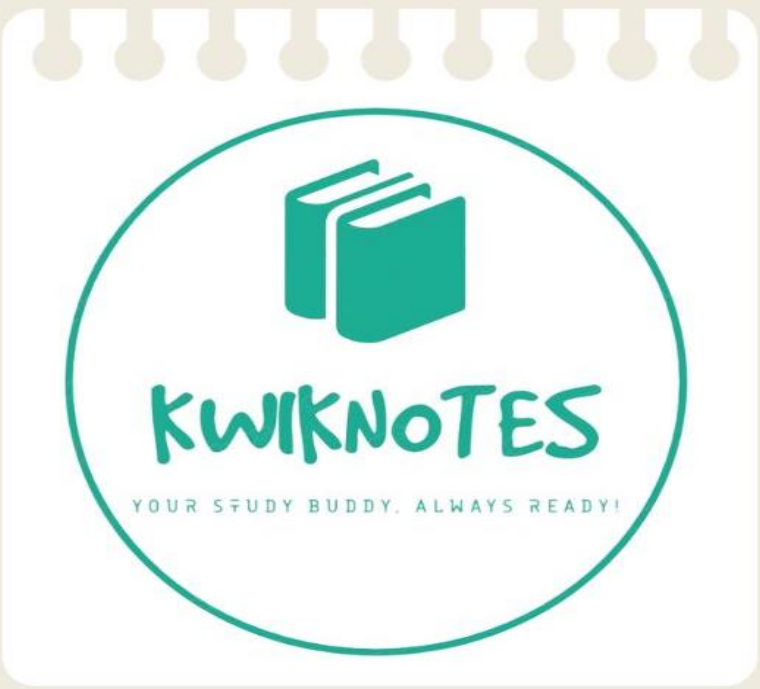


HANDWRITTEN NOTES

Download FREE Notes for Computer Science and related resources only at

Kwiknotes.in

Don't forget to check out our social media handles, do share with your friends.

Design and Analysis of algo - is a branch of CS

It helps to design the algo for solving different types of problems in computer Science
It helps to design & analyze the logic on how algo will work before developing actual code.
It is a unique mathematical approach
It can be said as a ~~design~~ decision making process. Anlaysis algo - best, averge, worst c

Algorithm

→ An algorithm is a finite set of instructions to solve a given problem before the implementation of actual code
→ step by step procedure
→ It accept 0 or more input but produce 1 output atleast.
→ Independnt of programming language
→ Designed to utilize minimum space & time

Properties
→ Should terminate in finite steps
→ Each instructions to be clear & unambiguous called definiteness
→ Atleast 1 output
→ Effectiveness → every instruction be be basic & feasible

PSeudoCode

→ It is highlevel description of algorithm
→ More structured, less detailed than a program. can be understood by layman

	Program	Algo
1)	Written in implementation phase	In design phase
2)	Syntax dependent	independent
3)	Need programming Knowledge	Need domain Knowledge
4)	Dependent on H/w & S/w	Independent
5)	Written in programming lang	Canbe written in Normal language

Advantage of Algo

1) Effective Communication (a)written in natural English
2) Independent of prog lang Provide understanding
3) Easy & efficient coding Provide blueprint of algo
4) Gives a prototype

Dis

- Difficult to understand complex problems
- Time Consuming

Scanned with OKEN Scanner

Problem Development Steps

Problem Definition — solving a prblem require clear understanding of problem
Development of Model — Think of best approach
Specification of Algo
Design Algo
Check its Correctness
Analysis
Implementation
Program testing
Documentation

Analysis of Algo - Donald Knuth.

In theoritical Analysis, we estimate complexity in asymptotic sense. Analysis is the determination of amount of time & space required to execute algo.
The efficiency of running time of algo is stated as a function relating the input length to no. of steps, known as time complexity, or volume of memory, known as space complexity. Main concern of analysis is required time or performance
types of analysis -
Worst case - max no. of steps
Best case - mimim no. of steps
Average case - avg no. of steps
} on any instance of size a.

Asymptotic Notation

describe behavior of function as input varies
are mathematical notations used to describe running time of algo
used to represent complexity of algo
Using this we can conclude best, worst & average case.

It is input bound i.e if no input given then it work in constant time

Upper bound | Worst Case

Big O Notation O(n)
↳ Input

It represents the upperbound of runtime of an algo. It calculates longest time an algo can take for its execution i.e it is calculated for worst case time algo

In this two function f(n) and g(n) are defined on a set of positive or real numbers that are not bounded Here g(n) is strictly positive of every value of n

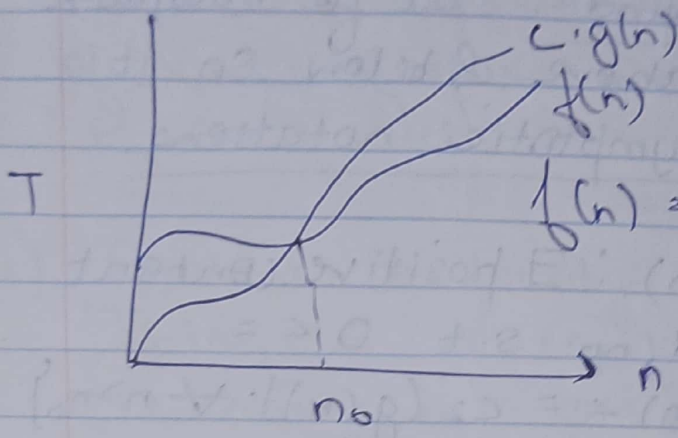
$$f(n) = O(g(n))$$

To calculate we need to count iterations

Scanned with OKEN Scanner

which is represented as n and f(n) is height order of that function.

