of length four with no three consecutive 0s.

Solution:



From the above tree we can get that there are total number of 13 bit strings of length four with no three consecutive zeroes. For this we can explain as if a bit string start with 1 then there is only one bit string that can have three consecutive 0s (1000), the total number of bit string of length starting with 1 and have no three consecutive 0s is thus 2.2.2 –1 =7, similarly if the bit string start with 0 then there is a possibility that the next two bits may be 0 so the possible bit strings of length four with consecutive 0s starting with 0 are 0001 and 0000, so the total number of bit string of length starting with 1 and have no three consecutive 0s is thus 2.2.2 –2 =6. Using the sum rule the total number of such bit strings is 13.

Pigeonhole principle

The pigeonhole principle states that if there are more pigeons than pigeonholes, then there must be at least one pigeonhole with at least two pigeons. The concept of pigeons can be extended to any objects.

Theorem 1: The pigeonhole principle

If k + 1 or more objects are placed into k boxes, then there is at least one box containing two or more of the objects.

Proof:

We use proof by contradiction here. Suppose that k+1 or more boxes are placed into k boxes and no boxes contain more than one object in it. If there are k boxes then there must be k objects such that there are no two objects in a box. This contradicts our assumption. So there is at least one box containing two or more of the objects.

Example:

Show that if there are 30 students in a class, then at least two have last names that begin with the same letter.

Proof:

There are 30 students in the class and we have 26 letters in English alphabet that can be used in beginning of the last name. Since there are only 26 letters and 30 students, by pigeonhole principle at least two students have the last name that begins with the same letter.

Theorem 2: The generalized pigeonhole principle

If N objects are placed into k boxes, then there is at least one box containing at least $\lceil N/k \rceil$ objects.

Proof:

Suppose N objects are placed into k boxes and there is no box containing more than $\lceil N/k \rceil$-1 objects. So the total number of objects is at most

$k(\lceil N/k \rceil - 1) < k((N/k + 1) - 1) = N$. This is the contradiction that N objects are placed into k boxes (since we showed that there are total number of objects less than N). Hence, the proof.

Example:

If a class has 24 students, what is the maximum number of possible grading that must be done to ensure that there at least two students with the same grade.

Solution:

There are total 24 students and the class and at least two students must have same grade. If the number of possible grades is k then by pigeonhole principle we have $\lceil 24/k \rceil = 2$.

Here the largest value that k can have is 23 since 24 = 23.1 + 1. So the maximum number of possible grading to ensure that at least two of the students have same grading is 23.

Applications: Pigeonhole principles

Example 1:

How many numbers must be selected from the set {1, 3, 5, 7, 9, 11, 13, 15} to guarantee that at least one pair of these numbers add up to 16?

Solution:

The pairs of numbers that sum 16 are (1,15), (3, 13), (5, 11), (7, 9) i.e. 4 pairs of numbers are there that add to 16. If we select 5 numbers then by pigeonhole principle there are at least $\lceil 5/4 \rceil = 2$ numbers, that are from the set of selected 5 numbers, that constitute a pair. Hence 5 numbers must be selected.

Example 2:

Find the least number of cables required to connect eight computers to four printers to guarantee that four computers can directly access four different printers. Justify your answer.

Solution:

If we connect first 4 computers directly to each of the 4 printers and the other 4 computers are connected to all the printers, then the number of connection required is 4 + 4.4 = 20. To verify that 20 is the least number of cables required we have if there may be less than 20 cables then we would have 19 cables, then some printers would be connected by at most $\lfloor 19/4 \rfloor = 4$ cables to the computers. Then the other 3 printers would have to connect the other 4 computers here all the computers cannot simultaneously access

different printer. So if we use 20 cables, then at least $\lceil 20/4 \rceil$ = 5 cables connects a printer to a computer directly. So the remaining 3 printers are required to connect only 3 computers. Hence the least number of cables required is 20.

Example 3:

Among n + 1 different integral powers of an integer a, there are at least two of them that have same remainder when divided by the positive integer n.

Proof:

Let a^1, a^2, …, a^{n+1}, be n+1 different integral powers of integer a. when these numbers are divided by n then the set of possible remainders is {0, 1, 2, …, n-1). Since there are n remainders and n+1 numbers by pigeonhole principle at least 2 of the reminders must be same.

Self Studies

Read chapter 4.1 and 4.2 of your textbook such that you can cover all the read materials in the class.

[Advance Counting]

Discrete Structures (CSc 511)

Samujjwal Bhandari

Central Department of Computer Science and Information Technology (CDCSIT)

Tribhuvan University, Kirtipur,

Kathmandu, Nepal.

Recurrence Relations

Some counting problems cannot be solved using the methods we have learnt before. One of the ways of solving counting problems is by finding relationships, called recurrence relation, between the terms of a sequence. When we represent some problem using recursive definition then we specify some initial condition and the recursive condition. We use such definition to solve the relation called recurrence relation.

A **recurrence relation** for the sequence $\{a_n\}$ is an equation that expresses a_n in terms of one or more of the previous terms of the sequence, namely, a_0, a_1, …, a_{n-1}, for all integers n with $n \geq n_0$, where n_0 is a nonnegative integer. A sequence is called a solution of a recurrence relation if its term satisfies the recurrence relation.

Example:

Let $\{a_n\}$ be a sequence that satisfies the recurrence relation $a_n = a_n + 1$ for n = 1, 2, …, and suppose that $a_1 = 1$. What is the sequence?

Solution:

We have $a_1 = 1$, $a_2 = a_1 + 1 = 1 + 1 = 2$ ….. In similar way we have the set {1, 2, ….}.

Example:

For $a_n = -2a_{n-1}$, $a_0 = -1$ find a_1, a_2, … a_5.

Solution:

We have $a_0 =- -1$

$a_1 = -2a_0 = -2.-1 = 2.$

$a_2 = -2a_1 = -2.2 = -4.$

$a_3 = -2a_2 = -2.-4 = 8.$

$a_4 = -2a_3 = -2.8 = -16.$

$a_5 = -2a_4 = -2.-16 = 32.$

Example:

Is the sequence $\{a_n\}$ a solution of the recurrence relation $a_n = -3a_{n-1} + 4a_{n-2}$ if a) $a_n = 0$? and b) $a_n = 2n$?

Solution:

$a_n = -3a_{n-1} + 4a_{n-2}$, for $a_n = 0$ we have **a)** $a_n = 0$ so the sequence $\{a_n\}$ is a solution. b) $a_n = -3.2(n-1) + 4.2(n-2) = 6n – 6 + 8n – 16 = 14n – 22 \neq 2n$, so it is not a solution.

Example:

Find the recurrence relation satisfied by the sequence $a_n = n!$, and $a_n = 2n + 3$ (There may be more than one relation for some sequence).

Solution:

Take $a_0 = 1$ and $a_n = a_{n-1}.n$ (this is the relation for $a_n = n!$)

Take $a_1 = 5$ and $a_n = a_{n-1} + 2$ (this is the relation for $a_n = 2n + 3$, verify!!)

Example:

Find a recurrence relation for the number of bit strings of length n with an even numbers of 0s.

Solution:

Let a_n denotes the number of bit strings of length n with even numbers of 0s. There is 1 bit string of length one that is valid since among two bits we can choose only 1. Recursively we can define this in terms of bit strings of length n –1. Here we have two conditions for getting bit strings of length n-1 with even numbers of 0s. First, if the bit strings end with 1, then we can have valid bit stings of length n – 1 ending with 1 so that there are a_{n-1} numbers with even numbers of 0s. Second, if the bit strings end with 0, then we can make a bit string valid if we add the bit strings of length n-1 that have odd numbers of 0s. Since there are 2^{n-1} possible ways of getting bit strings of length n-1 and a_{n-1} is the number of valid bit strings of length n-1 we have $2^{n-1} - a_{n-1}$ numbers of invalid bit strings of length n-1. So, the total numbers of valid bit strings is $a_{n-1} + 2^{n-1} - a_{n-1} = 2^{n-1}$. Hence we have the relation $a_n = 2^{n-1}$(This is not recursive but method used was recursive). So since we have $a_1 = 1$, we can write $a_n = 2a_{n-1}$.

Solving Recurrences

We encounter different types of recurrence relations. There is no specific technique to solve all the recurrence relation. However, we solve recurrence relation with some particular forms by using the systematic methods. In this section we are going to see few of them.

Solving Linear Homogeneous Recurrence Relations with Constant Coefficients

A linear homogeneous recurrence relation of degree k with constant coefficients is a recurrence relation of the form $a_n = c_1a_{n-1} + c_2a_{n-2} + \ldots + c_ka_{n-k}$, where $c_1, c_2, \ldots, c_k$ are real numbers, and $c_k \neq 0$. The above relation is linear since right hand side is a sum of the multiples of previous terms of the sequence. It is homogeneous since no term occurs without being multiple if some a_js. All the coefficients of the terms are constants and degree k is due to the representation of a_n in terms of previous k terms of the sequence.

In solving the recurrence relation of the type above, the approach is to look for the solution of the form $a_n = r^n$, where r is a constant. $a_n = r^n$ is a solution of a recurrence relation $a_n = c_1a_{n-1} + c_2a_{n-2} + \ldots + c_ka_{n-k}$ if and only if $r^n = c_1r^{n-1} + c_2r^{n-2} + \ldots + c_kr^{n-k}$. when we divide both sides by r^{n-k} and transpose the right hand side we have

$r^k - c_1r^{k-1} - c_2r^{k-2} - \ldots - c_k = 0$. Here we can say $a_n = r^n$ is a solution if and only if r is the solution if the equation $r^k - c_1r^{k-1} - c_2r^{k-2} - \ldots - c_k = 0$ (**characteristic equation** of the recurrence relation) and solutions to this equations are called **characteristic roots** of the recurrence relation.

Theorem 1: (without proof)

Let c_1 and c_2 be real numbers. Suppose that $r^2 - c_1r - c_2 = 0$ has two distinct roots r_1 and r_2. Then the sequence $\{a_n\}$ is a solution of the recurrence relation $a_n = c_1a_{n-1} + c_2a_{n-2}$ if and only if $a_n = \alpha_1r_1^n + \alpha_2r_2^n$ for n = 0, 1, 2, …, where α_1 and α_2 are constants.

Example:

Solve the recurrence relation $a_n = a_{n-1} + 6a_{n-2}$ for $n \geq 2$, $a_0 = 3$, $a_1 = 6$.

Solution:

Characteristic equation of the given relation is $r^2 - r - 6 = 0$. Its roots are r = 3 and r = -2 since $(r - 3)(r + 2) = 0$. Hence, the sequence $\{a_n\}$ is a solution to the recurrence relation if and only if $a_n = \alpha_1 3^n + \alpha_2(-2)^n$, for some constants α_1 and α_2. From the initial conditions we have $a_0 = 3 = \alpha_1 + \alpha_2$, $a_1 = 6 = 3\alpha_1 + (-2)\alpha_2$. Solving these two equations we have $\alpha_1 = 12/5$ and $\alpha_1 = 3/5$. Hence, the solution is the sequence $\{a_n\}$ with $a_n = (12.3^n + 3(-2)^n)/5$.

Theorem 2: (without proof)

Let c_1 and c_2 be real numbers with $c_2 \neq 0$. Suppose that $r^2 - c_1r - c_2 = 0$ has only one root r_0. Then the sequence $\{a_n\}$ is a solution of the recurrence relation $a_n = c_1a_{n-1} + c_2a_{n-2}$ if and only if $a_n = \alpha_1r_0^n + \alpha_2nr_0^n$ for n = 0, 1, 2, …, where α_1 and α_2 are constants.

Example:

Solve the recurrence relation $a_n = 2a_{n-1} - a_{n-2}$ for $n \geq 2$, $a_0 = 3$, $a_1 = 6$.

Solution:

Characteristic equation of the given relation is $r^2 - 2r + 1 = 0$. Its only root are r = 1. Hence, the sequence $\{a_n\}$ is a solution to the recurrence relation if and only if $a_n = \alpha_1 1^n + \alpha_2 n1^n$, for some constants α_1 and α_2. From the initial conditions we have $a_0 = 3 = \alpha_1$, $a_1 = 6 = \alpha_1 + \alpha_2$. Solving these two equations we have $\alpha_1 = 3$ and $\alpha_2 = 3$. Hence, the solution is the sequence $\{a_n\}$ with $a_n = 3(1^n + n1^n)$.

Theorem 3: (without proof)

Let $c_1, c_2, \ldots, c_k$ be real numbers. Suppose that $r^k - c_1r^{k-1} - \ldots - c_k = 0$ has k distinct roots $r_1, r_2, \ldots, r_k$. Then the sequence $\{a_n\}$ is a solution of the recurrence relation $a_n = c_1a_{n-1} + c_2a_{n-2} + \ldots + c_ka_{n-k}$ if and only if $a_n = \alpha_1r_1^n + \alpha_2r_2^n + \ldots + \alpha_kr_k^n$ for n = 0, 1, 2, …, where $\alpha_1, \alpha_2, \ldots, \alpha_k$ are constants.

Example:

Solve the recurrence relation $a_n = 2a_{n-1} + a_{n-2} - 2a_{n-3}$ for $n \geq 3$, $a_0 = 3$, $a_1 = 6$ and $a_2 = 9$.

Solution:

Characteristic equation of the given relation is $r^3 - 2r^2 - r + 2 = 0$. Its roots are r = 1, r = -1, and r = 2. Hence, the sequence $\{a_n\}$ is a solution to the recurrence relation if and only if $a_n = \alpha_1 1^n + \alpha_2(-1)^n + \alpha_3 2^n$, for some constants α_1, α_2, and α_3. From the initial conditions we have $a_0 = 3 = \alpha_1 + \alpha_2 + \alpha_3$, $a_1 = 6 = \alpha_1 - \alpha_2 + 2\alpha_3$, and $a_2 = 9 = \alpha_1 + \alpha_2 + 4\alpha_3$. Solving these two equations we have $\alpha_1 = 3/2$, $\alpha_2 = -1/2$, and $\alpha_3 = 2$. Hence, the solution is the sequence $\{a_n\}$ with $a_n = (3/2)1^n - (1/2)(-1)^n + 2.2^n$.

Theorem 4: (without proof)

Let $c_1, c_2, \ldots, c_k$ be real numbers. Suppose that $r^k - c_1r^{k-1} - \ldots - c_k = 0$ has t distinct roots $r_1, r_2, \ldots, r_t$ with multiplicity $m_1, m_2, \ldots, m_t$, respectively, so that $m_i \geq 1$ for $i = 1, 2, \ldots, t$ and $m_1 + m_2 + \ldots + m_t = k$. Then the sequence $\{a_n\}$ is a solution of the recurrence relation $a_n = c_1a_{n-1} + c_2a_{n-2} + \ldots + c_ka_{n-k}$ if and only if

$$a_n = (\alpha_{1,0} + \alpha_{1,1}n + \ldots + \alpha_{1,m1-1}n^{m1-1})\, r_1^n$$
$$+ (\alpha_{2,0} + \alpha_{2,1}n + \ldots + \alpha_{2,m2-1}n^{m2-1})\, r_2^n$$
$$+ \ldots + (\alpha_{t,0} + \alpha_{t,1}n + \ldots + \alpha_{t,mt-1}n^{mt-1})\, r_t^n$$

for $n = 0, 1, 2, \ldots$, where $\alpha_{i,j}$ are constants for $1 \leq i \leq t$ and $0 \leq j \leq m_i-1$.

Example:

Solve the recurrence relation $a_n = 5a_{n-1} - 7a_{n-2} + 3a_{n-3}$ for $n \geq 3$, $a_0 = 1$, $a_1 = 9$ and $a_2 = 15$.

Solution:

Characteristic equation of the given relation is $r^3 - 5r^2 + 7r - 3 = 0$. Its roots are $r = 1$, $r = 3$, and $r = 1$. i.e. $r_1 = 1$, $m_1 = 2$ and $r_2 = 3$, $m_2 = 1$ Hence, the sequence $\{a_n\}$ is a solution to the recurrence relation if and only if $a_n = (\alpha_{1,0} + \alpha_{1,1}n)\, 1^n + (\alpha_{2,0})\, 3^n$, for some constants $\alpha_{1,0}$, $\alpha_{1,1}$, and $\alpha_{2,0}$. From the initial conditions we have $a_0 = 5 = \alpha_{1,0} + \alpha_{2,0}$, $a_1 = 9 = \alpha_{1,0} + \alpha_{1,1} + 3\alpha_{2,0}$, and $a_2 = 15 = \alpha_{1,0} + 2\alpha_{1,1} + 9\alpha_{2,0}$. Solving these two equations we have $\alpha_{1,0} = 3/2$, $\alpha_{1,1} = 9$, and $\alpha_3 = -1/2$. Hence, the solution is the sequence $\{a_n\}$ with $a_n = (3/2)1^n + 9n1^n - (1/2)3^n$.

Solving Linear Nonhomogeneous Recurrence Relations with Constant Coefficients

The recurrence relation of the form $a_n = c_1a_{n-1} + c_2a_{n-2} + \ldots + c_ka_{n-k} + F(n)$, where $c_1, c_2, \ldots, c_k$ are real numbers and F(n) is a function depending upon n. The recurrence relation preceding F(n) is called **associated homogeneous recurrence relation**. For example $a_n = 7a_{n-1} + 3a_{n-2} + 6n$ is a linear nonhomogeneous recurrence relation with constant coefficients.

Theorem 5: (without proof)

If $\{a_n^{(p)}\}$ is a particular solution of the nonhomogeneous linear recurrence relation with constant coefficients $a_n = c_1a_{n-1} + c_2a_{n-2} + \ldots + c_ka_{n-k} + F(n)$, then every solution of the form $\{a_n^{(p)} + a_n^{(h)}\}$, where $a_n^{(h)}$ is a solution of the associated homogeneous recurrence relation $a_n = c_1a_{n-1} + c_2a_{n-2} + \ldots + c_ka_{n-k}$.

Example:

Find all the solutions of the recurrence relation $a_n = 4a_{n-1} + n^2$. Also find the solution of the relation with initial condition $a_1 = 1$.

Solution:

We have associated linear homogeneous recurrence relation as $a_n = 4a_{n-1}$. The root is 4, so the solutions are $a_n^{(h)} = \alpha 4^n$, where α is a constant. Since $F(n) = n^2$ is a polynomial of degree 2, a trial solution is a quadratic function in n, say, $p_n = an^2 + bn + c$, where a, b, and c are constants. To determine whether there are any solution of this form, suppose that $p_n = an^2 + bn + c$ is such solution. Then the equation $a_n = 4a_{n-1} + n^2$ becomes

$$\begin{aligned} an^2 + bn + c &= 4(a(n-1)^2 + b(n-1) + c) + n^2. \\ &= 4a\,n^2 - 8an + 4a + 4bn - 4b + 4c + n^2 \\ &= (4a + 1)n^2 + (-8a + 4b)n + (4a - 4b + 4c) \end{aligned}$$

Here $an^2 + bn + c$ is the solution if and only if $4a + 1 = a$ i.e. $a = -1/3$; $-8a + 4b = b$ i.e. $b = -8/3$; $4a - 4b + 4c = c$ i.e. $c = -28/3$. So $a_n^{(p)} = -(n^2 + 8n + 28)/3$ is a particular solution and all solutions are $a_n = \{a_n^{(p)} + a_n^{(h)}\} = -(n^2 + 8n + 28)/3 + \alpha 4^n$, where α is a constant.