A function f(n) is said to be O(g(n)) denoted $f(n) \in O(g(n))$ if f(n) is bounded by a some constant multiple of g(n) for all $n > n_0$



$f(n) = O(g(n))$

n = Steps
c = constant

merge sort - O(log n), Baki sari O(1)

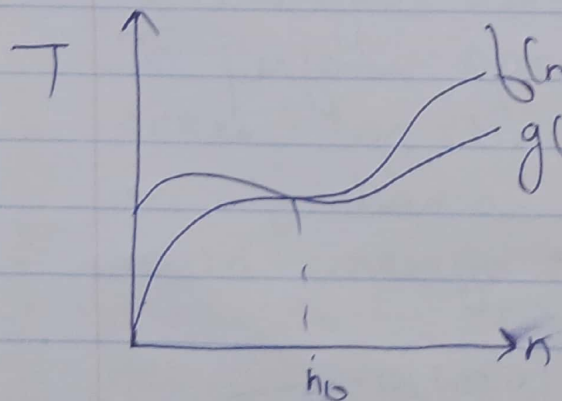
2) Omega notation (Ω) Lower bound Best case

It is a formal way to express Lower bound of an algorithm's run time. It measures best case complexity

$f(n) = \Omega(g(n))$

$\Omega f(n) \geq \{ g(n) :$ there exists $c > 0$ & n_0 such that $g(n) \leq c \cdot f(n)$ for all $n > n_0$

$n_0 \geq 1$



when $n \geq n_0$

useful to find lower bound astime complexity of algo

Theta (Θ) Notation

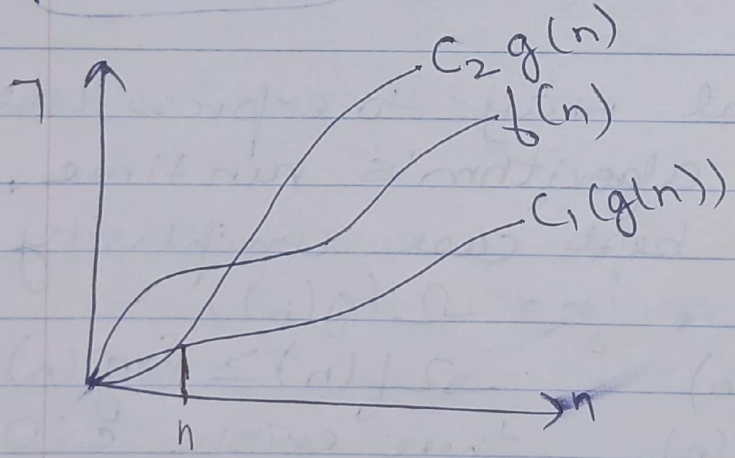
It is a formal way to express both Lower and upper bound of an algo's running time.

It is tightly bound. It bounds a function from above & below so it defines exact asymptotic notations

$\Theta(g(n)) = \{ f(n) : \exists$ positive constant c_1 and c_2 and n_0 s.t $0 <= c_1 g(n) <= f(n) <= c_2 (g(n)) \forall n > n_0 \}$

$c_1 g(n)$ — Lower bound

$c_2 (g(n))$ — upper bound



Complexity

Algorithm complexity measures how many steps are required by algo to solve a problem. It is a function that describe algorithms complexity in terms of amt of data

It computes amt of time & space for n inputs

To access complexity, the order (approximation) of count of operation is considered instead of exact steps.

1) Constant Complexity.

It imposes a complexity of O(1)

It undergoes execution of constant no. of steps for solving a given problem

It takes same no. of steps regardless of input size

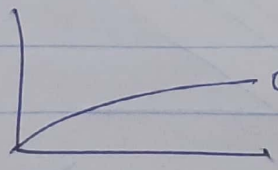
↑ n input

2) Logarithmic

It imposes a complexity of O(log(n)).

It undergoes execution of log(n) steps.

To perform operations on N elements, it takes log base as 2. for n=1000000 an algo has a complexity of O(log(N)) would undergo 20 steps.



3) Linear

It imposes a complexity of O(N)
It executes same no. of as that of no. of elements. For 500 elements, it takes 500 steps.
Not a good complexity.

4) Quadratic

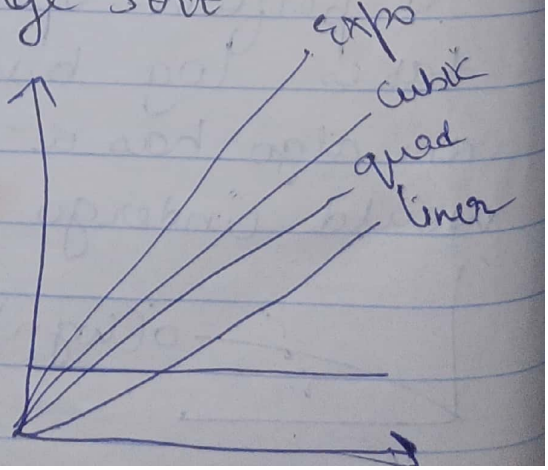
It imposes a complexity of $O(n^2)$. For n input data size, it undergoes order of n^2 count of operations. If n = 10, no. of steps = 100.

5) Cubic complexity

It imposes a complexity of $O(n^3)$
For n input data, it executes n^3 steps on n elements

6) Exponential Complexity - it imposes a complexity of $O(2^n)$, $O(N!)$, $O(n^k)$, occurs when growth rate doubles with each addition to input(n). ex merge sort

Order of all complexities
$1 < \log n < n < n\log n < n^2 < n^3 < 2^n < n!$



Scanned with OKEN Scanner

Algorithm design techniques.

1) Divide and conquer Approach
In this, a problem is divided into smaller problems, then smallest problems are solved independently & then solutions of smaller problems are combined into a solution for Larger problem.
3steps
1) Divide problem 2) Solve subproblem
3) Combine solution into original problem
Top Down Approach
It supports parallelism, run on Multiprocessor
Algo designed using Recursion, more memory req.

2) Greedy Technique

Greedy algo build a solution part by part choosing the next part in such a way that it gives immediate benefit. This never reconsider choices taken previously
Mainly used to solve optimization problem
Optimization problem is one in which we are given a set of input values which are required either to be maximized or minimized (known as objective) i.e some constraints and conditions.

Greedy algo always makes the choice looks best at the moment to optimize given objective.
It doesnot always guarentee optimal solution, but generally produces a solution that is very close to optimal.

Area of application
1) finding shortest path p/w two vertices using Dijkstra algorithm.
2) finding minimal spanning tree in graph using Prim's/Kruskal's algorithm.

3) Dynamic Programming
It is a bottom up approach, we solve all possible small problem and then combine them to obtain solution for bigger problem.
Used for solving a complex problem by first breaking into a collection of simpler subproblems

It is related to optimization problem.

Scanned with OKEN Scanner

4) Branch & Bound
In this a given subproblem, which cannot be bounded, has to be divided into atleast 2 new restricted subproblems. These methods are for global optimization in non-convex problems. These are slow, however in worst case they require effort.

5) Randomized Algorithm - An algo that is allowed to access a source of independent, unbiased random bits to use these random bits to influence its computation.
It uses random no. atleast once during the computation to make a decision.

6) Backtracking Algorithm
It tries each possibility until they find the right one. It is depth first ~~approach~~ search. During search, if an alternative doesn't work, then ~~brac~~ backtrack to choice point, the place which presented different alternatives, and tries the next alternative.

Scanned with OKEN Scanner

Page No.
DATE: / /

Elementary Operation

An elementary operation is one whose execution time is bounded by a constant for a particular machine & programming lang.

The complexity of an algo is often measured through no. of elementary operations that algo requires to arrive at an answer

To have optimization problem
Min cost, Max profit

feasible sol
that follow
constraints.

Greedy Algo solves a subset of optimization problem

It is one of the strategies like Divide & conquer to solve a problem.
It is used for solving optimization problem

Optimization problem - problem that demands either maximum or minimum results.

Simple & straight forward method. Not an algo but a technique.
Decision taken on the basis of currently available information without worrying about effect of current decision in future.
This technique is used to determine feasible solution that may or may not be optimal
give immediate benefit

- The feasible solution is a subset that satisfy given criteria
- Optimal solution is a sol which is best & most favourable in subset.

Characteristics

- Algo creates two sets where one set contain all chosen item & another contain rejected items
- A greedy algo makes good local choices in the hope that solution should either feasible or optimal.

Among all algorithmic approach, greedy prblm is simplest. Decision on current info

Components

- Candidate Set – solution created from set of candidate
- Selection function – function used to choose candidate or subset to add in sol (best candidate)
- Feasibility func – used to determine whether candidate or subset be used to contri to solution
- Objective func – used to assign value to sol or partial solution
- Sol func – used to intimate/indicate whether complete func has reached out or not

Application

→ Find Shortest Path
→ find minimum spaning tree using prism's algo
→ used in job sequencing with deadline

seled items that fit into Knapsack. Subject to constraints. with goal of maximizing total input choosed

Knapsack Algorithm

It is a classic optimization problem in CS and is often studied in context of DAA.

Here knpsack is like a container of a bag. Suppose we are given some items which have some weights or profits. we have to put items in Knapsack in such a way that total value produces maximum ~~obs~~ profit.

There are several variations but ~~knapsack~~ most common is 0/1 Knapsack problem. In this, each item can be either included in the collection or not (hence the name 0/1) goal is to maximize ~~total~~ value of selected item by staying within weight capacity.

It is NP-hard, i.e no known algo can solve problem optimally in polynomial time

Knapsack problem is used in dynamic prog, greedy algo, branch n bound algo

2 types - 0/1, fractional.

0/1

If we have 2 item 2 kg & 3 kg
If we pick 2 kg then we cannot pick 1 kg from 2 kg item (i.e not divisible) we have to pick 2 kg completely.

→ Either pick the item or leave.

This is Solved by dynamic programming.

Fractional

Means we can divide the item.
From 3 kg we can pick 2 kg & leave 1 kg

→ Solved by greedy approach.

See ex - of 0/1

Fractional →

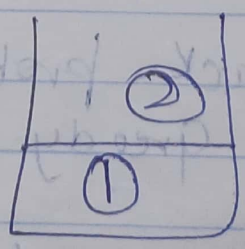
Obj	1	2	3	Weight
Profit	25	24	15	Capacity = 20
weight	18	15	10	

is mese 2 unit hi jasakte

1) Greedy abt Profit

$25 + \frac{2}{15} \times 24$

$= 25 + 3.2 = 28.2$



→ Not Optimal

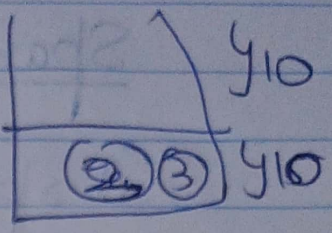
Scanned with OKEN Scanner

Look for low weight

2) Greedy abt weight

$15 + \frac{10^2}{15} \times 24^8$

$= 15 + 16 = 31$ ⇒ Not Optimal



3) Greedy about both

Obj	1	2	3
P	25	24	15
W	18	15	10
	1.3	1.6	1.5

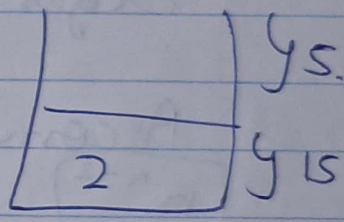
Find Ratios, Sort in descending Order jo badha h pehle ye jaega

So first 1.6 goes

$24 + \frac{8}{10} \times 15$

$= 24 + 7.5$

$= 31.5$ Maximum



Optimal

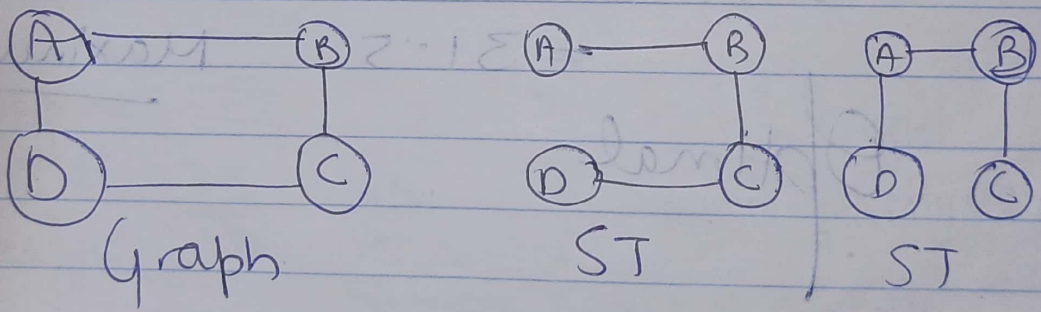
Graph - a graph can be defined as a set of vertices & edges. Undirected, directed, connected.

Spanning Tree

A spaning tree is a subgraph of an undirected connected graph. It include all vertices along with least no. of edges. If any vertex is missed it not a spaning tree.

* It is a subset of graph that doesnot have cycles.
It consist of (n-1) edges, where n is no vertices.
Edges may or may not be weighted

A complete graph can have n^{n-2} spanning trees.



Weighted ST

AMST or minimum weighted ST is a subset of graph of edgs of connected, edge weighted connected graph with minimum total weight of edges.