For solution with $a_n = 1$, we have $a_n = 1 = -(1 + 8 + 28)/3 + \alpha 4$ i.e. $\alpha = 10/3$. Then the solution is $a_n = (10.4^n - n^2 - 8n - 28)/3$.

Theorem 6: (without proof)

Suppose that {an} satisfies the linear nonhomogeneous recurrence relation $a_n = c_1a_{n-1} + c_2a_{n-2} + \ldots + c_ka_{n-k} + F(n)$, where $c_1, c_2, \ldots, c_k$ are real numbers and $F(n) = (b_tn^t + b_{t-1}n^{t-1} + \ldots + b_1n + b_0)s^n$, where $b_0, b_1, \ldots, b_t$ and s are real numbers.

When s is not a root of the characteristic equation of the associated linear homogeneous recurrence relation, there is a particular solution of the form

$(p_t n^t + p_{t-1} n^{t-1} + \ldots + p_1 n + p_0) s^n$.

When s is a root of the characteristic equation and its multiplicity is m, there is a particular solution of the form

$n^m(p_t n^t + p_{t-1} n^{t-1} + \ldots + p_1 n + p_0) s^n$.

Example:

Find the solution of the recurrence relation $a_n = 2a_{n-1} + n.2^n$.

Solution:

We have the associated linear homogeneous recurrence relation is $a_n = 2a_{n-1}$. The characteristic equation for this would be r-2 = 0, so the root is 2 and hence the solutions are $a_n^{(h)} = \alpha 2^n$, where α is a constant. We have $F(n) = n.2^n$. (Of the form $n.s^n$) where s is the root of the characteristic equation and the multiplicity of 2 is 1 so, the particular solution has the form $n.(p_1 n)2^n = p_1 n^2 2^n$. The solution is, $a_n = \alpha 2^n + p_1 n^2 2^n$.

Recurrences Applications

One of the application areas of recurrence relations is analysis of divide and conquer algorithms.

Divide and Conquer Algorithms

Divide and conquer algorithms divide a problem of larger size to the problem of smaller size so continually such that the problem of the smallest size that has trivial solution is obtained. If f(n) represents the number of operations required to solve the problem of size n, then follows the recurrence relation f(n) = af(n/b) + g(n), called divide and conquer recurrence relation. In the relation above the problem of size n is partitioned into a parts of the problem of the size n/b and g(n) is the operations required to conquer the solutions. In this section no algorithms are presented but their recurrence relations are tried to achieve.

Example 1:Fibonacci Numbers

We know that the fibonacci numbers are generated by the formula $f_n = f_{n-1} + f_{n-2}$. Here n^{th} Fibonacci number is the sum of $(n-1)^{th}$ and $(n-2)^{nd}$ fibonacci numbers. Here for the initial

conditions are $f_0 = 0$, and $f_1 = 1$. Use of the above relation does not exactly produce the recurrence relation mentioned above, however this is an example of divide and conquer algorithm since each time the problem is changed into two problems of smaller size.

Example 2: Merge Sort

In merge sorting the input sequence of n items is broken down into 2 halves (here there may be difference in 1 item between two parts). Since the list of size n need more comparisons than list of size n/2, the problem here is simplified. This process continues until all the comparisons are trivial. This problem satisfies the divide and conquer recurrence relation

M(n) = 2M(n/2) + O(1).

Theorem 7: (without proof)

Let f be an increasing function that satisfies the recurrence relation f(n) = af(n/b) + c whenever n is divisible by b, where a ≥ 1, b is an integer greater than 1, and c is a positive real number. Then $f(n)$ is $\begin{cases} O(n^{\log_b a}) & \text{if } a > 1, \\ O(\log n) & \text{if } a = 1. \end{cases}$. Furthermore, when n = b^k, where k is a positive integer, $f(n) = C_1 n^{\log_b a} + C_2$, where $C_1 = f(1) + c/(a-1)$ and $C_2 = -c/(a-1)$.

Example:

Solve the recurrence relations f(n) = 2f(n/2) + 4 and g(n) = g(n/2) + 2 to get their upper bound.

Solution:

Using theorem 7 above we have f(n) = O(n) since, $\log_b a = \log_2 2 = 1$. Similarly we have g(n) = O(logn).

Theorem 7: Master Theorem (without proof)

Let f be an increasing function that satisfies the recurrence relation f(n) = af(n/b) + cn^d whenever n = b^k, where k is a positive integer, a ≥ 1, b is an integer greater than 1, c is a positive real number, and d is nonnegative real number. Then

$$f(n) \text{ is } \begin{cases} O(n^d) & \text{if } a < b^d, \\ O(n^d \log n) & \text{if } a = b^d, \\ O(n^{\log_b a}) & \text{if } a > b^d. \end{cases}$$

Example:

Solve using Master Theorem the following recurrences where each n is by 2^k.

i) $f(n) = 2f(n/2) + n^2$.

ii) $f(n) = 2f(n/2) + n$.

iii) $f(n) = 7f(n/2) + n^2$.

Solution:

Using master theorem we have

i) $f(n) = O(n^2)$, since $(a)2 < 2^2(b^d)$

ii) $f(n) = O(n\log n)$, since $(a)2 = 2(b^d)$

iii) $f(n) = O(n^{2.807})$, since $(a)7 > 2^2(b^d)$

Inclusion and Exclusion and Applications

In the counting problems where the sets are not disjoint we extensively use inclusion exclusion principle. Given set A and set B the union of A and B is given by the formula $|A \cup B| = |A| + |B| - |A \cap B|$.

Example 1:

There are 345 students at a college who have taken a course in calculus, 212 who have taken a course in discrete mathematics, and 188 who have taken course in both calculus and discrete mathematics. How many students have taken the course in either calculus or discrete mathematics?

Solution:

Here we have $|C| = 345$ (students taking the calculus course), $|D| = 212$ (students taking the discrete mathematics course), and $|C \cap D| = 188$ (students taking both discrete mathematics and calculus courses). Number of students taking either discrete mathematics or calculus, $|C \cup D| = |C| + |D| - |C \cap D| = 345 + 212 – 188 = 369$.

Theorem 8: The Principle of Inclusion – Exclusion

Let $A_1, A_2, \ldots, A_n$ be finite sets. Then $|A_1 \cup A_2 \cup \ldots \cup A_n|$

$$= \sum_{1\le i\le n} |A_i| - \sum_{1\le i<j\le n} |A_i \cap A_j| + \sum_{1\le i<j<k\le n} |A_i \cap A_j \cap A_k| - \ldots + (-1)^{n+1} |A_1 \cap A_2 \cap \cdots \cap A_n|.$$

Proof:

We try to prove this theorem using counting technique. Suppose that the element a is a member of exactly r of the sets $A_1, A_2, \ldots, A_n$, where $1 \le r \le n$. Then this element is counted C(r, 1) times in $\sum|A_i|$. The element a is counted C(r, 2) times in $\sum|A_i \cap A_j|$. So if there are m sets C(r, m) times a is counted from the summation that contains m of the sets of A_i. Using these counts in the right hand side of the formula above we get

$C(r, 1) - C(r, 2) + C(r, 3) - \ldots + (-1)^{r+1}C(r, r)$ counts for the common element (i.e. a here)

We know that

$C(r, 0) - C(r, 1) + C(r, 2) - \ldots + (-1)^{r}C(r, r) = 0.$

So,

$-C(r, 0) + C(r, 1) - C(r, 2) - \ldots + (-1)^{r+1}C(r, r) = 0$

$C(r, 0) = C(r, 1) - C(r, 2) - \ldots + (-1)^{r+1}C(r, r) = 1$ [since C(r, 0) = 1]

From this fact we know that each element from the all sets are counted just once. Hence the proof.

Example:

How many elements are in the union of four sets if each of the set has 100 elements, each pair of the set shares 50 elements, each three of the sets share 25 elements, and there are 5 elements in all four sets?

Solution:

Let the four sets be A, B, C, and D. then we have $|A| = |B| = |C| = |D| = 100$, $|A \cap B| = |A \cap C| = |A \cap D| = |B \cap C| = |B \cap D| = |C \cap D| = 50$, $|A \cap B \cap C| = |A \cap B \cap D| = |A \cap C \cap D| = |B \cap C \cap D| = 25$ and $|A \cap B \cap C \cap D| = 5$. Principle of inclusion exclusion shows that

$|A \cup B \cup C \cup D| = |A| + |B| + |C| + |D| - |A \cap B| - |A \cap C| - |A \cap D| - |B \cap C| - |B \cap D| - |C \cap D| + |A \cap B \cap C| + |A \cap B \cap D| + |A \cap C \cap D| + |B \cap C \cap D| - |A \cap B \cap C \cap D|$

$=100 + 100 + 100 + 100 - 50 - 50 - 50 - 50 - 50 - 50 + 25 + 25 + + 25 + 25 - 5 = 195.$

Alternative form of Inclusion – Exclusion

Let A_i be the subset containing the elements that have property p_i. The number of elements with all the properties $P_{i1}, P_{i2}, \ldots, P_{ik}$ will be denoted by $N(P_{i1}\ P_{i2} \ldots P_{ik})$. In set notation we can write these quantities as

$|A_{i1} \cap A_{i2} \cap \ldots \cap A_{ik}| = N(P_{i1}\ P_{i2} \ldots P_{ik})$.

If the number of elements with none of the properties is denoted by $N(P'_1\ P'_2 \ldots P'_n)$ and the number of elements in the set is denoted by N. we have

$N(P'_1\ P'_2 \ldots P'_n) = N - |A_1 \cup A_2 \cup \ldots \cup A_n|$ Using inclusion exclusion principle, we have

$$N(P'_1\ P'_2 \ldots P'_n) = N - \sum_{1\le i\le n} N(P_i) + \sum_{1\le i<j\le n} N(P_iP_j) - \sum_{1\le i<j<k\le n} N(P_iP_jP_k) + \ldots + (-1)^n N(P_1P_2\cdots P_n).$$

Example:

Find the number of solutions of the equation $x_1 + x_2 + x_3 = 13$, where x_1, x_2 and x3 are nonnegative integer less than 6.

Solution:

Let properties P_1 be $x_1 \ge 6$, P_2 be $x_2 \ge 6$ and P_2 be $x_3 \ge 6$ then the number of solutions of the equation $x_1 + x_2 + x_3 = 13$, where x_1, x_2 and x3 are nonnegative integer less than 6 is

$N(P'_1\ P'_2\ P'_3) = N - N(P_1) - N(P_2) - N(P_3) + N(P_1P_2) + N(P_1P_3) + N(P_2P_3) - N(P_1P_2P_3)$

Now, the equation can be viewed as selecting 13 items where there are x_1 items of type one, x_2 items of type two, and x_3 items of type three. Here we can use the same numbers as much as we can from a set with three numbers (repetition is allowed) so we have,

$C(3 + 13 - 1, 13) = C(15,13) = 15!/(2!13!) = 15.14/2 = 105$ solutions i.e. $N = 105$.

Similarly, $N(P_1) = N(P_2) = N(P_3) = C(3 + 7 - 1,7) = C(9,7) = 36$. [Here for $x_i \ge 6$ we can select 6 items of the particular type and remaining 7 items are selected to have 13 items from other types.]. Number of solutions with $x_1 \ge 6$ and $x_2 \ge 6$ is $C(3 + 1 - 1,1) = 3$, number of solutions with $x_1 \ge 6$ and $x_3 \ge 6$ is $C(3 + 1 - 1,1) = 3$, number of solutions with $x_2 \ge 6$ and $x_3 \ge 6$ is $C(3 + 1 - 1,1) = 3$, and number of solutions with $x_1 \ge 6$, $x_2 \ge 6$ and $x_3 \ge 6$ is 0. so using above formula we have $N(P'_1\ P'_2\ P'_3) = 105 - 3.36 + 3.3 - 0 = 6$.

Applications

Example 1:(Sieve of Eratosthenes)

Find the number of primes less than 200 using the principle of inclusion exclusion.

Solution:

We know that the composite number not exceeding 200 has prime factor not exceeding $\sqrt{200} = 14.14$ i.e. not exceeding 14. Primes less than 14 are 2, 3, 5, 7, 11, and 13. Now we can say that the above 6 primes and all other numbers greater than 1 not exceeding 200 and not divisible by above 6 primes are primes. So if we take properties, P_1 that a number is divisible by 2, P_2 that a number is divisible by 3, P_3 that a number is divisible by 5, P_4 that a number is divisible by 7, P_5 that a number is divisible by 11, and P_6 that a number is divisible by 13, then prime numbers not exceeding 200 is $6 + N(P'_1 P'_2 P'_3 P'_4 P'_5 P'_6)$. We know that there are 199 integers not exceeding 200 and greater than 1 we have by principle of inclusion exclusion

$N(P'_1 P'_2 P'_3 P'_4 P'_5 P'_6) =$

$199 - N(P_1) - N(P_2) - N(P_3) - N(P_4) - N(P_5) - N(P_6) + N(P_1P_2) + N(P_1P_3) + N(P_1P_4) + N(P_1P_5) + N(P_1P_6) + N(P_2P_3) + N(P_2P_4) + N(P_2P_5) + N(P_2P_6) + N(P_3P_4) + N(P_3P_5) + N(P_3P_6) + N(P_4P_5) + N(P_4P_6) + N(P_5P_6) - N(P_1P_2P_3) - N(P_1P_2P_4) - N(P_1P_2P_5) - N(P_1P_2P_6) - N(P_1P_3P_4) - N(P_1P_3P_5) - N(P_1P_3P_6) - N(P_1P_4P_5) - N(P_1P_4P_6) - N(P_1P_5P_6) - N(P_2P_3P_4) - N(P_2P_3P_5) - N(P_2P_3P_6) - N(P_2P_4P_5) - N(P_2P_4P_6) - N(P_2P_5P_6) - N(P_3P_4P_5) - N(P_3P_4P_6) - N(P_3P_5P_6) - N(P_{34}P_5P_6) + N(P_1P_2P_3P_4) + N(P_1P_2P_3P_5) + N(P_1P_2P_3P_6) + N(P_1P_2P_4P_5) + N(P_1P_2P_4P_6) + N(P_1P_2P_5P_6) + N(P_1P_3P_4P_5) + N(P_1P_3P_4P_6) + N(P_1P_3P_5P_6) + N(P_1P_4P_5P_6) + N(P_2P_3P_4P_5) + N(P_2P_3P_4P_6) + N(P_2P_3P_5P_6) + N(P_2P_4P_5P_6) + N(P_3P_4P_5P_6) - N(P_1P_2P_3P_4P_5) - N(P_1P_2P_3P_4P_6) - N(P_1P_2P_3P_5P_6) - N(P_1P_2P_4P_5P_6) - N(P_1P_3P_4P_5P_6) - N(P_2P_3P_4P_5P_6) + N(P_1P_2P_3P_4P_5\,P_6)$

$$= 199 - \left\lfloor\frac{200}{2}\right\rfloor - \left\lfloor\frac{200}{3}\right\rfloor - \left\lfloor\frac{200}{5}\right\rfloor - \left\lfloor\frac{200}{7}\right\rfloor - \left\lfloor\frac{200}{11}\right\rfloor - \left\lfloor\frac{200}{13}\right\rfloor + \left\lfloor\frac{200}{2.3}\right\rfloor + \left\lfloor\frac{200}{2.5}\right\rfloor + \left\lfloor\frac{200}{2.7}\right\rfloor +$$

$$\left\lfloor\frac{200}{2.11}\right\rfloor + \left\lfloor\frac{200}{2.13}\right\rfloor + \left\lfloor\frac{200}{3.5}\right\rfloor + \left\lfloor\frac{200}{3.7}\right\rfloor + \left\lfloor\frac{200}{3.11}\right\rfloor + \left\lfloor\frac{200}{3.13}\right\rfloor + \left\lfloor\frac{200}{5.7}\right\rfloor + \left\lfloor\frac{200}{5.11}\right\rfloor + \left\lfloor\frac{200}{5.13}\right\rfloor + \left\lfloor\frac{200}{7.11}\right\rfloor +$$

$$\left\lfloor\frac{200}{7.13}\right\rfloor + \left\lfloor\frac{200}{11.13}\right\rfloor - \left\lfloor\frac{200}{2.3.5}\right\rfloor - \left\lfloor\frac{200}{2.3.7}\right\rfloor - \left\lfloor\frac{200}{2.3.11}\right\rfloor - \left\lfloor\frac{200}{2.3.13}\right\rfloor - \left\lfloor\frac{200}{2.5.7}\right\rfloor - \left\lfloor\frac{200}{2.5.11}\right\rfloor - \left\lfloor\frac{200}{2.5.13}\right\rfloor -$$

$$\left\lfloor \frac{200}{2.7.11} \right\rfloor - \left\lfloor \frac{200}{2.7.13} \right\rfloor - \left\lfloor \frac{200}{2.11.13} \right\rfloor - \left\lfloor \frac{200}{3.5.7} \right\rfloor - \left\lfloor \frac{200}{3.5.11} \right\rfloor - \left\lfloor \frac{200}{3.5.13} \right\rfloor - \left\lfloor \frac{200}{3.7.11} \right\rfloor - \left\lfloor \frac{200}{3.7.13} \right\rfloor -$$

$$\left\lfloor \frac{200}{3.11.13} \right\rfloor - \left\lfloor \frac{200}{5.7.11} \right\rfloor - \left\lfloor \frac{200}{5.7.13} \right\rfloor - \left\lfloor \frac{200}{5.11.13} \right\rfloor - \left\lfloor \frac{200}{7.11.13} \right\rfloor + \left\lfloor \frac{200}{2.3.5.7} \right\rfloor + \left\lfloor \frac{200}{2.3.5.11} \right\rfloor +$$

$$\left\lfloor \frac{200}{2.3.5.13} \right\rfloor + \left\lfloor \frac{200}{2.3.7.11} \right\rfloor + \left\lfloor \frac{200}{2.3.7.13} \right\rfloor + \left\lfloor \frac{200}{2.3.11.13} \right\rfloor + \left\lfloor \frac{200}{2.5.7.11} \right\rfloor + \left\lfloor \frac{200}{2.5.7.13} \right\rfloor +$$

$$\left\lfloor \frac{200}{2.5.11.13} \right\rfloor + \left\lfloor \frac{200}{2.7.11.13} \right\rfloor + \left\lfloor \frac{200}{3.5.7.11} \right\rfloor + \left\lfloor \frac{200}{3.5.7.13} \right\rfloor + \left\lfloor \frac{200}{3.5.11.13} \right\rfloor + \left\lfloor \frac{200}{3.7.11.13} \right\rfloor +$$

$$\left\lfloor \frac{200}{5.7.11.13} \right\rfloor - \left\lfloor \frac{200}{2.3.5.7.11} \right\rfloor - \left\lfloor \frac{200}{2.3.5.7.13} \right\rfloor - \left\lfloor \frac{200}{2.3.5.11.13} \right\rfloor - \left\lfloor \frac{200}{2.3.7.11.13} \right\rfloor - \left\lfloor \frac{200}{2.5.7.11.13} \right\rfloor -$$

$$\left\lfloor \frac{200}{3.5.7.11.13} \right\rfloor + \left\lfloor \frac{200}{2.3.5.7.11.13} \right\rfloor$$

$= 199 – (100 + 66 + 40 + 28 + 18 + 15) + (33 + 20 + 14 + 9 + 7 + 13 + 9 + 6 + 5 + 5 + 3 + 3 + 2 + 2 + 1) – (6 + 4 + 3 + 2 + 2 + 1 + 1 + 1 + 1 + 0 + 1 + 1 + 1 + 0 + 0 + 0 + 0 + 0 + 0 + 0) + all others will be 0.$

$= 199 – 267 +132 – 24 = 40.$

Now the total number of primes not exceeding 200 is 6 + 40 = 46.