Prim's Algo

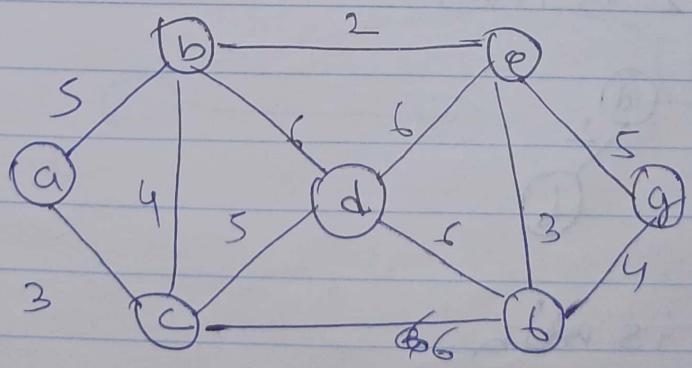
It is a greedy algo used to find MST of a graph.

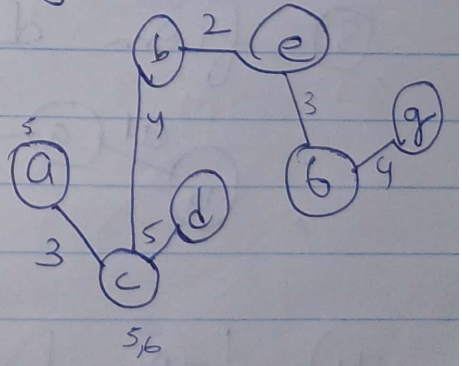
It finds the subset of edges that include every vertex of graph such that sum of weight of edges are minimum.

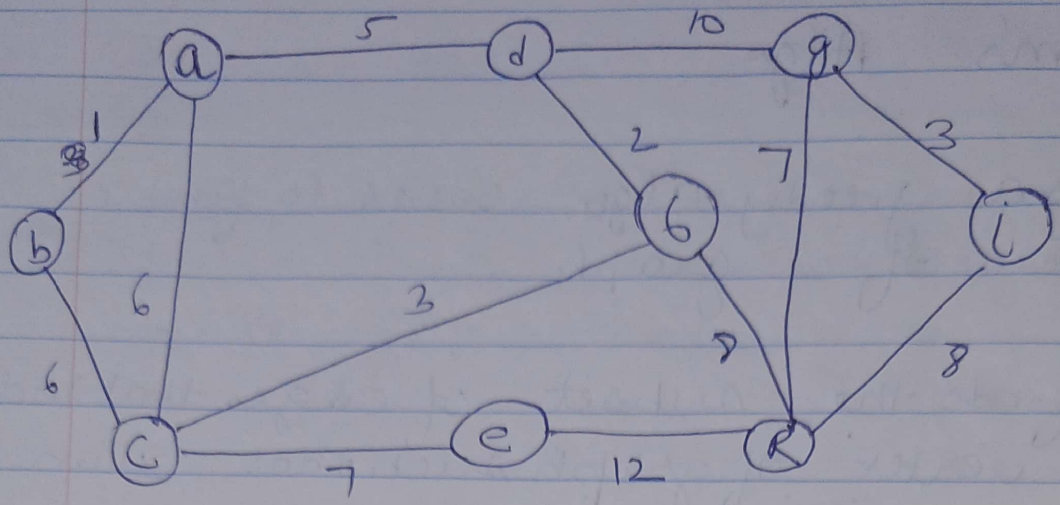
It starts with any single node & explore all adjacent vertices that are connected to that edge. The edge with minimum weight that does not makes a cycle is selected.

Applications

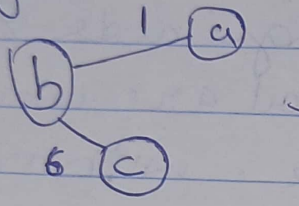
- N/W designing
- Lay down electrical wiring cables
- Make n/w cycles





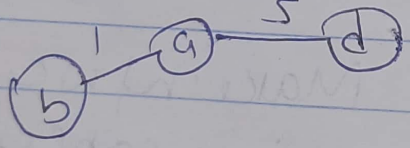


1) first select b as starting node

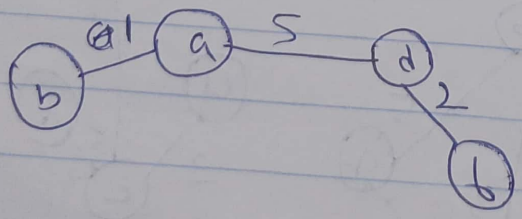


so slect with 1 as breight

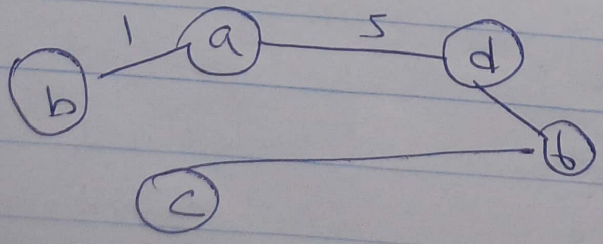
2) from a 5 is min



3) from d 2 is min

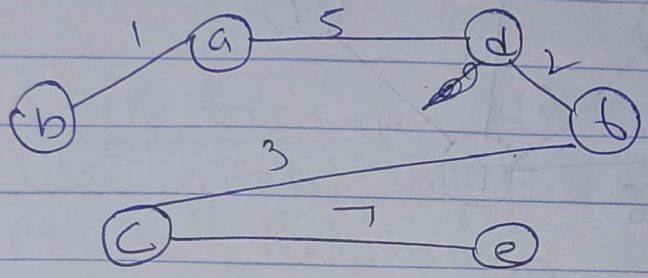


4) from b 3 is min

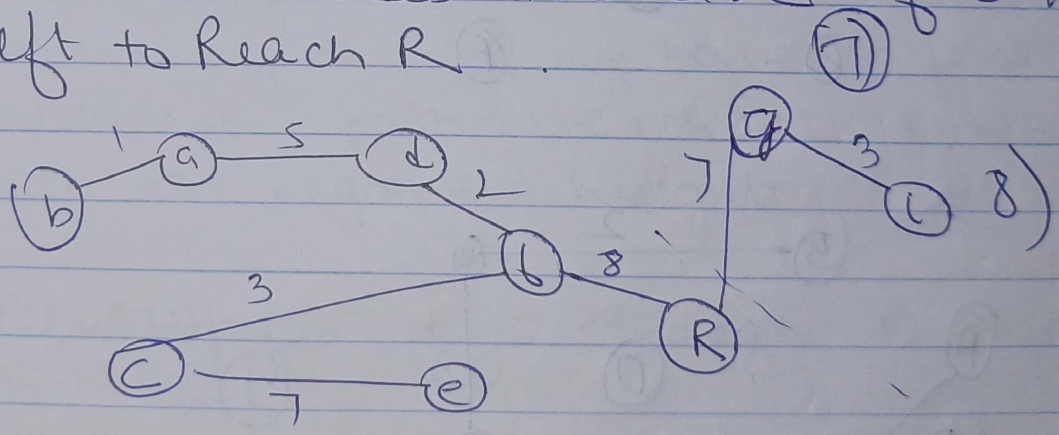


Scanned with OKEN Scanner

5) from c 6, 3, 7 are weight 3 is already used, so 6 is next minimum, but if we select 6 it will form cycle, so we go back to all minimum weights of previously visited nods.
If we go from b to c it will again make cycle. so choose 7 from c to e

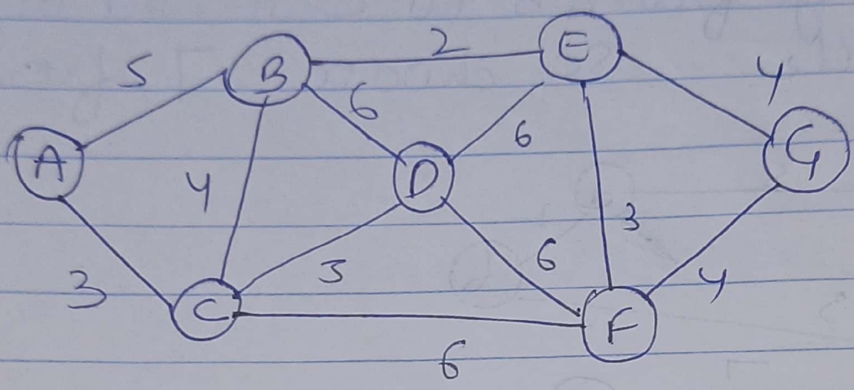


6) Now from e next is 12 but from visited nodes we have f's weight left to Reach R.

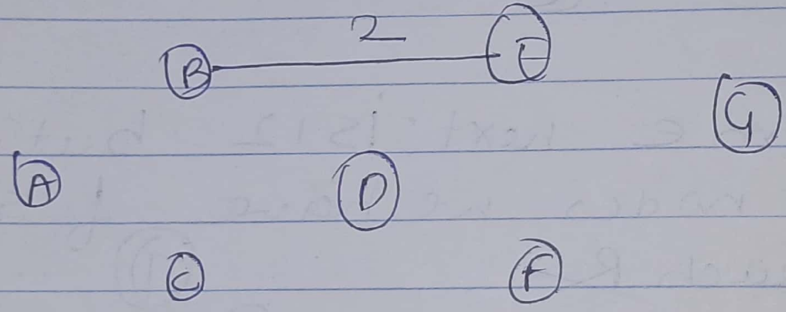


In Kruskal

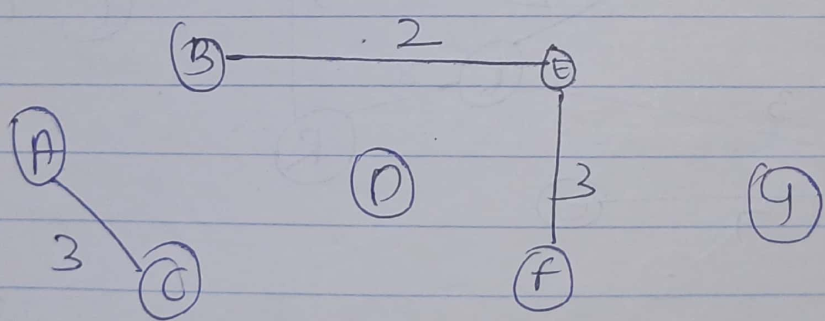
we join edges by arranging weights, it may be disconnected initially but at end it forms a connected graph



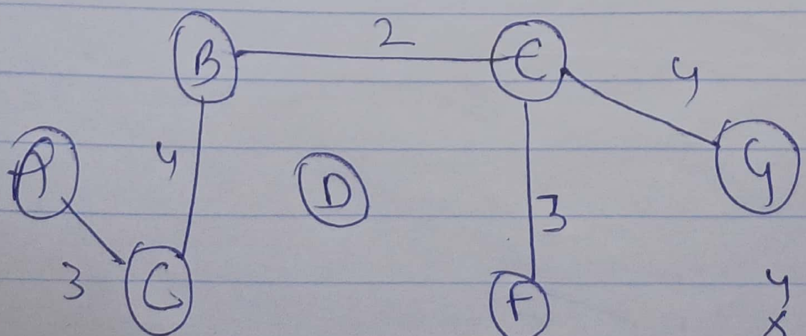
1)



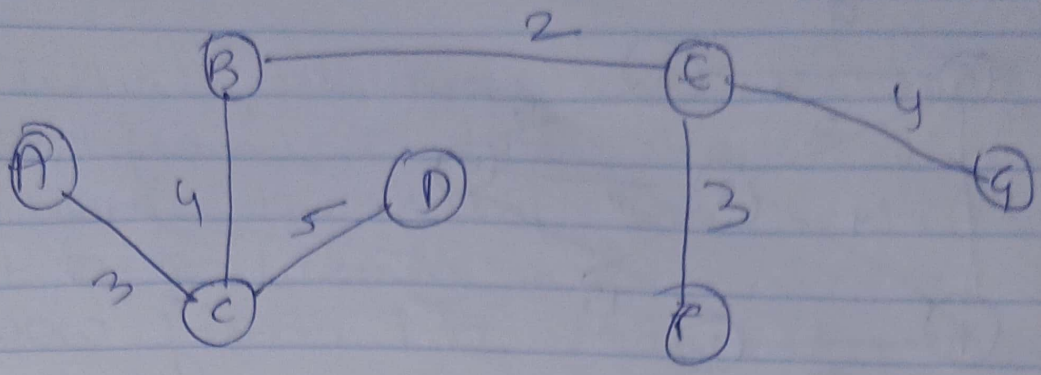
2)