Example 2:(The Number of Onto Functions)

How many onto functions are there from a set with seven elements to one with five elements?

Solution:

Suppose that the elements in codomain be a_i, i = 1 …5. Let P_i, i =1…5 be the properties that a_i's for i = 1, 2, …, 5 are not in the range of the functions, respectively. To have a number of onto function if we can exclude the functions holding the above properties we are done. So, using principle of inclusion exclusion we have,

$$N(P'_1P'_2P'_3P'_4P'_5)=N - \sum_{1\le i\le n} N(P_i) + \sum_{1\le i<j\le n} N(P_iP_j) - \sum_{1\le i<j<k\le n} N(P_iP_jP_k) + \sum_{1\le i<j<k<l\le n} N(P_iP_jP_kP_l) - N(P_1P_2P_3P_4P_5).$$

In the formula above,

N, total number of functions = 5^7.

We can select any one element of codomain to be not in the range then selecting this

element can be done in C(5,1) ways. There are 4 elements left in the codomain after not taking one element for function then for each such a removal of elements from range we can make 4^7 numbers of total functions. So total number of functions with one element missing from the range is $\sum_{1\le i\le n} N(P_i) = C(5,1)\ 4^7$.

We can select any two elements of codomain to be not in the range then selecting this element can be done in C(5,2) ways. There are 3 elements left in the codomain after not taking two elements for function then for each such a removal of elements from range we can make 3^7 numbers of total functions. So total number of functions with one element missing from the range is $\sum_{1\le i<j\le n} N(P_iP_j) = C(5,2)\ 3^7$.

Using similar reason as above,

$$\sum_{1\le i<j\le k\le n} N(P_iP_jP_k) = C(5,3)\ 2^7.$$

$$\sum_{1\le i<j\le k\le l\le n} N(P_iP_jP_kP_l) = C(5,4)\ 1^7.$$

$$N(P_1P_2P_3P_4P_5) = 0.$$

Then we have total number of onto functions as 5^7 - C(5,1) 4^7 + C(5,2) 3^7 - C(5,3) 2^7 + C(5,4) 1^7 – 0 = 78125 – 81920 + 21870 – 1280 + 5 = 16800.

Example 3:(Derangements)

The permutations of n elements that leave no objects in their original position are called derangements.

How many derangements are there of a set with four elements?

Solution:

Let a permutation have property P_i if it fixes an element i. Then we can say that the number of derangements is the number of permutations having none of the properties P_i for i = 1, 2, 3, 4. Then we can write $N(P'_1P'_2P'_3P'_4)$, using principle of inclusion exclusion, equals to $N - \sum_{1\le i\le n} N(P_i) + \sum_{1\le i<j\le n} N(P_iP_j) - \sum_{1\le i<j<k\le n} N(P_iP_jP_k) + N(P_1P_2P_3P_4)$., where

N is the total number of permutations i.e. 4! = 24. If one element is fixed some where in

C(4,1) place then total number of such permutations would be $\sum_{1\le i\le n} N(P_i)$ = C(4,1) 3! = 24.

If two elements are fixed some where in C(4,2) places then the total number of such permutations would be $\sum_{1\le i<j\le n} N(P_iP_j)$ = C(4,2) 2! = 12.

Similarly, $\sum_{1\le i<j\le k\le n} N(P_iP_jP_k)$ = C(4,3)1! = 4 and

$N(P_1P_2P_3P_4)$ = C(4,4)0! =1. Hence total number of derangements is

24 – 24 + 12 – 4 +1 = 9.

Self Studies

Read chapter 6 of your textbook such that you can cover all the read materials in the class.

[Relations]

Discrete Structures (CSc 511)

Samujjwal Bhandari

Central Department of Computer Science and Information Technology (CDCSIT)

Tribhuvan University, Kirtipur,

Kathmandu, Nepal.

Representing Relations

Listings of ordered pairs (read yourself)

Matrix Representation

The relation with finite sets can be represented using the matrix (zero one matrix). Let A be a set (a_1, a_2, …a_n) and B be the set (b_1, b_2, …, b_m), where elements are listed in some arbitrary order we represent relation from A to B by matrix $M_R = [m_{ij}]$, where

$$m_{ij} = \begin{cases} 1 \text{ if } (a_i, b_j) \in R. \\ 0 \text{ if } (a_i, b_j) \notin R. \end{cases}$$

Example:

Represent the relation {(1,1), (1,2), (1,3), (2,2), (2,3), (3,2), (3,3)} on the set {1,2,3) with matrix, where elements of the set is listed in increasing order.

Solution:

$$M_R = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix}$$

Identifying properties

Reflexive: If all the diagonal elements are 1 i.e. all $m_{ij} = 1$ whenever i = j, then the relation represented by the matrix is reflexive **(is above matrix reflexive? Yes)**.

Symmetric: If $m_{ij} = 1$ in the matrix then $m_{ji} = 1$ must be true and if $m_{ij} = 0$ then $m_{ji} = 0$ is also true. (it means that the relation represented by a matrix is symmetric if and only if the matrix is equal to its transpose). **(What about above matrix? No)**.

Antisymmetric: If $m_{ij} = 1$ and $i \neq j$, then $m_{ji} = 0$ or, in other words, either $m_{ij} = 0$ or $m_{ji} = 0$ when $i \neq j$. **(What about above matrix? No)**.

Note: Read composition of relations

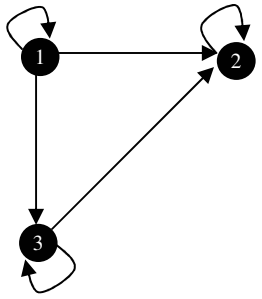
Directed Graph Representation

A directed graph, or digraph is a set of vertices V together with the set of edges. The vertex a is called initial vertex of the edge (a, b), and the vertex b is called the terminal vertex of this edge.

Example:

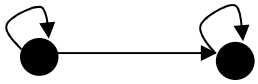
Draw the directed graph for the relation given in above example (example in matrix representation relation is {(1,1), (1,2), (1,3), (2,2), (2,3), (3,2), (3,3)}).

Solution:

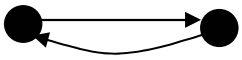


Identifying properties

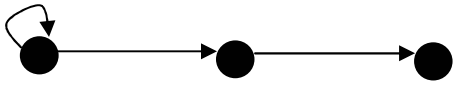
Reflexive: If every vertex has edge from the vertex to itself.



Symmetric: If for every edge of one direction there is another edge in opposite direction joining same two vertices as of first edge.



Antisymmetric: If no two distinct vertices have an edge going in both directions.



Closures of Relations

Let R be a relation on a set A. R may or may not have some property P, like symmetry, reflexivity, etc. If there is a relation S with property P containing R such that S is a subset of every relation with property P containing R, then S is called the closure of R with respect to P.

Reflexive closure

For any relation R on A, reflexive closure of R is formed by adding to R all pairs of the form (a, a) with $a \in A$, not already in R i.e. $R \cup D$, where $D = \{(a, a) \mid a \in A\}$ is the diagonal relation on A.

Example:

Let R be the relation on the set {1, 2, 3, 4} containing the ordered pairs (1,2), (1,3), (2,2), (2, 4), (3,1), (3,2), (3, 4), and (4, 4) find reflexive closure.

Solution:

We have R = {(1,2), (1,3), (2,2), (2, 4), (3,1), (3,2), (3, 4), (4, 4)} and

D = {(1,1), (2,2), (3,3), (4,4)}

So, $R \cup D$ = {(1,1), (1,2), (1,3), (2,2), (2, 4), (3,1), (3,2), (3,3), (3, 4), (4, 4)} is the reflexive closure of R.

Symmetric closure

If all ordered pairs of the form (b, a) is added to the relation R on A, where (a, b) is in the relation, that are not already in R, then the newly formed set after addition is symmetric closure of R i.e. for the relation R on A $R \cup R^{-1}$ is symmetric closure of relation R, where $R^{-1} = \{(b, a) \mid (a, b) \in A\}$.

Example:

Find symmetric closure for the relation given in above example.

Solution:

We have R = {(1,2), (1,3), (2,2), (2, 4), (3,1), (3,2), (3, 4), (4, 4)} and

R^{-1} = {(2,1), (3,1), (4,2), (1,3), (2,3), (4,3)}

So, $R \cup R^{-1}$ = {(1,2), (1,3), (2,1), (2,2), (2,3), (2,4), (3,1), (3,2), (3,4), (4,2), (4,3), (4,4)} is the symmetric closure of R.

Transitive closure

A **Path** from a to b in the directed graph G is a sequence of edges (x_0, x_1), (x_1, x_2), … (x_{n-1}, x_n), in G, where n is a nonnegative integer, and x_0 =a and x_n = b. In other words a sequence of edges where the terminal vertex of one edge is same as the initial vertex in next edge in the path. This path is denoted by $x_0, x_1, \ldots, x_n$ and has length n. A path of length $n \geq 1$ that begins and ends at the same vertex is called a circuit or cycle.

Let R be a relation n a set A. The connectivity relation R* consists of the pairs (a,b) such that there is a path of at least one from a to b in R. In notation we write

$$R^* = \bigcup_{n=1}^{\infty} R^n.$$

Theorem 1: (Prove yourself)

The transitive closure of a relation R equals the connectivity relation R*.

Theorem 2:

Let M_R be the zero-one matrix of the relation R on a set with n elements. Then the zero-one matrix of the transitive closure R* is

$M_{R^*} = M_R \vee M_R^{[2]} \vee M_R^{[3]} \vee \ldots \vee M_R^{[n]}$.

Suggestion: Try to read an algorithm using the above theorem.

Warshall's algorithm

Given a set S with n elements $v_1, v_2, \ldots, v_n$ where elements are listed in arbitrary order and R is a relation on a set S. If a, $x_1, x_2, \ldots, x_{m-1}$, b is a path, then $x_1, x_2, \ldots, x_{m-1}$ are called interior vertices of the given path. More clearly, all the vertices of a path that appears somewhere but not as a first or last vertices in a path are interior vertices (If first vertex is met again in the path then it can be interior vertex, similarly if the last vertex has been met already then it can be interior vertex). Warshall's algorithm uses the concept of interior vertices of a path.

Lemma 1:

Let $W_k = [w_{ij}^{[k]}]$ be the zero one matrix that has a 1 in its $(i, j)^{th}$ position if and only if there is a path from v_i to v_j with interior vertices from the set $\{v_1, v_2, \ldots, v_k\}$. Then

$w_{ij}^{[k]} = w_{ij}^{[k-1]} \vee (w_{ij}^{[k-1]} \wedge w_{ij}^{[k-1]})$, whenever i, j, and k are positive integers not exceeding n.

Algorithm:

Warsahll(W: n by n 0-1 matrix)

{

for(k =0;k<n;k++)

for(i =0;i<n;i++)

for(j =0;j<n;j++)

$w_{ij} = w_{ij} \vee (w_{ik} \wedge w_{kj})$

//W = [w_{ij}] is the result

}

What is the complexity of above algorithm?

Equivalence Relations

Equivalence Relations

A relation R on set A is called equivalence relation if it satisfies the three properties namely, reflexive, symmetric, and transitive. The two elements related by equivalence relations are called equivalent.

Equivalence Classes

Given R, an equivalence relation on set A, the set of all elements that are related to an element a of A is called the equivalence class of a. The equivalence class of a with respect to R is denoted by $[a]_R$ or [a] when only one relation is in consideration. In notational term we can write $[a]_R = \{b \mid (a,b) \in R\}$. If $c \in [a]_R$, then c is called a representative of equivalence class $[a]_R$.

Equivalence Classes and Partitions

Given a set A, a partition of A is a collection P of disjoint subsets whose union is A i.e.

$\forall$ B $\in$ P, B $\subseteq$ A;

$\forall$ B,C $\in$ P, B $\cap$ C = ϕ, or B = C; and

$\forall$ x $\in$ A $\exists$ B $\in$ P such that x $\in$ B;

Theorem 1:

Let R be an equivalence relation on a set S. Then the equivalence classes of R form a partition of S. Conversely, given a partition $\{A_i \mid i \in I\}$ of the set S, there is an equivalence relation R that has the sets Ai, $i \in I$, as its equivalence classes.

Example:

Define the relation R on the set A of positive integers by $(a,b) \in R$ iff a/b can be expressed in the form 2^m, where m is an arbitrary integer.

a) Show that R is an equivalence relation.

b) Determine the equivalence classes under R.

Solution:

a)

For all $a \in A$, $a/a = 1 = 2^0$, where m = 0 hence R is reflexive. If $(a,b) \in R$ then we have $a/b = 2^m$ such that $b/a = 2^{-m}$. So we have $(b,a) \in R$ whenever $(a,b) \in R$ hence R is symmetric. Take $(a,b) \in R$ and $(b, c) \in R$ then we can write $a/b = 2^m$ and $b/c = 2^n$ so we have $a/c = 2^{m+n}$, where m+n is an arbitrary integer so that $(a,c) \in R$ hence R is transitive. For the facts above we showed that R is an equivalence relation.

b)

Some equivalence classes may be

[1] = {1,2,4,8,… …}

[3] = {3,6,12,24,48, … …}

[5] = {5,10,20,40, … …}

[7] = {7,14,28,56, … …}

… …

We can generalize these classes as

$[p] = \{2^m * p \mid m$ is nonnegative integers and p is either 1 or prime greater than 2$\}$.

Partial Orderings

A relation R on a set A is called **partial order** or partial ordering if it is reflexive, antisymmetric, and transitive. A set A together with the partial order R is called a partially ordered set, or a poset, denoted by (A, R).

Example:

Show that a relation ≤ "less than or equal" is partial order on the set of integers.

Solution:

We know $\forall a \in Z$, $a \leq a$, hence ≤ is reflexive. If $a \leq b$ and $b \leq a$, then $a = b$, hence ≤ is antisymmetric, and if $a \leq b$ and $b \leq c$, then $a \leq c$, hence ≤ is transitive. It follows that ≤ is a partial ordering on the set of integers Z and (Z, ≤) is a poset.

In a poset (S, R), the elements a and b of a poset are comparable if either aRb or bRa. When neither aRb nor bRa, then a and b are incomparable. If every two elements of a set S are comparable, then S is called a **totally ordered** or **linearly ordered set** or a **chain**, and R is called **total order** or a linear order.

(S,R) is a **well ordered set** if it is a poset such that R is a total order and such that every nonempty subset of S has a least element.

Lexicographic order

Given two posets (A_1,L_1) and (A_2,L_2). The lexicographic ordering L on $A_1 \times A_2$ is defined by specifying that one pair is less than a second pair i.e. $(a_1, a_2)L(b_1, b_2)$ if $a_1L_1b_1$ or both $a_1 = b_1$ and $a_2L_2b_2$. Similarly we can extend this definition to Cartesian product or more than two sets.

Lexicographic Ordering of Strings

Suppose $a_1a_2 \ldots a_m$ and $b_1b_2 \ldots b_n$ are strings on a partially ordered set S and suppose two strings are not equal. If t is the minimum of m and n, then we define lexicographic ordering that the string $a_1a_2 \ldots a_m$ is less than $b_1b_2 \ldots b_n$ if and only if

$(a_1a_2 \ldots a_t) < (b_1b_2 \ldots b_t)$, or

$(a_1a_2 \ldots a_t) = (b_1b_2 \ldots b_t)$ and $m < n$.

Example #1:

Find the lexicographic ordering of the 3 tuples (1,1,2), (1,2,1)

Solution:

(1,1,2) < (1,2,1) because first elements of both the 3 tuples are same so if we look forward then the second element of first 3 tuple is less than the second element of second 3 tuple.

Example #2:

Find the lexicographic ordering of the strings *quack, quick, quicksilver, quicksand, quacking*.

Solution:

quack < *quick* [strings differ in third position and a < i]

quick < *quicksilver* [letter of the first string agrees but second string is longer]

quicksand < *quicksilver* [strings differ in seventh position and a < i]

quacking < *quicksand* [strings differ in third position and a < i]

Similarly others are

quack < *quacking, …..*

So the ordering is

quack < *quacking* < *quick* < *quicksand* < *quicksilver*

Hasse Diagrams

A partial ordering on a finite set can be represented using the pictorial notation as follows:

Construct the directed graph of a relation.

Remove all the loops (since it is clear that partial order is reflexive so every vertex has a loop)

Remove all the edges that is requires due to the transitivity

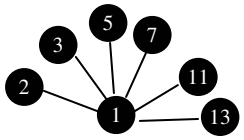
Arrange each edge so that its initial vertex is below its terminal vertex and remove all arrows from the edges since edges point in upward direction only.

The diagram formed using above steps contains sufficient information to find the partial ordering. This diagram is called **Hasse diagram**.

Example #1:

Draw the Hasse diagram for divisibility on the set {1,2,3,5,7,11,13}

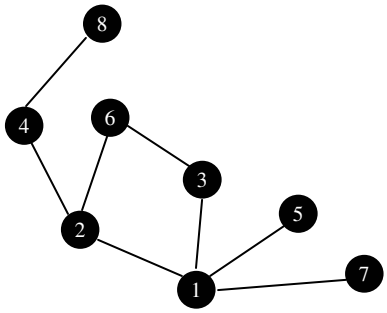
Solution:



Example #2:

Draw the Hasse diagram for divisibility on the set {1,2,3,4,5,6,7,8}

Solution:



Maximal and Minimal Elements

Some Definitions:

Maximal Elements: An element of a poset is maximal if it is not less than any elements of the poset i.e. a is maximal in the poset (S,≤) if there is no b ∈ S such that a < b. In Hasse diagram maximal elements are top elements.

Minimal Elements: An element of a poset is minimal if it is not greater than any elements of the poset i.e. a is minimal in the poset (S,≤) if there is no b ∈ S such that b < a. In Hasse diagram minimal elements are bottom elements.

Greatest Element: An element in a poset that is greater than every other element i.e. a is the greatest element of the poset (S,≤) if b ≤ a for all b ∈ S. The greatest element may exist or may not exist in a poset.

Least Element: An element in a poset that is less than every other element i.e. a is the least element of the poset (S,≤) if $a \leq b$ for all $b \in S$. The least element may exist or may not exist in a poset.

Upper Bound: If u is an element of S such that $a \leq u$ for all elements $a \in A$, then u is called an upper bound of A, where $A \subseteq S$.

Lower Bound: If l is an element of S such that $l \leq a$ for all elements $a \in A$, then l is called a lower bound of A, where $A \subseteq S$.

Least Upper Bound: l_u is the least upper bound of the subset A if $a \leq l_u$ whenever $a \in A$ and $l_u \leq u$ whenever u is an upper bound of A.

Greatest Lower Bound: g_l is the greatest lower bound of the subset A if g_l is a lower bound and $l \leq g_l$ whenever l is a lower bound of A.

Example:

Answer these questions for the poset ({3,5,9,15,24,45},|)

a) Find the maximal elements
b) Find the minimal elements
c) Is there a greatest element?
d) Is there a least element?
e) Find all upper bounds of {3,5}
f) Find the least upper bound of {3,5} if exists
g) Find all lower bounds of {15,45}
h) Find the greatest lower bound of {15,45}, if exists.