(3)



4 coz
✗ form cycle



Done

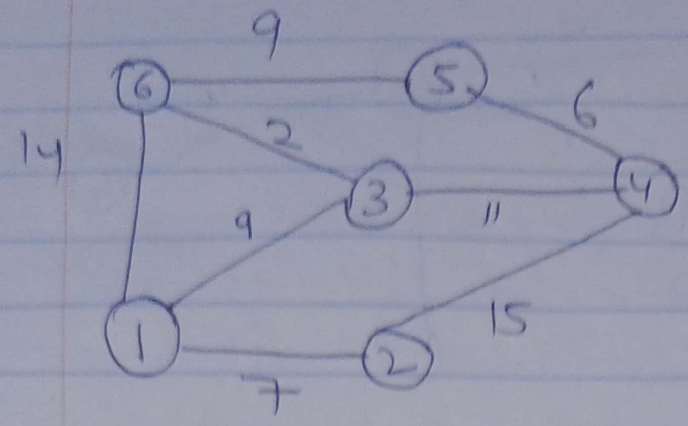
Dijkistra - Mostly used in Google Map
Uses greedy approach.
Provide Relaxation.

It is a single source shortest path algo
Here single source means that only one source is given, & we have to find shortest path to reach all vertices.
With non negative weights

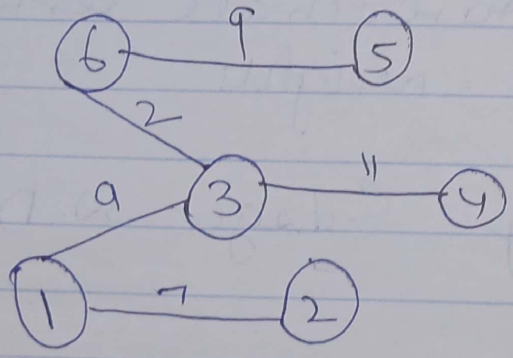
Revelopld by Edsger W. Dijkstra in 1956

- Digital Mapping Service in Google Maps
- Various Routes connected to show shortest path
- Telephone network - vertices Represent switching stations & edges transmission link weight is bandwidth.
- flighting Agenda, IP Routing (OSPF)

Scanned with OKEN Scanner



	Source	Destination				
	1	2	3	4	5	6
1)		∞	∞	∞	∞	∞
2)	1,2	7	9	∞	∞	14
3)	1,2	7	9	22 (1-2-4)	∞	14
4)	1,2,3	7	9	20 (1-3-4)	∞	11 (1-3-6)
5)	1,2,3,6	7	9	20	20 (1-3-6-5)	11
	1,2,3,6,4	7	9	20	20	11
	1,2,3,6,5.					



1) Initially distance is calculated as ∞ for all start from 1 & find shortest direct path
2) 7 being the shortest we select it & 1, 2 & become source
Now from 1, 2 reach next shortest destination
3) When in second, we got 9 as shortest path so we added 3 to source now we can reach ~~other~~ ~~vertices~~ vertices using 1, 2, 3 path. Or any path made of 1, 2, 3.
4) From 1, ~~2~~ 3, 6 we reached 6 through shortest path so Relaxed 14 to 11
5) Reaches through 1-3-6-5 as both 4 & 5 have same cost 20 ~~as~~ so choose any

Formulas

$$d(x,y) = d(x) + c(x,y) < d(y)$$
$$= (0 + 7) < \infty$$
$$= 7 < \infty$$
$$d(x,y) = 7$$

Scanned with OKEN Scanner

END TERM

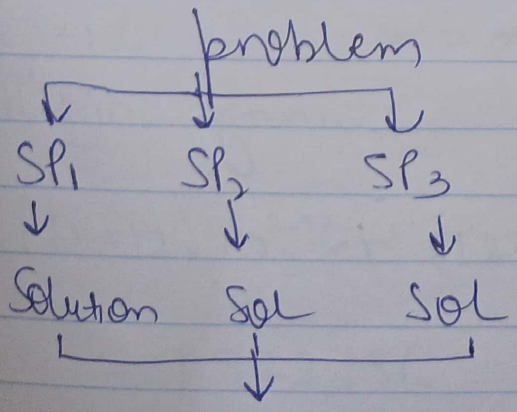
Divide & Conquer Approach

It is an algorithmic pattern. In algorithmic methods the design is to take dispute on huge input, break input into minor pieces, decide the problem on each of small piecce & then merge into global solution. This is Divide & Conquer Strategy

3 steps:

- Divide - problem divided into Subproblem Involves Spitting input data
- Conquer - Subproblems are solved independently Stepinvolves applying same algo recusively to each Subploblem
- Combine - Solution to subproblem are combined to obtain the solution to original problem.

Ex - Merge sort, Quit Soort, Binary Search

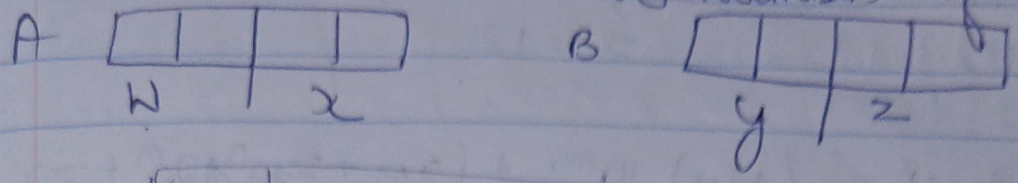


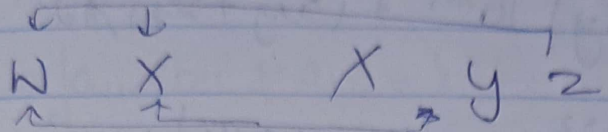
fundamental:
- Relational formula formula thatwe generate from given technique.
- Stopping Condition - when we break using D&C we need to know time to apply D&C It is condition where we need to stop our recursion Steps of D&C

$= \underbrace{(a_1 * b_1)10^n}_{c_2} + \underbrace{(a_1 b_0 + a_0 b_1)10^{n/2}}_{c_1} + \underbrace{(a_0 b_0)}_{c_0}$

Multiplication of large no.