Solution:

a) 24 and 45 are maximal elements.
b) 3 and 5 are minimal elements
c) No greatest element since there is no element in a set is divisible by all the elements in a set.
d) No least element since no element in a set divides all the elements in a set.
e) Since 15 and 45 are divisible by both 3 and 5 and they are follow the relation we have 15 and 45 as upper bounds of {3,5}

f) Since among the upper bounds 15 is the least, 15 is least upper bound.

g) The elements 3, 5 , and 15 divides the elements 15 and 45. Hence 3, 5, and 15 are lower bounds

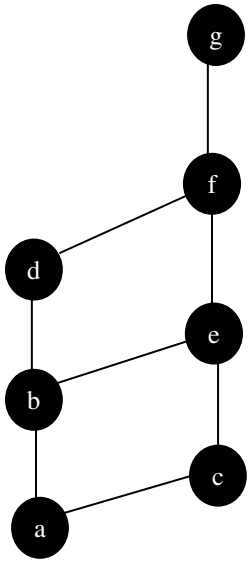
h) Amongst the lower bounds 15 is the greatest lower bound.

Lattices

A partially ordered set in which every pair of elements has both the least upper bound and the greatest lower bound is called **lattice**.

Example #1:

Identify whether the poset given by following Hasse diagram is lattice or not?



Solution:

Every pair of elements for e.g. (a,b), (a,c), (a,e), (b,e) …. Has least upper bound and greatest lower bound so the poset given by above Hasse diagram is a lattice.

Example #2:

Determine whether the poset (Z, ≥) is a lattice

Solution:

Suppose x and y are two integers. If we have a relation $x \geq y$ for all $x,y \in Z$, then we can

say that x is the least upper bound and y is the greatest lower bound. Conversely $y \geq x$ for all $x,y \in Z$, then we can say that y is the least upper bound and x is the greatest lower bound. These conditions hold for all elements in Z, hence poset $(Z, \geq)$ is a lattice.

Topological Sorting

A total ordering $\leq$ is called **compatible** with the partial ordering R if $a \leq b$ whenever aRb. Construction of a compatible total ordering from a partial ordering is called topological sorting.

Lemma 2:

Every finite nonempty poset $(S, \leq)$ has a minimal element.

Proof:

Take an element a_0 from the set S. if a_0 is not minimal, then there is an element a_1 with $a_1 < a_0$. If a_1 is not minimal there is an element a_2 with $a_2 < a_1$. this process is continued until all the elements in the set are tested for and if it goes up to nth element of the set which is the last one from the set then it will be the minimal element.

Algorithm:

```
toposort(S: finite poset)
{
k =1;
while (S ≠ ϕ)
{
        a_k = a minimal element of S   // from lemma 1 it is sure
        S = S – { a_k };
        k = k+1;
}
return {a_1, a_2, …, a_n} //compatible total ordering of S.
}
```

Example:

Find a compatible total order for the divisibility relation on the set {1,2,3,6,8,12,24,36}

Solution:

From the set we can identify that 1 is the minimal element so, 1 is selected. Next, minimal of {2,3,6,8,12,24,36} are 2 and 3, so we can choose any one element. Lets choose 2. Similarly, minimal of {3,6,8,12,24,36} is 3 and 8, so choose 8. Again select minimal element i.e. 3 from the set {3,6,12,24,36}. Continuing like this we have one of the total ordering as

$1 \prec 2 \prec 8 \prec 3 \prec 6 \prec 12 \prec 24 \prec 36.$

Self Studies

Read chapter 7 of your textbook such that you can cover all the read materials in the class.

[Graphs]

Discrete Structures (CSc 511)

Samujjwal Bhandari

Central Department of Computer Science and Information Technology (CDCSIT)

Tribhuvan University, Kirtipur,

Kathmandu, Nepal.

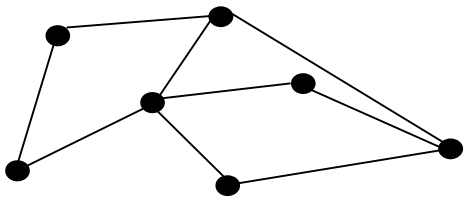
Graphs

Graph is a discrete structure with vertices and edges connecting the vertices. In this lecture we will discuss different types of graph and their applications.

Types of Graphs

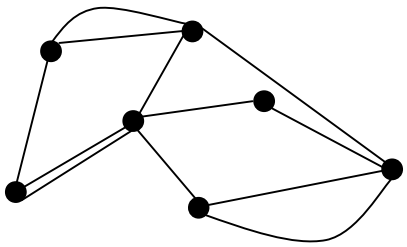
Simple Graph

We define a simple graph as 2 – tuple consists of a non empty set of vertices V and a set of unordered pairs of distinct elements of vertices called edges. We can represent graph as G = (V, E). This kind of graph has no loops and can be used for modeling networks that do not have connection to themselves but have both ways connection when two vertices are connected but no two vertices have more than one connection. The figure below is an example of simple graph.



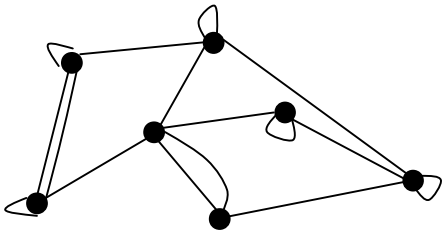
Multigraph

A multigraph G =(V, E) consists of a set of vertices V, a set of edges E, and a function *f* from E to $\{\{u, v\} | u, v \in V, u \neq v\}$. The edges e_1 and e_2 are called multiple or parallel edges if $f(e_1) = f(e_2)$. In this representation of graph also loops are not allowed. Since simple graph has single edges every simple graph is a multigraph. The figure below is an example of a multigraph.



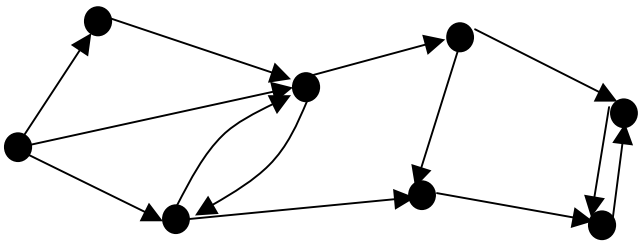
Pseudograph

A pseudograph G =(V, E) consists of a set of vertices V, a set of edges E, and a function *f* from E to {{u, v}|u, v ∈ V}. An edge is a loop if *f(e)* = {u, u} = {u} for some u ∈ V. The figure below is an example of a multigraph.



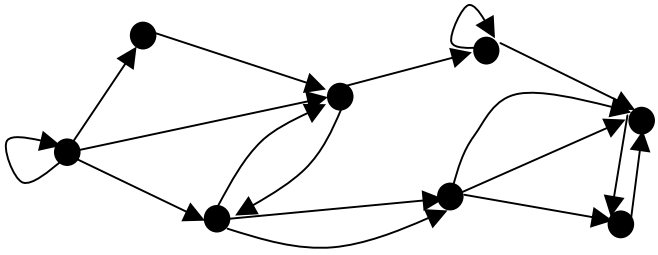
Directed Graph

A directed graph (V, E) consists of a set V of vertices, a set E of edges that are ordered pairs of elements of V. The below figure is a directed graph. In this graph loop is allowed but no two vertices van have multiple edges in same direction.



Directed Multigraph

A directed multigraph G =(V, E) consists of a set of vertices V, a set of edges E, and a function *f* from E to {(u, v)|u, v ∈ V}. The edges e_1 and e_2 are called multiple edges if $f(e_1) = f(e_2)$. The figure below is an example of a directed multigraph.



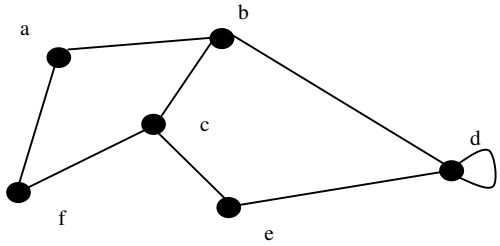
Terminologies

Two vertices u, v are adjacent vertices of a graph if {u, v} is an edge.

The edge e is called incident with the vertices u and v if e = {u, v}. This edge is also said to connect u and v. where u and v are end points of the edge.

Degree of a vertex in an undirected graph is the number of edges incident with it, except a loop at a vertex. Loop in a vertex counts twice to the degree. Degree of a vertex v is denoted by deg (v).A vertex of degree zero is called isolated vertex and a vertex with degree one is called pendant vertex.

Example: Find the degrees of the vertices in the following graph.



Solution:

deg(a) = deg(f) = deg(e) = 2 ; deg(b) = deg(c) = 3; deg(d) = 4

Theorem 1: The Handshaking Theorem

Let G = (V, E) be an undirected graph with e edges. Then $2e = \sum_{v \in V} \deg(v)$.

Theorem 2:

An undirected graph has an even number of vertices of odd degree.

Proof:

Take two sets of vertices, V_1, a set of vertices with even degree, and V_2, a set of vertices with odd degree. In an undirected graph G =(V, E) we have

$$2e = \sum_{v \in V} \deg(v) = \sum_{v \in V_1} \deg(v) + \sum_{v \in V_2} \deg(v).$$

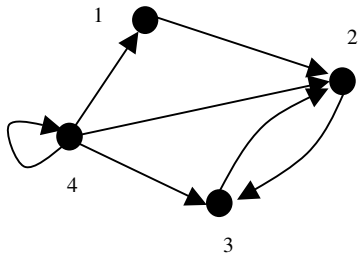
From the equality above we can say the left part is even i.e. 2e is even, the sum of deg(v) for $v \in V_1$ is even since every vertices has even degree. So for the left hand to be even sum of deg(v) for $v \in V_2$ must be even. Since all the vertices in the set V_2 have odd degree the number of such vertices must be even for the sum to be even. Hence proved.

Let (u, v) be an edge representing edge of a directed graph G. u is called adjacent to v and v is called adjacent from u. The vertex u is called **initial vertex** and the vertex v is called **terminal or end vertex**. Loop has same initial and terminal vertex.

In directed graph the **in-degree** of a vertex v, denoted by $\deg^-(v)$, is the number of edges that have v as their terminal vertex. The **out-degree** of a vertex v, denoted by $\deg^+(v)$, is the number of edges that have v as their initial vertex. Loop at a vertex adds up both in-degree and out-degree to one more than calculated in-degree and out-degree.

Example:

Find the in-degree and out-degree of each vertex in the following graph.



Solution:

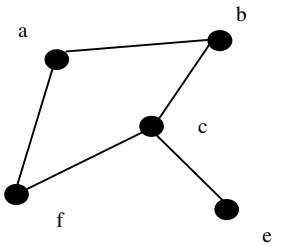
In-degrees of a graph are $\deg^-(1) = \deg^-(4) = 1$; $\deg^-(2) = 3$; $\deg^-(3) = 2$ and the out-degrees of a graph are $\deg^+(1) = \deg^+(2) = \deg^+(3) = 1$; $\deg^+(4) = 4$.

Theorem 3:

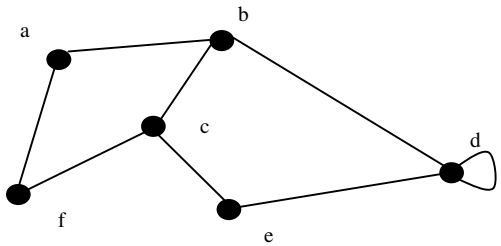
Let G(V, E) be a graph with directed edges. Then $\sum_{v \in V} \deg^-(v) = \sum_{v \in V} \deg^+(v) = |E|$.

A **subgraph** of a graph G =(V, E) is a graph H = (W, F) where $W \subseteq V$ and $F \subseteq E$.

Example:



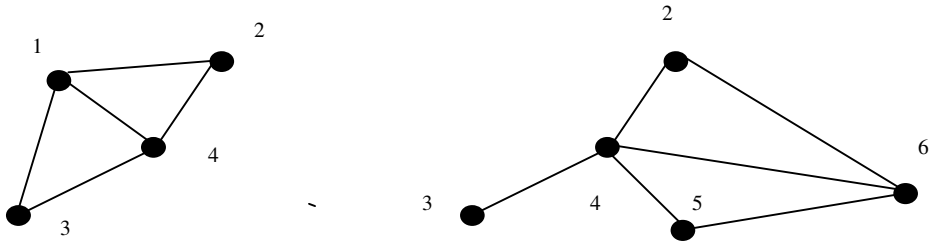
is a subgraph of



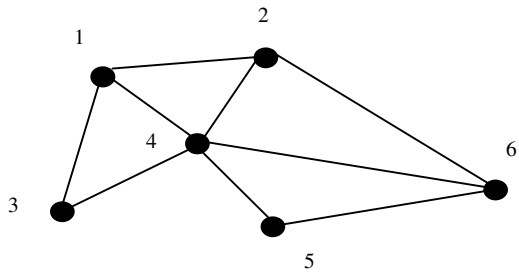
The **union** of two simple graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ is the simple graph with vertex set $V_1 \cup V_2$ and the edge set $E_1 \cup E_2$. The union of G_1 and G_2 is denoted by $G_1 \cup G_2$.

Example:

Find the union of two graphs given below.



Solution:



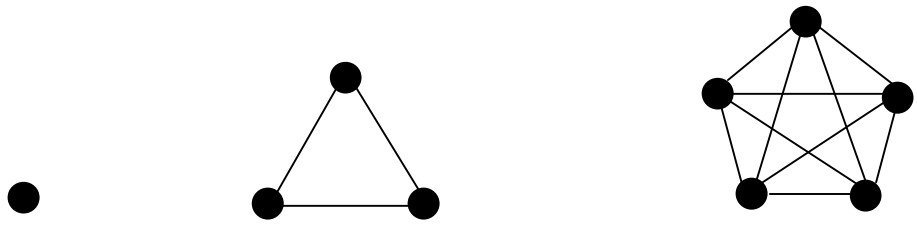
Complete Graphs

The complete graph of n vertices, denoted by K_n, is the simple graph that contains exactly one edge between each pair of distinct vertices.

Example:

What are K_1, K_3, and K_5?

Solution:



K_1 K_3 K_5

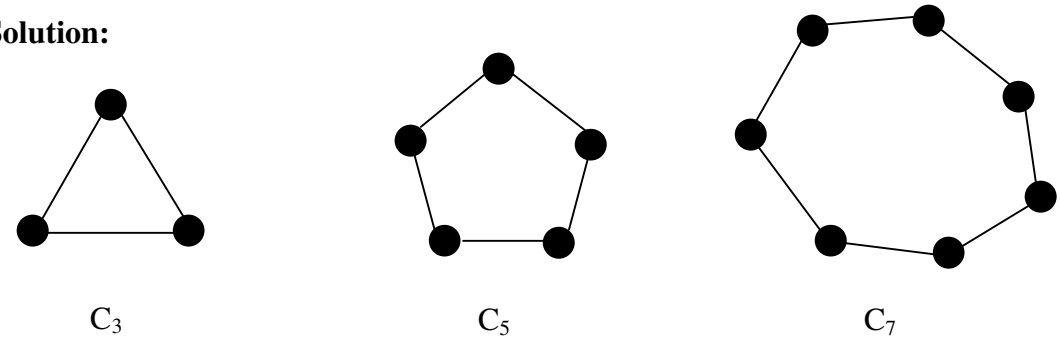
Cycles

The cycle C_n, $n \geq 3$, consists of n vertices $v_1, v_2, \ldots, v_n$ and edges $\{v_1, v_2\}, \{v_2, v_3\}, \ldots, \{v_{n-1}, v_n\}$, and $\{v_n, v_1\}$.

Example:

What are C_3, C_5, and C_7?

Solution:



C_3 C_5 C_7

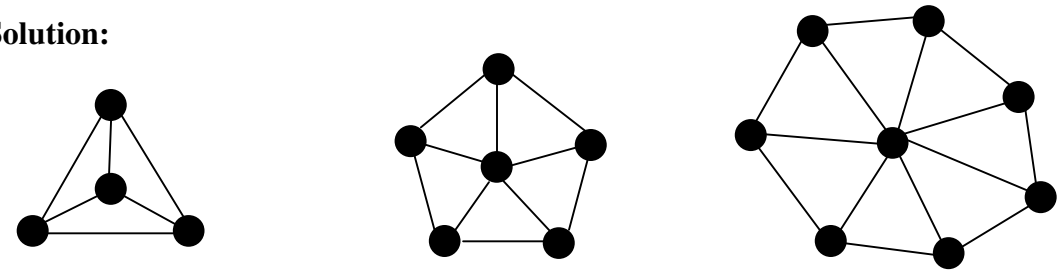
Wheels

The wheel W_n, for $n \geq 3$, is an union of C_n and additional vertex where the new vertex is connected by each vertex of the cycle.

Example:

What are C_3, C_5, and C_7?

Solution:



W_3 W_5 W_7

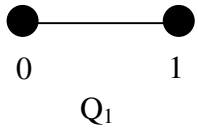
n- Cubes

The n-dimensional cube, or n-cube, denoted by Q_n, is the graph that has vertices representing the 2^n bit strings of length n. Two vertices are adjacent if and only if the bit strings that they represent differ in exactly one bit position.

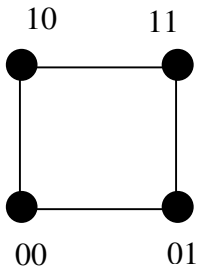
Example:

What are Q_1, Q_2, and Q_3?

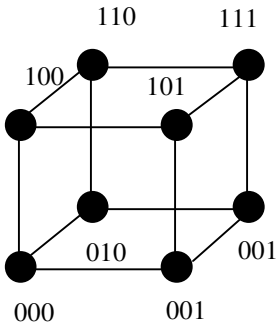
Solution:



Q_1



Q_2



Q_3

Bipartite Graphs

A simple graph G is bipartite if its vertex set V can be partitioned into two disjoint subsets V_1 and V_2 such that every edge in the graph connects a vertex from the set V_1 to the vertex of the set V_2. No two vertices of the same set are connected by an edge.

A graph is bipartite if and only if it is possible to color the vertices of the graph with at most two colors such that no two adjacent vertices have the same color.

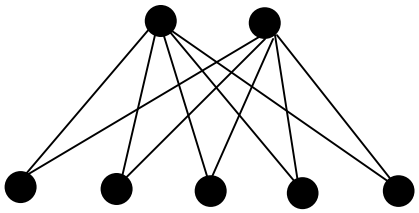
A graph is bipartite if and only if it is not possible to start at a vertex and return to this vertex by traversing an odd number of distinct edges.

Complete Bipartite Graphs: The complete bipartite graph $K_{m,n}$ is the graph where the vertex set is partitioned into two subsets of m and n vertices, respectively. In this graph there is an edge between two vertices if and only if two vertices are in different subsets of vertices.

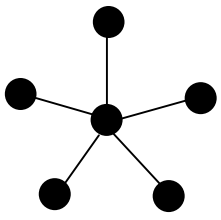
Example:

Sketch $K_{2,5}$, $K_{1,5}$.

Solution:



$K_{2,5}$



$K_{1,5}$

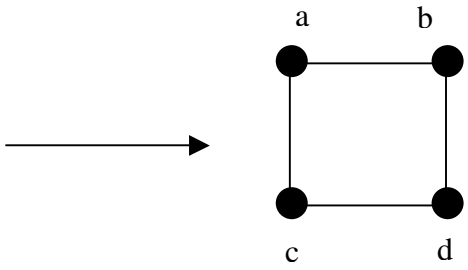
Graph Representations

Graph can be represented in many ways; one of the ways of representing a graph without multiple edges is by listing its edges. Some other ways are described below:

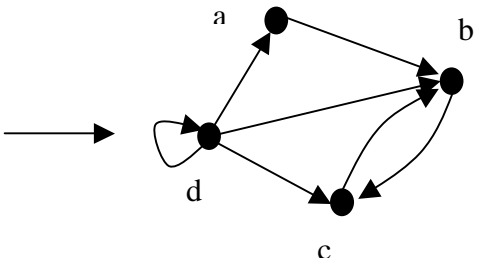
Adjacency List

This type of representation is suitable for the undirected graphs without multiple edges, and directed graphs. This representation looks as in the tables below.

Edge List for Simple Graph	
Vertex	Adjacent Vertices
a	b, c
b	a, d
c	a, d
d	b, c



Edge List for Directed Graph	
Initial Vertex	End Vertices
a	b
b	c
c	b
d	a, b, c, d



If we try to apply the algorithms of graph using the representation of graphs by lists of edges, or adjacency lists it can be tedious and time taking if there are high number of edges. For the sake of the computation, the graphs with many edges can be represented in other ways. In this class we discuss two ways of representing graphs in form of matrix.

Adjacency Matrix

Given a simple graph G =(V, E) with |V| = n. assume that the vertices of the graph are listed in some arbitrary order like v_1, v_2, …, v_n. The adjacency matrix A of G, with respect to the order of the vertices is n-by-n zero-one matrix (A = $[a_{ij}]$) with the condition, $a_{ij} = \begin{cases} 1 & \text{if } \{v_i, v_j\} \text{ is an edge of G,} \\ 0 & \text{otherwise.} \end{cases}$. Since there are n vertices and we may order vertices in any order there are n! possible order of the vertices. The adjacency matrix depends on the order of the vertices, hence there are n! possible adjacency matrices for a graph with n vertices.

Adjacency matrix for undirected graph is symmetric, in case of the pseudograph or multigraph the representation is similar but the matrix here is not zero-one matrix rather the $(i, j)^{th}$ entry of the matrix contains the number of edges appearing between that pair of vertices.

In case of the directed graph we can extend the same concept as in undirected graph as dictated by the relation $a_{ij} = \begin{cases} 1 & \text{if } (v_i, v_j) \text{ is an edge of G,} \\ 0 & \text{otherwise.} \end{cases}$. The main difference is that the matrix may not be symmetric.

If the number of edges is few then the adjacency matrix becomes sparse. Sometimes it will be beneficial to represented graph with adjacency list in such a condition.

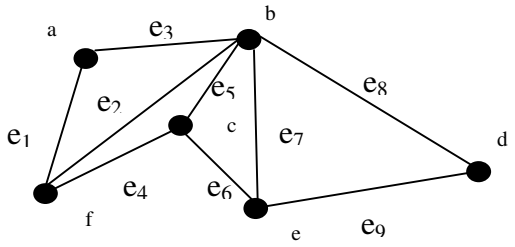
Incidence Matrix

This is another way of representing graph. Given an undirected graph G = (V, E). Assume that the vertices of the graph are v_1, v_2, …, v_n and the edges of the graph are e_1, e_2, …, e_m. the incidence matrix of a graph with respect to the above ordering of V and E is n-by-m matrix M = $[m_{ij}]$, where $m_{ij} = \begin{cases} 1 & \text{when edge } e_j \text{ is incident with } v_i, \\ 0 & \text{otherwise.} \end{cases}$.

When the graph is not simple then also the graph can be represented by using incidence matrix where multiple edges corresponds to two different columns with exactly same entries. Loops are represented with column with only one entry.

Example #1:

Represent the following graph using adjacency matrix and incidence matrix.



Solution:

Let the order of the vertices be a, b, c, d, e, f and edges order be e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_8, e_9.

$$\begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \end{bmatrix}$$

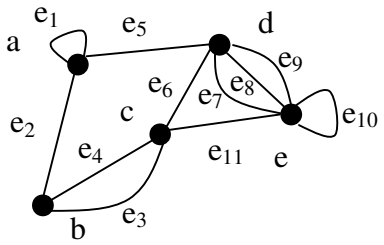
Adjacency Matrix

$$\begin{array}{c} \\ a \\ b \\ c \\ d \\ e \\ f \end{array} \begin{array}{c} \begin{array}{ccccccccc} e_1 & e_2 & e_3 & e_4 & e_5 & e_6 & e_7 & e_8 & e_9 \end{array} \\ \begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

Incidence Matrix

Example #2:

Represent the following graph using adjacency matrix and incidence matrix.



Solution:

Let the order of the vertices be a, b, c, d, e, f and edges order be e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_8, e_9, e_{10}, e_{11}.

$$\begin{bmatrix} 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 2 & 0 & 0 \\ 0 & 2 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 3 \\ 0 & 0 & 1 & 3 & 1 \end{bmatrix}$$

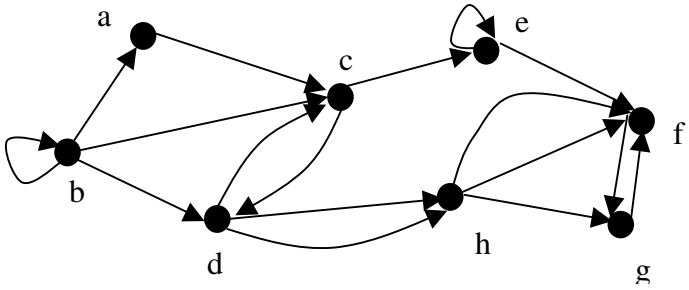
Adjacency Matrix

$$\begin{array}{c} \\ a \\ b \\ c \\ d \\ e \end{array} \begin{array}{c} \begin{matrix} e_1 & e_2 & e_3 & e_4 & e_5 & e_6 & e_7 & e_8 & e_9 & e_{10} & e_{11} \end{matrix} \\ \begin{bmatrix} 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \end{array}$$

Incidence Matrix

Example #3:

Represent the following directed graph using adjacency matrix.



Solution:

Let the order of the vertices be a, b, c, d, e, f, g and h. The adjacency matrix is

$$\begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 2 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2 & 1 & 0 \end{bmatrix}$$

Graph Isomorphism

Simple graphs G = (V, E) and H = (W, F) are isomorphic if there is a bijection function f from V to W with the property that a and b are adjacent in G if and only if f(a) and f(b) are adjacent in H, for all a and b in V. The bijection f is called an isomorphism.

Note: Extend the same concept for other types of graphs.

Determining whether two graphs are isomorphic or not is a difficult task since we can have n! possible one-to-one correspondence between two graphs with n vertices and testing for adjacency preservation between vertices is still cumbersome.

We generally can determine the two graphs to be not isomorphic by finding whether the graphs under consideration have the property needed for them to be isomorphic. Such a properties are called **invariant**.

Isomorphic simple graphs have same number of vertices (one-to-one correspondence between vertices of two graphs is required).

Isomorphic simple graphs have same number of edges (due to adjacency preservation).

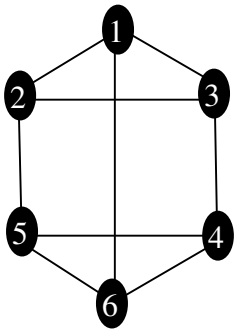
Degrees of the vertices in the isomorphic graphs must be same because the number of edges from that vertex is determined by degree.

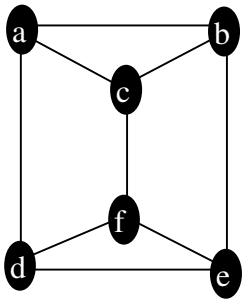
Existence of a simple circuit of length k, where k is a positive integer greater than 2, in both the graphs is an invariant.

The subgraphs formed by connecting the edges from the vertex with same degree in both the graphs must be isomorphic.

Example #1:

Determine whether the given two graphs are isomorphic or not?



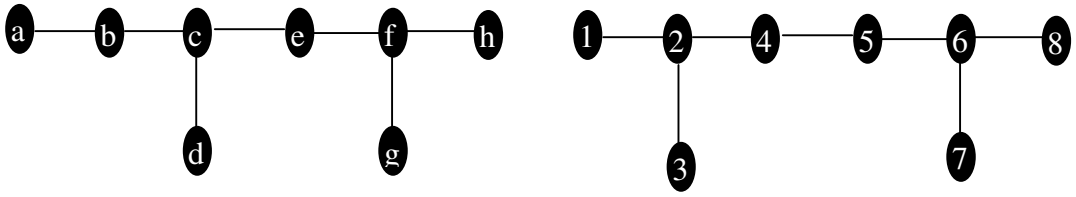


Solution:

In the above two graphs number of vertices in both graphs is same (i.e. 6), number of edges equal to 9 in both the graphs and all the vertices in both the graphs have degree 3. Since the invariants agree in both the graphs, we can try out to find the function that is isomorphism. Take the sequence of vertices from the first graph as 1, 2, 3, 4, 5, and 6. Now define f(1) = c, f(2) = a here there is adjacency preservation since we have {1, 2} as and edge in the first graph where as {f(1), f(2)} ={c, a} is an edge in the second graph. Similarly we can assign f(3) = b, f(4) = e, f(5) = d, f(6) = f. Since we found one to one correspondence between vertices of two graphs persevering the adjacency, the above two graphs are isomorphic. We can note that the adjacency matrices of two isomorphic graphs in which the vertices are ordered in terms of function i.e. in our example 1, 2, 3, 4, 5, and 6 for the first graph and c, a, b, e, d, and f in the second graph are same (verify!). *[In the above two graphs note that the number of circuits of same length in both the graphs is same.]*

Example #2:

Determine whether the given two graphs are isomorphic or not?



Solution:

In the above two graphs number of vertices in both graphs is 8, the number of edges equal to 7 in both the graphs, in both graphs two vertices have degree 3, 4 vertices have degree 1 and the remaining 2 vertices have degree 2. Since the invariants agree in both the graphs, we can continue to get the function such that it is isomorphism. However, in case of first graph the subgraph containing the vertex c (degree 3), with vertices a, b, c, d, and e is not isomorphic with any of the subgraph formed by connecting edges with vertex 2 or 6 (both of degree 3). Hence the two above graphs are not isomorphic.

Paths*

In an undirected graph G, a path of length n, where n is a nonnegative integer, from a to b in G is a sequence of n edges $e_1, e_2, \ldots, e_n$ of G such that $f(e_1) = \{x_0, x_1\}$, $f(e_2) = \{x_1, x_2\}$, …, $f(e_n) = \{x_{n-1}, x_n\}$, where $a = x_0$ and $b = x_n$. For the simple graph the path is denoted by a vertex sequence $x_0, x_1, \ldots, x_n$. If a = b and the path length is greater than zero, then the path is called circuit or cycle. The path or circuit is said to pass through the vertices $x_1, x_2, \ldots, x_{n-1}$ or traverse the edges $e_1, e_2, \ldots, e_n$. A path or circuit is simple if it does not contain the same edge more than once.

The similar definition can be provided to define the paths in directed multigraph, however the edges are instead ordered pairs in the directed multigraph.

Connectedness in undirected graphs*

An undirected graph is called connected if there is a path between every pair of distinct vertices of the graph. A graph that is not connected is the union of more than one connected graphs that do not share the common vertex. These disjoint connected subgraphs are called connected component of a graph.

Theorem 4:

There is a simple path between every pair of distinct vertices of a connected undirected graph.

Proof:

Suppose a and b are two distinct vertices of the connected undirected graph G =(V, E). we know that G is connected so by definition there is at least one path between a and b . Let $x_0, x_1, \ldots, x_n$, where $x_0 = a$ and $x_n = b$, be the vertex sequence of a path of a least length. Now if this path of the least length is not simple then we have $x_i = x_j$, for some i and j with $0 \leq i < j$. This implies that there is a path from a to b of shorter length with the vertex sequence $x_0, x_1, \ldots, x_i, \ldots, x_{j+1}, \ldots, x_n$ obtained by removing the edges corresponding to the vertex sequence $x_{i+1}, \ldots, x_j$. This shows that there is a simple path.

Cut vertices (**articulation points**) are those vertices in the graph whose removal along with the edges incident on them produces subgraph with more connected components than in the original graph.

Cut edge (**bridge**) is an edge whose removal produces a graph with more connected components than in the original graph.

[See example in the textbook]

Connectedness in directed graphs*

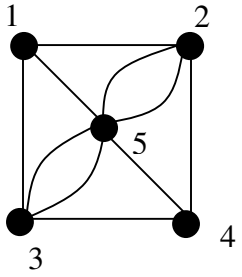
A directed graph is **strongly connected** if there is a path from a to b and from b to a whenever a and b are vertices in the graph. A directed graph is **weakly connected** if there is a path between every two vertices in the underlying undirected graph. The subgraphs of a directed graph G that are strongly connected but not contained in larger strongly connected subgraphs are called **strongly connected components** or **strong components** of G.

Euler Paths and Circuits

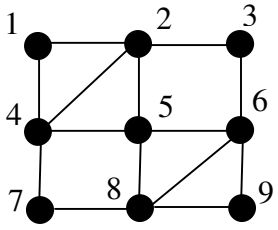
An **Euler circuit** in a graph G is a simple circuit containing every edge of G. An **Euler path** in G is a simple path containing every edge of G.

Example:

Find the Euler path or circuit in the following graphs



No Euler circuit exist, but the Euler path is
1,2,5,1,3,5,4,3,5,2,4



The Euler circuit is 1,2,3,6,9,8,7,4,5,8,6,5,2,4,1

Necessary and Sufficient Conditions for Euler Circuits and Paths

Theorem 5:

A connected multigraph has an Euler circuit if and only if each of its vertices has even degree.

Proof:

Take a connected multigraph G =(V, E) where V and E are finite. We can prove the theorem in two parts.

First we prove that if a connected multigraph has an Euler circuit, then all the vertices have even degree. For this, take a vertex v, where the Euler circuit begins. There is some edge that is incident to v and some other vertex say u then we have an edge {v, u}. This edge {v, u} contributes one to the degree of v and u both. Again there must be some edge other than {v,u} that is incident to u and some other vertex. In this case the total degree of the vertex u becomes even, so whenever in the circuit the vertex is met the degree of that vertex is even since every time entering and leaving the vertex gives even degree to all the vertices other than the initial vertex. However since the circuit must terminate in the vertex v and the edge that is terminating the circuit contributes one to the degree of the initial vertex v the total degree of the vertex v is also even. Now we have every time the vertex is entered and left it gives even degree and the initial vertex also gives even degree, we can conclude that if a graph has Euler circuit, then all the vertices have even degree.

Now we try to prove that if all the vertices in the connected multigraph have even degree, the there exist Euler circuit. For this, take a connected multigraph G with all the vertices having even degree. To make a circuit start at arbitrary vertex, say a of G. now start from the vertex a = x_0 and arbitrarily choose other vertex x_1 to form and edge $\{x_0, x_1\}$. Continue building the simple path $\{x_0, x_1\}$, $\{x_1, x_2\}$, …, $\{x_{n-1}, x_n\}$. This path terminates since it has a finite number of edges. It begins at a with an edge {a, x} and terminate at a with some edge {y, a}. This is correct since every vertex has even degree in the graph we are considering, if an edge left some vertex then there must be an edge entering that vertex to make its degree even. Now we have shown that there exists simple circuit in the graph with all the vertices of even degree. If this circuit has all the edges of the graph in

it, then the simple circuit is itself an Euler circuit. If all the edges are not in the circuit, then we have next possibility. Now, consider the subgraph, say H that is formed by removing all the edges that are already in the simple circuit formed above and by removing the isolated vertices after edges are removed. Since the original graph G is connected, there must be at least one vertex of H that is common with the circuit we have formed. Let w be such a vertex. Every vertex in H has even degree since it is a subgraph of original graph. In case of w, while forming the circuit pairs of incident edges are used up. So the degree of w is again even. Beginning at w we can build a simple circuit as described above. We can continue this process until all edges have been used. Now if we combine the formed circuit in a way that it makes use of common vertex to make a circuit then we can say that the circuit is an Euler circuit. Hence if every vertices of a connected graph has an even degree then it has an Euler circuit.

This concludes the proof.

Theorem 6:

A connected multigraph has an Euler path but not Euler circuit if and only if it has exactly two vertices of odd degree.

Proof:

This fact can be proved if we can prove that first, if the connected multigraph has Euler path exactly two vertices have odd degree and second if the connected multigraph has exactly two vertices of odd degree, then it has Euler path.

Now, if the graph (in this proof graph means connected multigraph) has an Euler path say from a to z but not Euler circuit, then it must pass through every edge exactly once. In this scenario the first edge in the path contributes one to the degree of vertex a, and at all other time when other edges pass through vertex a it contributes twice to the degree of a, hence we can say that degree of a is odd. Similarly the last edge in the path coming to z contributes one to the degree of z, all the other edges contributes two one for entering and one for leaving. Here also the degree of last vertex, z is odd. All the other vertices other than a and z must have even degree since the edges in those vertices enter and leave the vertex contributing two to the degree every time the vertices are met. Hence if there is an

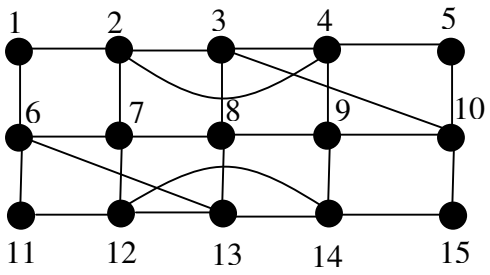
Euler path but not an Euler circuit, exactly two vertices of the graph have odd degree.

Secondly, if exactly two vertices of a graph have odd degree and lets consider they are a and z. Now, consider another graph that adds an edge {a, z} to the original graph, then the newly formed graph will have every vertices of even degree. So there exists Euler circuit in the new graph and the removal of the new edge gives us the Euler path in the original graph. Hence if exactly two vertices of the graph have an odd degree, then the graph has an Euler path but not Euler circuit.

This concludes the proof.

Example:

Find Euler path or circuit?



Solution:

In the above graph, all the vertices have even degree, hence there is an Euler circuit. The Euler circuit is 1,2,3,4,5,10,15,14,13,12,11,6,13,8,9,14,12,7,8,3,10,9,4,2,7,6,1

Hamilton paths

A path $x_0, x_1, \ldots, x_{n-1}, x_n$ in the graph G = (V, E) is called Hamilton path if V ={$x_0, x_1, \ldots, x_{n-1}, x_n$} and $x_i \neq x_j$ for $0 \leq i < j \leq n$. A circuit $x_0, x_1, \ldots, x_{n-1}, x_n, x_0$, with $n > 1$, in a graph G =(V, E) is a Hamilton circuit if $x_0, x_1, \ldots, x_{n-1}, x_n$ is a Hamilton path.

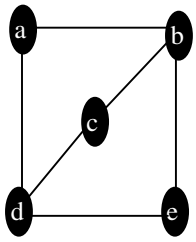
A graph with a vertex of degree one cannot have a Hamilton circuit.

If a vertex in a graph has degree 2, then both edges incident with this vertex must be part of Hamilton cycle.

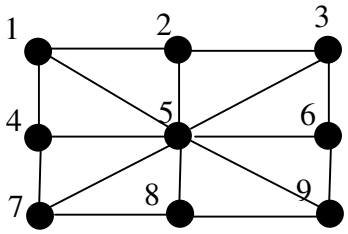
Example:

Find Hamilton circuit from the following graph if exists? What about Hamilton path?