Let A & B be two numbers of 4 digit





Wy xy

zw zx

Wy * (xy+Wz) zx

$A = W \cdot 10^m + x$

$B = y \cdot 10^m + z$

Product $= (Wy)10^{2m} + (Wz + xy)10^m + xz$

Ex A = 1234 B = 1234

W x y z

$m = n/2 = 4/2 = 2$

W = 12 x = 34

y = 12 z = 34

Put in formula

$\rightarrow (12 \times 12)10^{2\times2} + (12 \times 34 + 34 \times 12)10^2 + 34 \times 34$

$= 1522756.$

96878000000
3584160000
14806
97236430806

Ex – A = 1212 B = 1212
wx yz

w = 12 y = 12
x = 12 z = 12

Product = $(wy)10^{2m} + (xy + zw)10^{m} + zx$

$= (12 \times 12) 10^{2\times2} + (144 + 144) * 10^{2} + 144$

$= (144)10^{4} + (288) 10^{2} + 144$

= 1440000 + 28800 + 144

144 + 144 = 288

1440000
28800
144
1468944

$a_1 b_1 \cdot 10^n + (a_0 b_1 \cdot a_1 b_0) 10^{n/2} + a_0 b_0$

Ex – a = 123456 (w = 123 = a_1, x = 456 = a_0) b = 786932 (y = 786 = b_1, z = 932 = b_0)

$C_2 10^n + C_1 10^{n/2} + C_0$

$= (123 * 786) 10^{6} + (456 * 786 + 123 * 932) 10^{3} + 456 * 932$

$= 96678 \times 10^{6} + 358416 \times 10^{3} + 14806$

786 × 123 = 2358 + 1572 + 786 = 96678

456 × 786 = 2736 + 3648 + 3192 = 358416

123 × 932 = 46

MATRIX MULTIPLICATION

Brute force Algorithm

8 mul 4 add

$$\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} * \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix}$$

$$= \begin{vmatrix} a_{11} * b_{11} + a_{12} * b_{21} & a_{11} * b_{12} + a_{12} * b_{22} \\ a_{21} * b_{11} + a_{22} * b_{21} & a_{21} * b_{12} + a_{22} * b_{22} \end{vmatrix}$$

Efficiency $O(n^3)$

Strassen Matrix Multiplication (2X2)

$$\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} = \begin{pmatrix} C_1 = E+I+J-G & C2 = D+G \\ C3 = E+F & C4 = D+H+J-A \end{pmatrix}$$

D = A1 (B2 - B4)
E = A4 (B3 - B1)
F = (A3 + A4) B1
G = (A1 + A2) B4
H = (A3 - A1) (B1 + B2)
I = (A2 - A4) (B3 + B4)
J = (A1 + A4) (B + B4)

Scanned with OKEN Scanner

$$A = \begin{matrix} A1 & A2 \\ \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \\ A3 & A4 \end{matrix} \qquad B = \begin{matrix} B_1 & B_2 \\ \begin{pmatrix} 1 & 1 \\ 2 & 2 \end{pmatrix} \\ B_3 & B_4 \end{matrix}$$

$D = 1(1-2) = -1$

$E = 4(2-1) = 4$

$F = (3+4)1 = 7$

$G = (1+2)2 = 6$

$H = (3-1)(1+1) = 2 \times 2 = 4$

$I = (2-4)(2+2) = -8$

$J = (1+4)(1+2) = 5(3) = 15$

$C_1 = E + I + J - G = 4 + (-8) + 15 - 6$

$= 19 - 14 = 5$

$C_2 = D + G = -1 + 6 = 5$

$C_3 = E + F = 4 + 7 = 11$

$C_4 = D + H + J - F = -1 + 4 + 15 - 7$

$= 19 - 8 = 11$

$$\begin{pmatrix} 5 & 5 \\ 11 & 11 \end{pmatrix}$$

Scanned with OKEN Scanner

for Greater than 2x2

$$\begin{pmatrix} c_{00} & c_{01} \\ c_{10} & c_{11} \end{pmatrix} = \begin{pmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{pmatrix} \begin{pmatrix} b_{00} & b_{01} \\ b_{10} & b_{00} \end{pmatrix}$$

$$= \begin{pmatrix} M_1 + M_4 - M_5 + M_7 & M_3 + M_5 \\ M_2 + M_4 & M_1 + M_3 + M_2 + M_6 \end{pmatrix}$$

$M_1 = (A_{00} + A_{11}) * (B_{00} + B_{11})$ same as J
$M_2 = (A_{10} + A_{11}) * B_{00}$ same as F
$M_3 = A_{00} * (B_{01} - B_{11})$ D
$M_4 = A_{11} (B_{10} - B_{00})$ E
$M_5 = (A_{00} + A_{01}) * B_{11}$ G
$M_6 = (A_{10} - A_{00}) * (B_{00} + B_{01})$ H
$M_7 = (A_{01} - A_{11}) (B_{10} + B_{11})$ I

$$A = \begin{matrix} A_{00} & A_{01} \\ 2 & 1 \\ 3 & 4 \\ A_{10} & A_{11} \end{matrix} \qquad B = \begin{matrix} B_{00} & B_{01} \\ 5 & 2 \\ 1 & 2 \\ B_{10} & B_{11} \end{matrix}$$

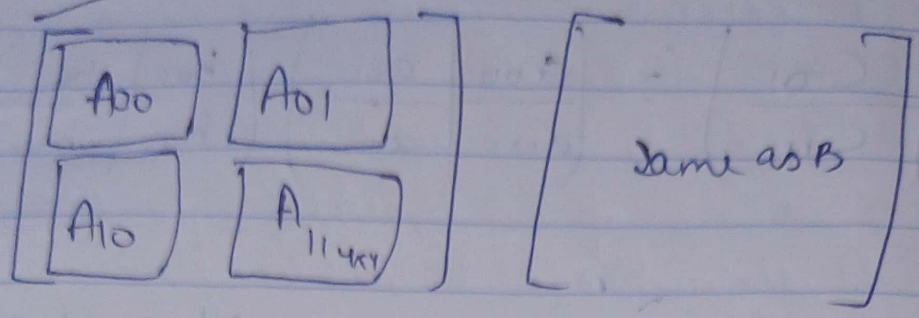
$M_1 = 2+4 * 5+2 = 6 * 7 = 42$
$M_2 = (3+4) * 5 = 7 * 5 = 35$
$M_3 = 2 * (2-2) = 0$
$M_4 = 4 (1-5) = -16$
$M_5 = (2+1) * 2 = 6$
$M_6 = (3-2) (5+2) = 1 (7) = 7$
$M_7 = (1-4) 1+2 = -3 (3) = -9$

$$\begin{pmatrix} M_1 + M_4 - M_5 + M_7 & M_3 + M_5 \\ M_2 + M_4 & M_1 + M_3 + M_2 + M_6 \end{pmatrix}$$