(a)



(b)



Solution:

In graph (a) there is no Hamilton circuit since the node c has degree 2 and both the edges from it must be in Hamilton circuit, which is not possible. One of the Hamilton path in the graph (a) is a, b, c, d, e.

In graph (b) we can find Hamilton circuit, the circuit can be 1,2,3,5,6, 9,8,7,4,1. Since there is circuit we can have path also.

Theorem 7: Dirac's Theorem

If G is a simple graph with n vertices with $n \geq 3$ such that the degree of every vertex in G is at least n/2, then G has a Hamilton circuit.

Theorem 8: Ore's Theorem

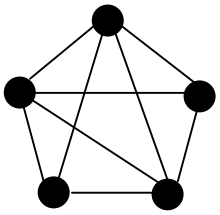
If G is a simple graph with n vertices with $n \geq 3$ such that $\deg(u) + \deg(v) \geq n$ for every pair of nonadjacent vertices u and v in G, then G has a Hamilton path.

Planar graphs

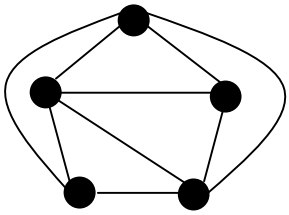
A graph is called a planar if it can be drawn in the plane without any edges crossing. Such drawing is called a planar representation of the graph.

Example:

Draw the graph below as planar representation of the graph.



Solution:



Theorem 9: Euler's Formula

Let G be a connected planar simple graph with e edges and v vertices. Let r be the number of regions in a planar representation of G. Then r = e – v + 2.

Example:

Suppose that a connected planar graph has 30 edges. If a planar representation of this graph divides the plane into 20 regions, how many vertices does this graph have?

Solution:

We have, r = 20, e = 30, so by Euler's formula we have v = e – r + 2 = 30 – 20 + 2 = 12. So the number of vertices is 12.

Corollary 1:

If G is a connected planar simple graph with e edges and v vertices where $v \geq 3$, then $e \leq 3v - 6$.

Corollary 2:

If G is a connected planar simple graph, then G has a vertex of degree not exceeding five.

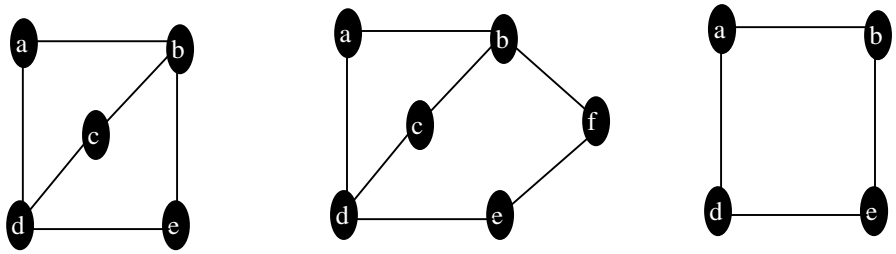
Corollary 3:

If a connected planar simple graph has e edges and v vertices with $v \geq 3$ and no circuits of length three, then $e \leq 2v - 4$.

If a graph is planar, so will be any graph obtained by removing an edge {u, v} and adding a new vertex w together with edges {u, w} and {w, v}. Such an operation is called an **elementary subdivision**. The graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are called **homeomorphic** if they can be obtained from the same graph by a sequence of elementary subdivisions.

Example:

The below example graphs are homeomorphic to the third graph.

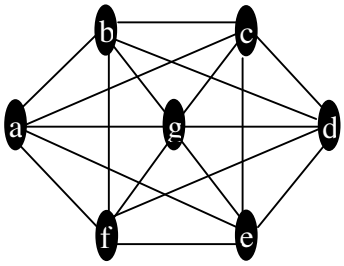


Theorem 10: Kuratowski's Theorem

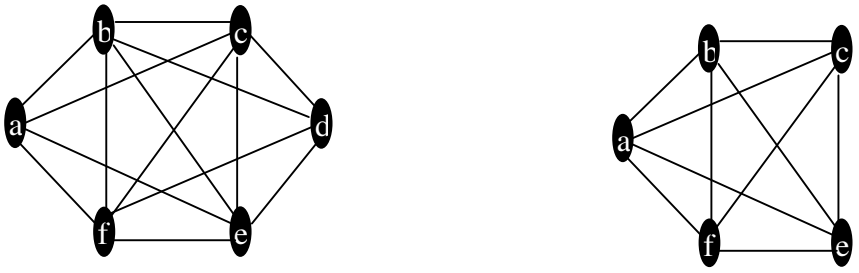
A graph is nonplanar if and only if it contains a subgraph homeomorphic to $K_{3,3}$ or K_5.

Example:

Determine whether the following graph is planar or not?



Solution:



Above we saw that the graph is homeomorphic to K_5, the given graph is not planar.

Graph coloring

A coloring of a simple graph is the assignment of a color to each vertex of the graph so that no two adjacent vertices are assigned the same color.

A chromatic number of a graph is the least number of colors needed for a coloring of this graph.

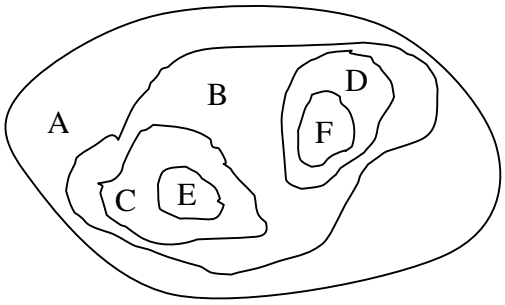
Theorem 11:The Four Color Theorem

The chromatic number of a planar graph is no greater than four.

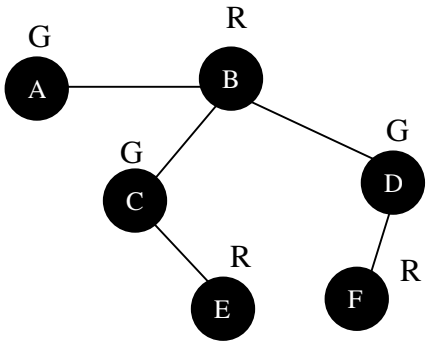
To show the chromatic number of a graph as k we must show that the graph can be colored using k colors and the given graph cannot be colored using fewer than k colors.

Example#1

Construct the dual graph for the map shown. Then find the number of colors needed to color the map so that no two adjacent regions have the same color.



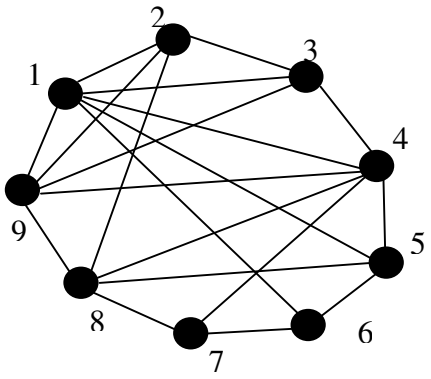
Solution:



We can color the graph with at most 2 colors as shown in the graph. Take R and G as red and green

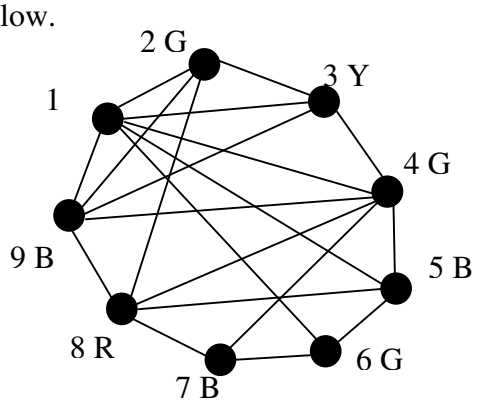
Example#2

Find the chromatic number of the graph below.



Solution:

Lets start with vertex 1, it has adjacent vertices as 2, 3, 4, 5, 6, 9 so using only 2 colors would suffice for the graph having the edges from 1 to its adjacent vertices. However since 9 has its adjacent vertices as 1, 2, 3, 4 we cannot just color the above graph with 2 colors because at least 1, 2 and 9 must have different colors. Trying with 3 colors we found that at least 1, 2, 3, and 9 must have different colors. So trying with four colors we can color the graph. Hence the chromatic number of the above graph is 4. possible coloring is shown in the figure below.



Self Studies

Read chapter 8 of your textbook such that you can cover all the read materials in the class.

* ***Student: this portion won't be covered in detail in the class, read yourself.***

[Trees]

Discrete Structures (CSc 511)

Samujjwal Bhandari

Central Department of Computer Science and Information Technology (CDCSIT)

Tribhuvan University, Kirtipur,

Kathmandu, Nepal.

Trees

A connected simple graph without a cycle is called tree. It has got wide varieties of applications in the field of computer science like in searching and sorting. The simple graph that no simple circuit but not connected is called **forest**. The forest has each of its connected components as tree.

Theorem 1:

An undirected graph is a tree if and only if there is a unique simple path between any two of its vertices.

Proof:

Assume that T is a tree. Since T is a tree it is a connected simple graph with no simple circuits. Let x and y be two vertices of T. we know that every connected graph has a simple path between every pair of vertices. So there is a simple path from x to y. This path must be unique because, if the path between x and y is not unique then there is another path between x and y that uses edges different from the path between x and y for first path, then reversing the path i.e. going from x to y from the first path and going from y to x through the second path forms a circuit. This is a contradiction that T is a tree; hence there is a unique simple path between any two vertices of a tree.

Again assume that there is a unique simple path between any two vertices of a graph, say T. Since there is a path between any two vertices of a graph, the graph is connected. Now, we can show that the graph T cannot have simple circuit. Had there been a simple circuit, there would be two simple paths between two vertices, say x and y, and the two simple path between x and y would create a simple circuit where first path goes from x to y and the second path goes from y to x. This violates our assumption that the path is unique. Hence, a graph with a unique simple path between any two vertices is a tree.

A **rooted tree** is a tree in which one vertex has been designated as the root and every edge is directed away from the root. The tree in which root is defined produces a directed graph (since path between two vertices (theorem 1) in a tree is unique).

Take a rooted tree T. if v is the vertex in T other than root, then the **parent** of v is a vertex u in T such that there is a directed edge from u to v. In this scenario v is called

child of u. Vertices with same parents are called **siblings**. All the vertices that appear in the path from root to some vertex v in T, including root are called **ancestors** of v. The **descendents** of a vertex v are those vertices that have v as their ancestor. All the vertices that have children are called **internal vertices** (root is also an internal vertex if the tree has more than one vertices).

A **subtree** of a rooted tree T, with root a, is a subgraph of the tree consisting of a and all of its descendents and all the edges incident to these descendents. Here all the vertices must be in T also.

A **m-ary tree** is a rooted tree in which every internal vertex has no more than m children. It is called **full m-ary tree** if every internal vertex has exactly m children.

Example: binary tree i.e. 2-ary tree.

An **ordered rooted** tree is a rooted tree where the children of each internal vertex are ordered. For e.g. in ordered binary tree (also called just a binary tree) if an internal vertex has two children then the first child is called **left child** and the second child is called **right child**.

[Work out some examples and read some other terminologies like height of a tree, level of a tree, complete m-ary tree, balanced tree, left subtree, and right subtree, etc. not covered here yourself to understand them.]

From now onwards unless stated explicitly tree is referred to as rooted tree.

Properties of Trees

Theorem 2:

A tree with n vertices has n-1 edges.

Proof:

Basis Step: For n =1, since there is only one vertex, there are no edges i.e. number of edge = 1-1 =0, true.

Inductive Hypothesis: Assume that the tree with k vertices had k-1 edges is true.

Inductive Step: Now for the tree with k+1 vertices, take a vertex v such that v is a leaf and take another vertex w such that w is a parent of v i.e. there is an edge {w, v}. If we

remove the edge {w, v} and the leaf v, then the new tree formed (the graph will be connected again) will have k vertices and by the induction hypothesis it will have k-1 edges. Now we had deleted 1 vertex and 1 edge form the tree with k+1 vertices to form a tree with k vertices and k-1 edges, hence it is seen that the tree with k+1 vertices has k edges.

This completes the proof.

Theorem 3:

A full m-ary tree with i internal vertices contain n = mi + 1 vertices.

Proof:

In a full m-ary tree all the vertices but the root are child of the internal vertices. Since each internal vertex has m children there are mi vertices in the tree other than the root. Hence the tree has n =mi + 1 vertices.

Theorem 4:

A full m-ary tree with

i) n vertices has i = (n-1)/m internal vertices and l =[(m-1)n + 1]/m leaves,

ii) i internal vertices has n = mi + 1 vertices and l = (m-1)i + 1 leaves,

iii) l leaves has n = (ml –1)/(m –1) vertices and i = (l-1)/(m-1) internal vertices.

Proof:

Let n be number of vertices, i be number of internal vertices, and l be number of leaves. From the theorem 3 we know that n = mi +1. Again we know the fact that n = l + i.

i) From n = mi + 1 we have i = (n-1)/m and putting this value of i in n = l + i and solving for l we get l =[(m-1)n + 1]/m

ii) We know n = mi + 1, now putting value of n in n = l + i we get l = (m-1)i + 1.

iii) Form the result of ii i.e. l = (m-1)i + 1 we get i = (l –1)/(m-1) so putting this value in n = mi + 1, we get n = (ml –1)/(m –1) and similarly putting this obtained value of n in n = l + i, we get (l-1)/(m-1).

Theorem 5:

There are at most m^h leaves in an m-ary tree of height h.

Proof:

Basis Step: when height is 1 i.e. h = 1, then the root has at most m children and for the m-ary tree of height 1 all the children of root are leaves. So, the statement is true for h =1.

Inductive Hypothesis: Assume that the statement is true for all m-ary trees with height less than h.

Inductive step: If we can prove that the statement is true for h then we are done. Take a tree of height h. Now cut off the edges that are going from the root to its children. Then there will be subtrees rooted at children of the root of the tree of height h. However, the newly formed rooted tree will have height h-1, so from inductive hypothesis we can say that each subtree has at most m^{h-1} leaves. Here there is at most m such subtrees, since root has at most m children. So the total number of leaves at the tree of height h is at most m. $m^{h-1} = m^h$.

This proves the theorem.

Corollary 1:

If an m-ary tree of height h has l leaves, then $h \geq \lceil \log_m l \rceil$. If the m-ary tree is full and balanced, then $h = \lceil \log_m l \rceil$.

Applications

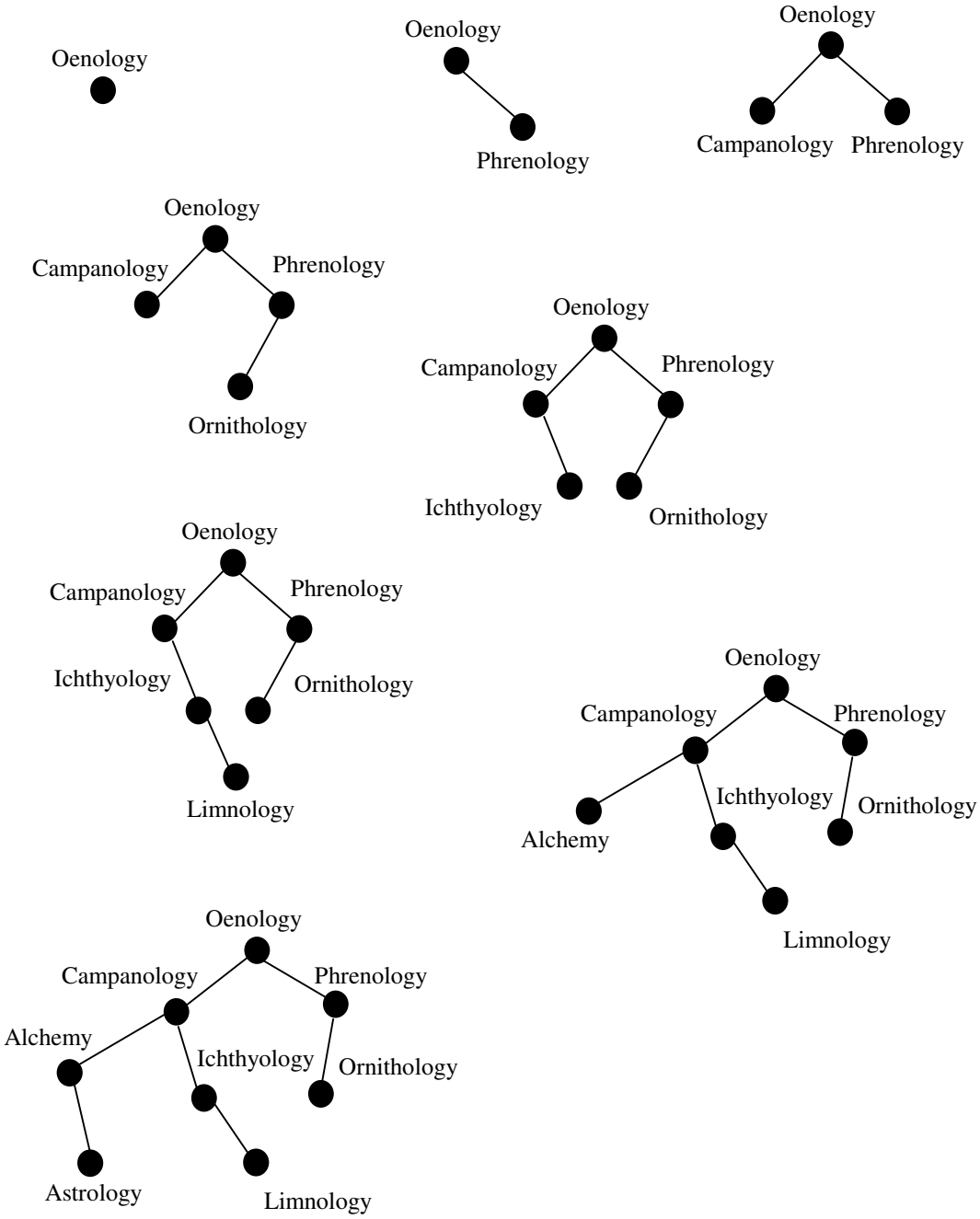
Binary Search Trees

Binary search tree is a binary tree in which each child vertex is considered as a right or left child, no vertex has more than one left or right child, and each vertex is labeled with a key, which is one of the items. The keys are places such that a key of a vertex is larger than all the key of the vertices of its left subtree and smaller than the keys of the vertices of its right subtree.

Example:

Build a binary search tree for the words oenology, phrenology, campanology, ornithology, ichthyology, limnology, alchemy, and astrology using alphabetic order.

Solution:



Decision Trees

Binary trees can be used to locate items based on comparisons, in this situation each comparison tells us to visit either left subtree or right subtree. A rooted tree where internal vertices correspond to a decision and subtree at these vertices gives possible outcomes of the decisions made is called decision tree.

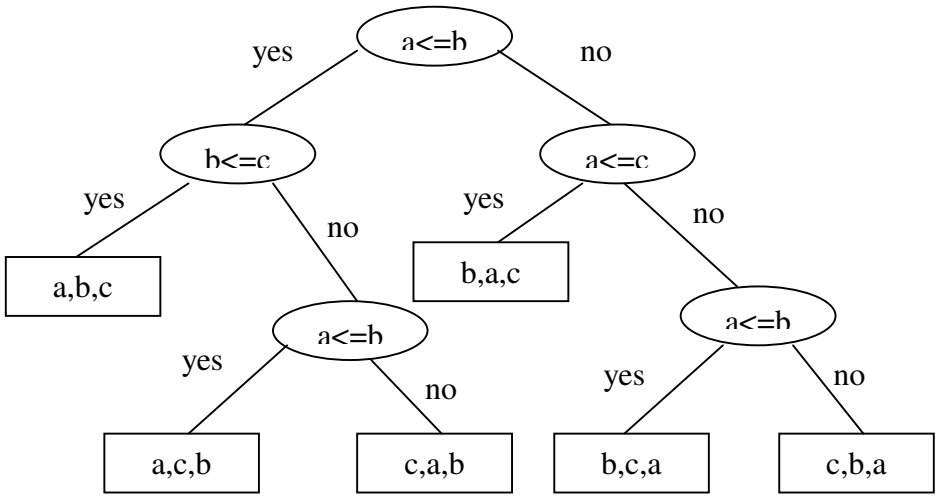
Example:

Complexity of sorting algorithms

Solution:

You know many of the sorting algorithms. In most of those algorithms the basic idea for sorting is comparison between two items at a time. In this situation those comparisons can be pictorially represented as full binary tree. Since we know that for n elements there may be n! solution that could be obtained from the comparison, we can say that there will be n! leaves in the formed decision tree. By the result we know the height of the tree is $\lceil \log_m n! \rceil$. We again can say that each internal vertex represents to the comparison. So, total number of comparison for getting output is same as reaching the leaf i.e. $\lceil \log_m n! \rceil$.

We can show that this comparison is Ω(nlogn). Hence a lower bound for any comparison sorting is Ω(nlogn). The decision tree below depicts the above idea for three elements a, b, and c.



Spanning Trees

Given a simple graph G. A **spanning tree** of G is a subgraph of G that is a tree containing every vertex of G. Here since we can forma tree, a simple graph G must be connected. This is true conversely too. The theorem below gives explanation for this prediction.

Theorem 6:

A simple graph is connected if and only if it has a spanning tree.

Proof:

Lets assume that a simple graph G has a spanning tree T. We know that T contains every vertex of G. Since T is a tree there is a path between any two of its vertices. Again, since T is a subgraph of G there is a path between any two of the vertices of G. Hence G is connected.

Now, assume that G is connected. Here if G is not a tree then there must be some simple circuit. Remove the edge that connects two vertices of G and that makes the simple circuit. This removal of an edge breaks the circuit from G and creates subgraph of G, but still contains all the vertices of G and it is connected. If the newly created subgraph is not a tree, we can process similarly as above to break the circuit. Since there are only finite numbers of edges the removal of edges terminates at last making the subgraph as a tree. This tree will contain all the vertices of a graph G thus becoming a spanning tree of G.

This is a proof.

Depth First Search

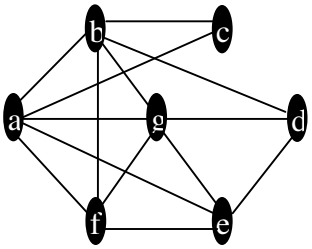
This is a technique that can be used to find the spanning tree of a connected graph. In this method we will arbitrarily choose a vertex that is regarded as a root. When root is chosen the path is formed by starting at a root vertex by successively adding vertices and edges, where the added edges is incident with the last vertex in the path and is not repeated. This process is continued until no possible path can be formed. If the path contains all the vertices then the tree consisting this path is spanning tree. If the path does not go through all the vertices then we must add other edges and vertices. For this move back from the last vertex that is met in the previous path and find whether it is possible to find new path

starting from the vertex just met. If there is such a path continue the process above. If this cannot be done, move back to another vertex and repeat the process. The whole process is continued until all the vertices are met. Since we have finite graph the process terminates giving us a spanning tree. The underlying undirected graph of the found rooted tree is a spanning tree of the given graph.

This method of search is also called **backtracking**.

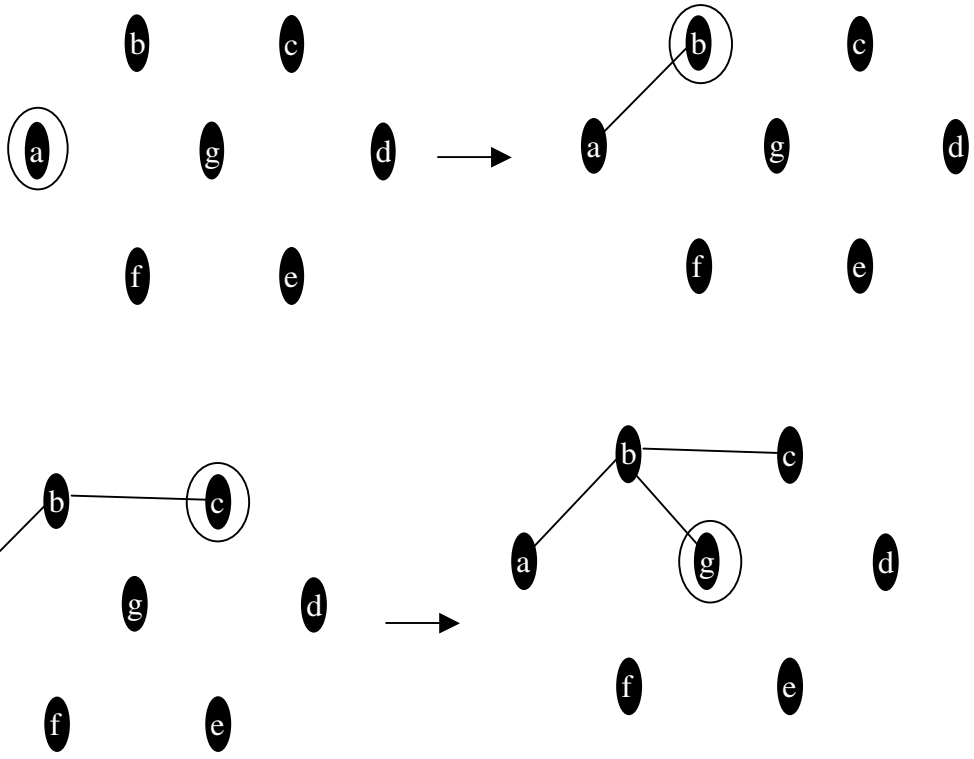
Example:

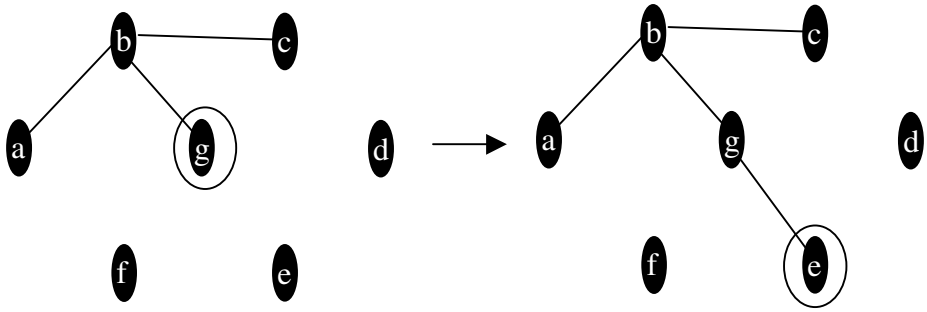
Use depth first search to find a spanning tree of the following graph.

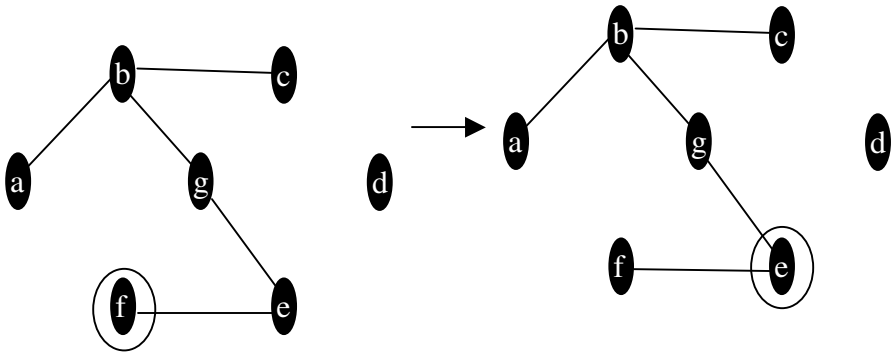


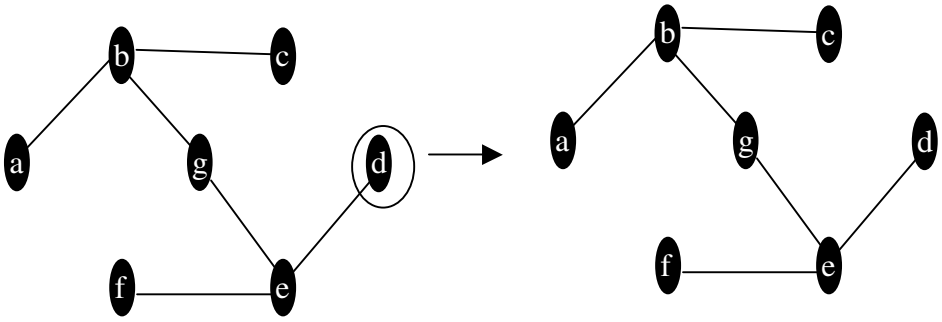
Solution:

Choose a as initial vertex then we have









Algorithm: Depth First Search

function DFS(G: a connected graph with vertices v_1, v_2, …, v_n)

{

T = tree consisting only of the vertex v_1 (arbitrarily chosen root)

visit(v_1)

}

function visit(v: vertex of G)

{

for each vertex w adjacent to v and not yet in T

begin

add vertex w and edge {v,w} to T

visit (w)

end

}

Analysis:

If we represent the graph as adjacency list, no computation is needed for finding the adjacent vertices. The complexity of the algorithm is greatly affected by **visit** function we can write its running time in terms of the relation T(n) = T(n-1) + O(n), here O(n) is for each vertex at most all the vertices are checked (for loop). At each recursive call a vertex is decreased. Solving this we can find that the complexity of an algorithm is $O(n^2)$.

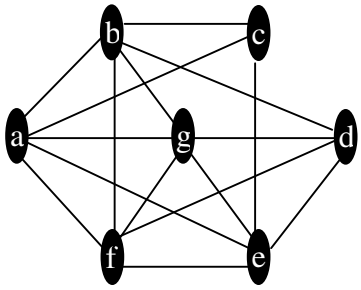
Breadth First Search

In this process also rooted tree is formed and its underlying undirected graph is a spanning tree. Here choose some vertex arbitrarily as a root. Add all the vertices and edges that are incident in the root. The new vertices added will become the vertices at the level 1 of the spanning tree. Arbitrarily order these vertices. Visit in order all the added vertices (level 1) and follow the process above to add new edges and vertices Remember you cannot add same vertex in twice. This process gives us level 2 vertices. Follow the same procedure until all the vertices in a tree have been added. Since the graph is finite

this process terminates and contains all the vertices making the tree as a spanning tree.

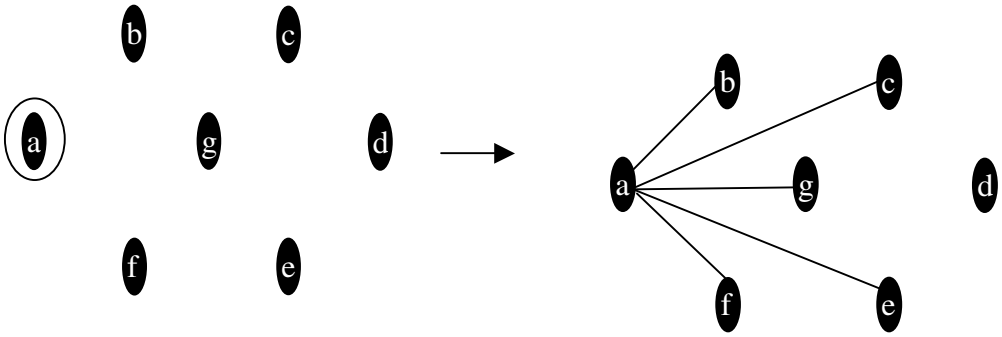
Example:

Use breadth first search to find a spanning tree of the following graph.

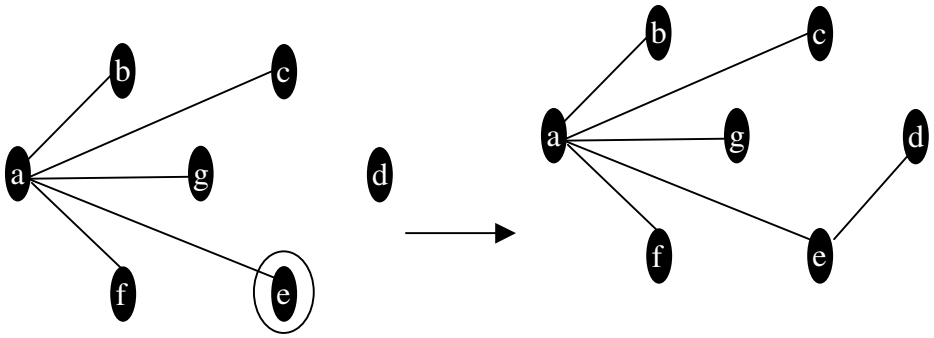


Solution:

Choose a as initial vertex then we have



Order the vertices of level 1 i.e. {b, c, g, e, f}. Say order be {e, f, g, b, c}.



Algorithm: Breadth First Search

function BFS(G: a connected graph with vertices v_1, v_2, …, v_n)

{

T = tree consisting only of the vertex v_1 (arbitrarily chosen root)

L = an empty list

put v_1 in the list L of unprocessed vertices

while L is not empty

begin

remove the first vertex v, form L

for each neighbor w to v

if w is not in L and not in T then

begin

add w to the end of the list L

add w and edge {v,w} to T

end

end

}

Analysis:

From the algorithm above all the vertices are put once in the list and they are accessed. For each accessed vertex from the list their adjacent vertices are looked for and this can be done in O(n) time. This computation for all the possible vertices that may be in the list i.e. n produce complexity of an algorithm as $O(n^2)$.

Minimum Spanning Trees

A minimum spanning tree in a connected weighted graph is a spanning tree that has the smallest possible sum of weights of its edges.

In this part we study one algorithm that is used to construct the minimum spanning tree from the given connected weighted graph.

Prim's Algorithm

Working principle: This is a greedy algorithm that chooses optimal solution at a particular instance. Choose an edge of the smallest edge, put it into the spanning tree. Successively add to the tree edges of minimum weight that are incident to the vertex already in the tree and not forming a simple circuit. Stop when n-1 edges are added.

Algorithm:

Tree prim(G: connected weighted undirected graph with n vertices)

{

T = a minimum weight edge.

for i = 1 to n-2

{

e = an edge of minimum weight incident to a vertex in T and not forming a simple circuit in T if added to T

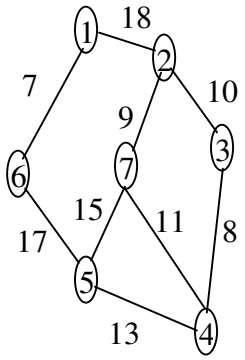
T = T with e added

}

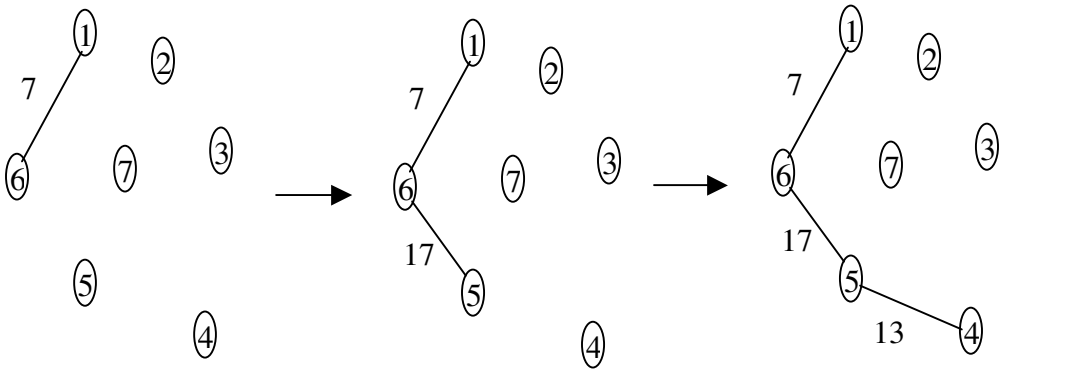
}

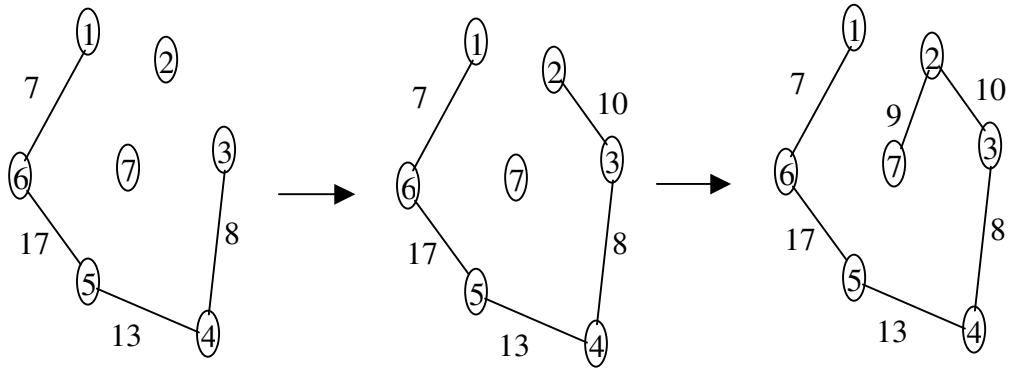
Example:

Find the minimum spanning tree of the following graph using Prim's algorithm.



Solution:





Self Studies

Read chapter 9 of your textbook such that you can cover all the read materials in the class.

Note: Tree traversals are not covered here; it is to the students as self-study.

[Boolean Algebra]

Discrete Structures (CSc 511)

Samujjwal Bhandari

Central Department of Computer Science and Information Technology (CDCSIT)

Tribhuvan University, Kirtipur,

Kathmandu, Nepal.

Boolean functions

Boolean algebra gives us rules and operations that can be used for working on the set {0,1}. It has got applications in many fields like computing, electronics, etc. The most basic operations that are used in Boolean algebra are:

Complementation: Complementation of an element is denoted with bar, or prime (we use prime here) and is defined by the rules 0' =1 and 1' = 0.

Boolean Sum: The Boolean sum is denoted by + or OR and is defined by the rules 1 + 1 = 1, 1 + 0 = 1, 0 + 1 = 1, and 0 + 0 = 0.

Boolean Product: The Boolean product is denoted by dot (.) or AND, with the rules defined as 1.1 = 1, 1.0 = 0, 0.1 = 0, and 0.0 = 0.

Remember: sometimes we may disregard dot symbol.

Given A = {0,1}, then $A^n = \{(x_1, x_2, \ldots, x_n) \mid x_i \in A \text{ for } 1 \leq i \leq n\}$ is the set of all possible n-tuples of 0s and 1s. The variable x is called **Boolean variable** if it has only 0 or 1 as its value i.e. values from A only. A function from A^n to A is called **Boolean function of degree n**.

Example:

Find the values of Boolean function represented by F(x, y, z) = x + yz.

Solution:

x	y	z	yz	x + yz
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	1	1
1	0	0	0	1
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

The Boolean expressions in the variables $x_1, x_2, \ldots, x_n$ can be defined as

$0, 1, x_1, x_2, \ldots, x_n$ are Boolean expressions;

if E and F are Boolean expressions, then E', (EF), and (E + F) are Boolean expressions.

Equality of Boolean functions: Boolean functions F and G of n variable are equal if and only if $F(b_1, b_2, \ldots, b_n) = G(b_1, b_2, \ldots, b_n)$ whenever $b_1, b_2, \ldots, b_n$ belongs to A where A is the set on which Boolean function is defined. Two Boolean expressions representing same Boolean function are called **equivalent**. For e.g. ab, ab + 0 and ab.1 are equivalent.

The complement of the Boolean function F is the function F', where $F'(x_1, x_2, \ldots, x_n) = [F(x_1, x_2, \ldots, x_n)]'$.

Given two Boolean functions F and G on n variables,

The Boolean sum F + G is given by

$(F + G)(x_1, x_2, \ldots, x_n) = F(x_1, x_2, \ldots, x_n) + G(x_1, x_2, \ldots, x_n)$ and

The Boolean product F.G is defined by

$(FG)(x_1, x_2, \ldots, x_n) = F(x_1, x_2, \ldots, x_n)\, G(x_1, x_2, \ldots, x_n)$

Number of Boolean functions of degree n

A Boolean function of degree n is a function from the set of 2^n elements that is formed as a pair from the set A ={0,1} to the set with two elements. Hence by the product rule there are ${}_2 2^n$ different Boolean function of degree n.

Example:

Boolean functions of degree 2 are given in table below.

x	y	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}	F_{15}	F_{16}
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

Identities of Boolean Algebra

x . 1 = x	Identity law
x + 0 = x	Identity law
x . 0 = 0	Domination law
x + 1 = 1	Domination law
x . x = x	Idempotent law
x + x = x	Idempotent law
(x')' = x	Complementation law
x . y = y . x	Commutative law
x + y = y + x	Commutative law
(x . y) . z = x . (y . z)	Associative law
(x + y) + z = x + (y + z)	Associative law
x . (y + z) = (x . y) + (x . z)	Distributive law
x + (y . z) = (x + y) . (x + z)	Distributive law
(x . y)' = x' + y'	De Morgan's law
(x + y)' = x' . y'	De Morgan's law
x + (x . y) = x	Absorption law
x.(x + y) = x	Absorption law
x + x' = 1	Unit property
x . x' = 0	Zero property

Duality

The dual of Boolean expression is obtained by interchanging Boolean sums and Boolean products and interchanging 0s and 1s. See identities above to note that all the properties have duals.

The dual of a Boolean function F represented by Boolean expressions denoted by F^d is the function represented by duals of these Boolean expressions.

All the identities are valid for the duals. This is called duality principle.

Representation Of Boolean functions

The problem of getting the Boolean expression when the Boolean function is given will be covered here. Every Boolean function can be represented by using three operators +, ., and '. We also study whether we can use smaller set of operators to represent the Boolean functions.

A **literal** is a Boolean variable or its complement. A **minterm** of the Boolean variables $x_1, x_2, \ldots, x_n$ is a Boolean product $y_1y_2, \ldots y_n$, where $y_i = x_i$ or $y_i = x_i'$. So minterm is a product of n literals, with one literal for each variable. The minterm has value 1 if and only if all the literals have value 1. This is true only if $x_i = 1$ when $y_i = x_i$ and $x_i = 0$ when $y_i = x_i'$. A **maxterm** of the Boolean variables $x_1, x_2, \ldots, x_n$ is a Boolean sum $y_1 + y_2 + \ldots + y_n$, where $y_i = x_i$ or $y_i = x_i'$. So maxterm is a sum of n literals, with one literal for each variable. The maxterm has value 0 if and only if all the literals have value 0. This is true only if $x_i = 0$ when $y_i = x_i$ and $x_i = 1$ when $y_i = x_i'$.

Sum of Products and Product of Sums Expansions

By taking Boolean sum of distinct minterms we can build up a Boolean expression with a specified set of values. This representation of Boolean function is called **sum of products expansion** or **disjunctive normal form (DNF)**. Similarly by taking Boolean products of distinct maxterms we can build up a Boolean expression with a specified set of values. This representation of Boolean function is called **product of sums expansion** or **conjunctive normal form (CNF)**.

Example:

Find the sum of products expansion and product of the sums expansion of the Boolean function F(x, y, z) = (x + z)y.

Solution:

F(x, y, z)	= (x + z)y	
	= xy + zy	Distributive law
	= xy + yz	Commutative law
	= xy1 + 1yz	Identity law
	= xy(z + z') + (x + x')yz	Unit property
	= xyz + xyz' + xyz + x'yz	Distributive law.

= xyz + xyz + xyz' + x'yz Associative law

= xyz + xyz' + x'yz Idempotent law

xyz + xyz' + x'yz is the sum of products

We use next approach for finding product of sums however for identity property use 0 while taking sum if you are using above approach.

x	y	z	x + z	(x + z)y
0	0	0	0	0
0	0	1	1	0
0	1	0	0	0
0	1	1	1	1
1	0	0	1	0
1	0	1	1	0
1	1	0	1	1
1	1	1	1	1

The five rows in the table above gives 0 value so the product of sums is

(x + y + z) (x + y + z') (x + y' + z) (x' + y + z) (x' + y + z') Here put complement where there is 1.

Functional Completeness

We have seen that every Boolean function can be represented by CNF or DNF and they just use the operators ', ., and + so we say that the set of operators {', ., +} is **functionally complete**. We can use less operator than mentioned above if we eliminate every Boolean sum by using the fact x + y = ((xy)')' to get functionally complete set {', .} or by eliminating every Boolean product by using the fact xy = ((x + y)')' to get functionally complete set {', +}.

We have the definition of the operator ↓ (NOR) as 1↓1 = 1↓0 = 0↓1 = 0 and 0↓0 = 1. Here we can show that {↓} is functionally complete. This can be done if we can show that it can express the function having both ' and . Since {', .} is functionally complete.

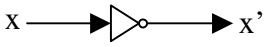
x' = x↓x, x + y = (x↓y) ↓ (x↓y). Hence {↓} is functionally complete. If we define NAND {|} with 1 | 1 = 0 and 0 | 0 = 0 | 1 = 1 | 0 = 1 show {|} is functionally complete.

Logic gates

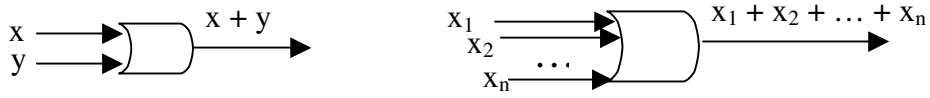
The basic elements in the electronic circuit are **gates**. They implement the Boolean operations. Here we study different types of gates and their use in circuit design. While designing the circuits we generally consider circuits that has no memory and the output depends upon the input only. This type of circuit is called **combinational circuit** or **gating networks**.

Gates

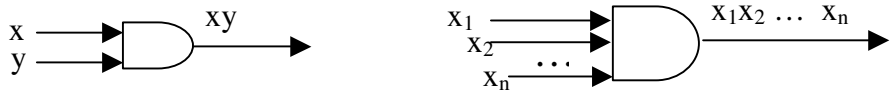
Inverter: This type of gate accepts the value of Boolean variable as input and returns the complement of the value. The pictorial symbol is like below:



OR gate: This gate as shown in symbol below, carries out the function as provided by Boolean sum. We will permit more than two inputs to OR gate.

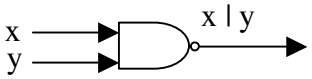


AND gate: This gate as shown in symbol below, carries out the function as provided by Boolean product. We will permit more than two inputs to AND gate.

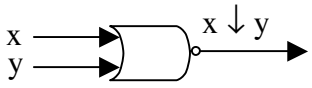


NAND gate:

This gate as shown in symbol below, carries out the function as provided by Boolean operation given by 1 | 1 = 0 and 0 | 0 = 0 | 1 = 1 | 0 = 1.



NOR gate: This gate as shown in symbol below, carries out the function as provided by Boolean operation given by 1 ↓ 1 = 0 ↓ 1 = 1 ↓ 0 = 0 and 0 ↓ 0 = 1.



Example Circuits

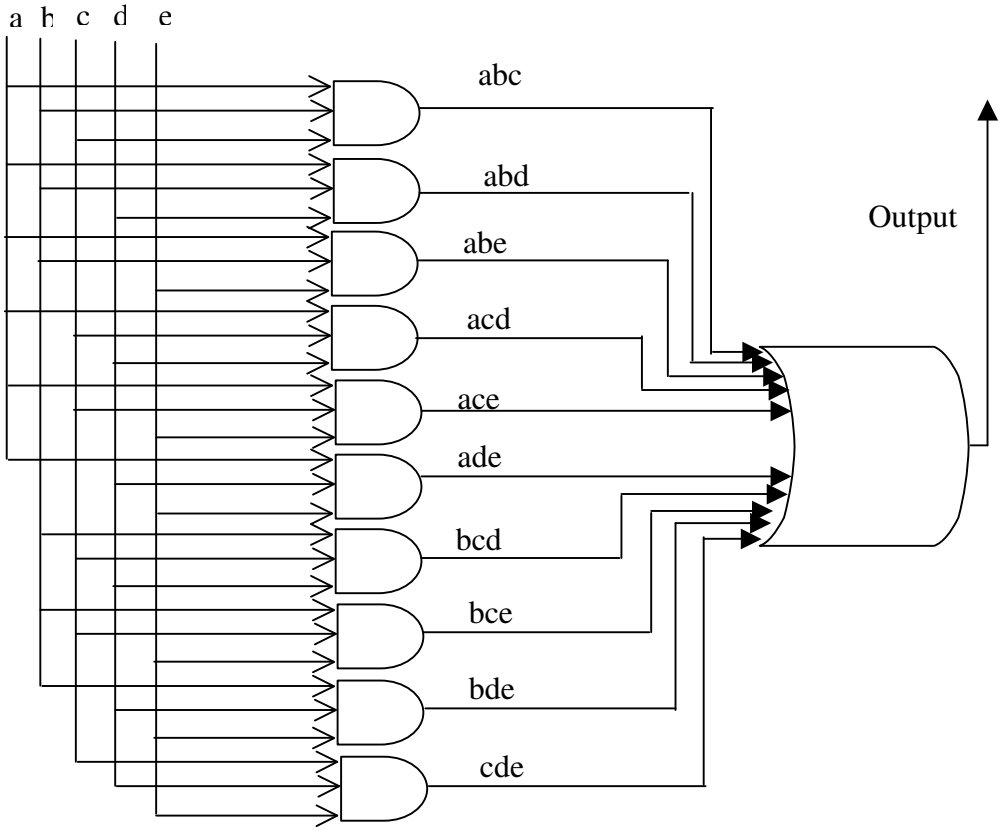
Example #1:

Design a circuit that implements majority voting for five individuals.

Solution:

Here, let each of the five individuals is represented by Boolean variables say, a, b, c, d, e respectively. The result of the vote will be winning if at least 3 of the individual have their ok vote. Let ok vote is assignment of 1 to the Boolean variable and not ok vote be assignment of 0 to the Boolean variable. Then we have the winning combination as given by the Boolean function below.

abc + abd + abe + acd + ace + ade+ bcd + bce + bde + cde. This can be drawn as.



Example #2:

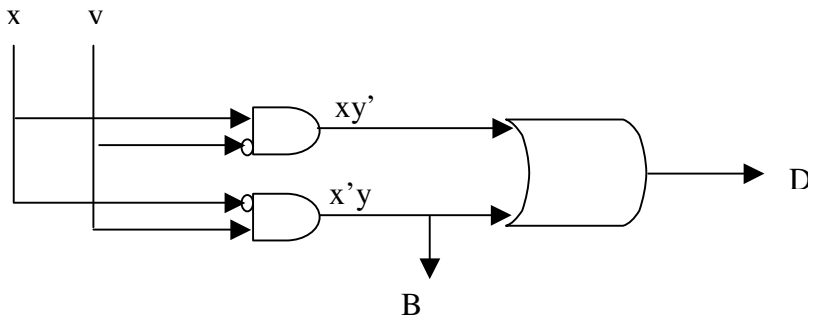
Construct a circuit for half subtractor using AND, OR gates and inverters. A half subtractor has two bits as input and returns as output their difference bit and a borrow bit.

Solution:

Half adder has two bits as input, let x and y be the inputs and x be minuend and y be subtrahend, then we can design circuit for x – y. There will be two cases whether x ≥ y or x < y. In the first case we have F(0, 0) = 0, F(1, 0) = 1 and F(1, 1) = 0 these all are difference bits. In the second case F(0, 1) = 1 where 1 is borrowed. The below truth table gives input output relationship of a half subtractor.

x	y	B	D
0	0	0	0
0	1	1	1
1	0	0	1
1	1	0	0

From the truth table we have functions B = x'y and D = x'y + xy' in the form of sum of products. So the circuit is as below.



Minimization of Circuits

When we try to design the circuit its efficiency greatly depends upon the number of gates in use. So the minimization of Boolean function is the technique that helps in minimizing the number of gates that should be used by creating new equivalent Boolean function from the old one with larger number of literals.

Karnaugh Maps

The graphical method for reducing the number of terms in Boolean expression representing a circuit by finding the terms that can be combined together is called Karnaugh map or K-map. This technique is useful for Boolean function having very

small number of variables. This method is not suitable for mechanizing the process. Here we study minimization of Boolean functions using K-map where functions are in the sum of products form.

Minimization of Boolean function with two variables

There are four possible minterms in the sum of products expansions of a Boolean function having two variables. K-map for this type of function is as below where adjacent cells differ in only one bit position. Simplification can be done if we get two 1's in the adjacent cells.

The product of literals corresponding to a block of all 1s in the K-map is called an **implicant** of the function being minimized. It is called **prime implicant** if this block of 1s is not contained in a larger block of 1s representing the product of fewer literals than this product. The goal of using Karnaugh map is to identify the largest possible blocks in the map and cover all the 1s in the map with least number of blocks, using the largest block first. We choose a block if it is the only block of 1s covering a 1 in the K-map and such block represents an **essential prime implicant**. The sum of products that will be obtained will be the sum of prime implicants.

	y	y'
x	xy	xy'
x'	x'y	x'y'

Example:

Find the K-map for x'y + x'y' and simplify.

Solution:

	y	y'
x	0	0
x'	1	1

The simplified function is x'.

Minimization of Boolean function with three variables

The rectangle of a K-map with three variables has eight cells as shown in the figure. Two cells have common border if and only if they have only one bit difference i.e. adjacent. The simplification is similar as of K-map with variables but we can also take 2 by 2 or 4 by 1 block of cells here.

	yz	yz'	y'z'	y'z
x	xyz	xyz'	xy'z'	xy'z
x'	x'yz	x'yz'	x'y'z'	x'y'z

Example:

Use K-map to minimize the sum of products expansion xyz + xyz' +x'yz' + x'y'z

Solution:

The K-map is

	yz	yz'	y'z'	y'z
x	1	1	0	0
x'	0	1	0	1

The simplification is xy + yz' + x'y'z.

The K-maps for four variables have 16 cells where two of the cells are adjacent if and only if they differ by single literal. The possible combinations for simplification are 1 by 2 or 1 by 4 or 1 by 8 or 2 by 2 or 2 by 4 or 2 by 8 or 4 by 4 or 4 by 8 and their reverses.

Students: try this yourself and in case of any difficulties contact me!

The K-maps for other variables can be generalized as above where function with n variables will have 2^n cells.

Don't Care Conditions

Sometimes while designing the circuit, some of the input combinations may not occur or are not possible. For these types of combinations we can take the value of that combination as 1 or 0 arbitrarily in the minimization of Boolean function. Such values for these functions for the combinations are called don't care conditions. We denote this with d or x symbol in the map.

Example:

Find a minimal sum of products expansion.

	yz	yz'	y'z'	y'z
wx	d	1	d	1
wx'	0	d	d	0
w'x'	0	d	1	0
w'x	0	1	d	0

Solution:

	yz	yz'	y'z'	y'z
wx	d	1	d	1
wx'	0	d	d	0
w'x'	0	d	1	0
w'x	0	1	d	0

The simplified function is z' + wx.

More Examples:

Example #1:

Build a circuit using OR gates, AND gates, and inverters that produces an output of 1 if a decimal digit, encoded using binary coded decimal expansion, is divisible by 3 and an output 0 otherwise.

Solution:

The truth table below gives the input output relation.

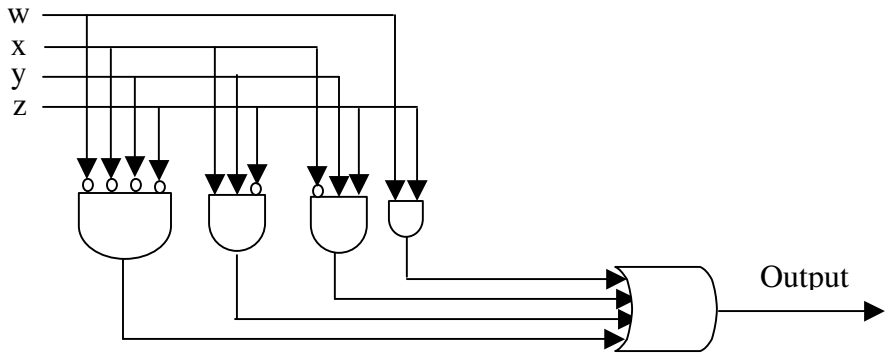
Digit	w	x	y	z	F
0	0	0	0	0	1
1	0	0	0	1	0
2	0	0	1	0	0
3	0	0	1	1	1
4	0	1	0	0	0
5	0	1	0	1	0
6	0	1	1	0	1
7	0	1	1	1	0
8	1	0	0	0	0
9	1	0	0	1	1

From the above truth table we can draw K-map as

	yz	yz’	y’z’	y’z
wx	d	d	d	d
wx’	d	d	0	1
w’x’	1	0	1	0
w’x	0	1	0	0

The simplification gives x’yz + xyz’ + w’x’y’z’ + wz

The logic circuit for this is



Example #2:

Design a circuit that implements majority voting for three individuals. Use Karnaugh map for simplification.

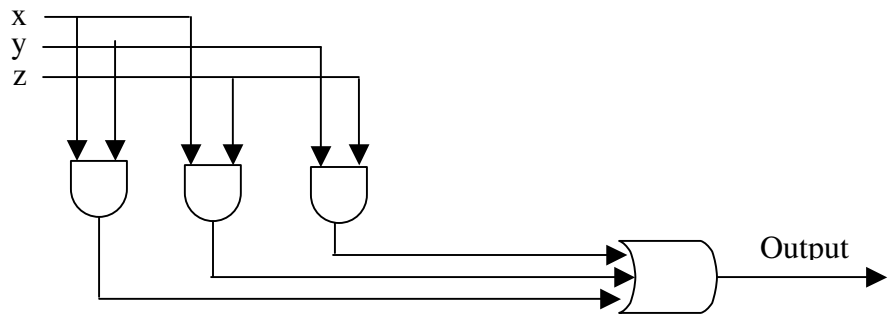
Solution:

If any two of the individuals have positive vote then the outcome is 1. If we assume x, y, z for three individuals voting, then we have the truth table as and corresponding K-map is as shown below:

x	y	z	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

	yz	yz’	y’z’	y’z
x	1	1	0	1
x’	1	0	0	0

The simplification is yz + xz + xy.
The circuit is as given below.



Self Studies

Read chapter 10 of your textbook such that you can cover all the read materials in the class.