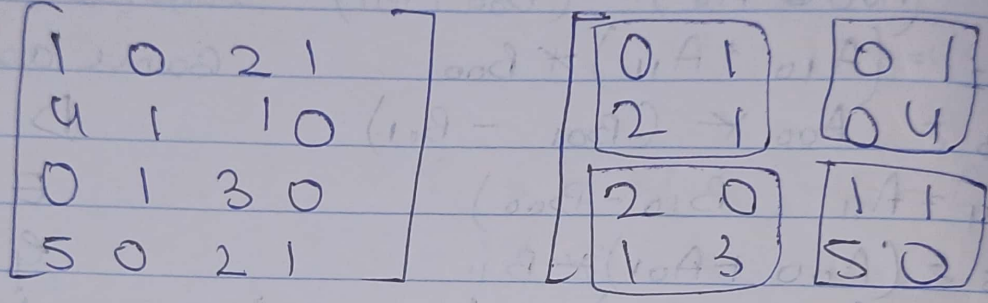
$$\begin{matrix} 11 & 6 \\ 19 & 16 \end{matrix}$$

Scanned with OKEN Scanner

4x4



first find ALL M taking A_{00} as whole 2x2 matrix.



$M_1 = (A_{00} + A_{11}) * (B_{00} + B_{11})$
$M_2 = (A_{10} + B_{11}) * B_{00}$
$M_3 = A_{00} * (B_{01} - B_{11})$
$M_4 = A_{11} * (B_{10} - B_{00})$
$M_5 = (A_{00} + A_{01}) * B_{11}$
$M_6 = (A_{10} - A_{00}) * (B_{00} + B_{01})$
$M_7 = (A_{01} - A_{11}) * (B_{0} + B_{11})$

$A_{00} = \begin{matrix} 1 & 0 \\ 4 & 1 \end{matrix}$
$A_{01} = \begin{matrix} 2 & 1 \\ 1 & 0 \end{matrix}$
$A_{10} = \begin{matrix} 0 & 1 \\ 5 & 0 \end{matrix}$
$A_{11} = \begin{matrix} 3 & 0 \\ 2 & 1 \end{matrix}$

Scanned with OKEN Scanner

$B_{00} = \begin{matrix} 0 & 1 \\ 2 & 1 \end{matrix}$ $B_{01} = \begin{matrix} 0 & 1 \\ 0 & 4 \end{matrix}$

$B_{10} = \begin{matrix} 2 & 0 \\ 1 & 3 \end{matrix}$ $B_{11} = \begin{matrix} 1 & 1 \\ 5 & 0 \end{matrix}$

$$M_1 = \begin{bmatrix} 1 & 0 \\ 4 & 1 \end{bmatrix} + \begin{bmatrix} 3 & 0 \\ 2 & 1 \end{bmatrix} * \begin{bmatrix} 0 & 1 \\ 2 & 1 \end{bmatrix} \begin{bmatrix} 1 & 4 \\ 5 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 4 & 0 \\ 6 & 2 \end{bmatrix} * \begin{bmatrix} 1 & 2 \\ -7 & 1 \end{bmatrix} = \begin{bmatrix} 4+0 & 8+0 \\ 6+14 & 12+2 \end{bmatrix} = \begin{matrix} 4 & 8 \\ 20 & 14 \end{matrix}$$

Similarly calculate till M_7

then Calculate $\begin{matrix} C_{00} & C_{01} \\ C_{10} & C_{11} \end{matrix}$

Strassens Matrix Mul is a divide & conquer approach to solve Matrix Multiplication problems.

Time complexity - $O(n^3)$ as it takes two loops to multiply

It reduce total number of operations

Median in Unsorted Array Algo

1) Define a function named med that tak input array as parameter.
2) Inside function check length of array If it is less or equal to 1, return element as median
3) If length > 1, choose a pivot element (selected Randomly)
4) Now place all smaller elements to left & greater Element than pivot to Right
5) Determine pivot index
6) If index of pivot is length/2, return pivot as median
7) If pivot index > length/2 call med func on Right sub array else on Left sub array
8) If length of array is odd return mid value, if it is even return average of left & Right mid value

Matrix Multiplication Algo
Strassen's Algo

1) Define a function named "strass_mul" that take two matrices as input.
2) Inside function check Matrix Compatibility for Multiply. If not return Error.
3) If dimension are larger than 1*1 divide it to equal submatrices A11, A12, A21, A22, B11, B12, B21, B22.
4) Compute 7 products required to find C11, C12, C21, C22 and construct a new matrix C
5) End the function

Write all formula of Strassen's Mul.

All Algos

1) Linear Search

- Start at beging of list
- Set index variable as 0
- Take first element & compare it with target value
- If found, return the index
- If not fount move to next element & increment index by 1.

Scanned with OKEN Scanner

- Traverse each element one by one & if target value is found return index
- Else print value Not found.

Binary Search

~~1) Start with sorting the array~~

or [1) Define a function that takes input & target value as parameter

2) Sort the array & find Middle Element, if target value is equal to middle value, return middle value index

3) Else if the target value is smaller than middle value Search in left subarray, else in Right subarray.

4) If function is set on left subarray set end = mid - 1, else if function is set to Right subarray, Set start = mid + 1.

Selection Sort

1) Start Selection - sort function which takes array as parameter.
2) Iterate over array fuorm begining to end - 1 (i)
3) Set min index as i.
   Compare i with i+1 and swap with minimum number
4) End function.

Merge Sort

1) Start mergesort function which taks input as array
2) If length of array is 1, return array
3) Else divide the array into two equal halves by finding middle index.
4) Assign left half to left-subarray & Right to right-subarray
5) Create empty array to store sorted elements.
6) Apply divide & conquer on each array formed till each element is separated using a pointer pointing to small Element.
7) Return the array after prossesing sub arrays. as result.

Scanned with OKEN Scanner

See a end

Quick Sort

1) Start Quick sort function which takes input array as parameter
2) Check if start index is less than end index. If not return 0.
3) Choose a pivot element i.e selected Randomly
4) Partion array based on pivot Create two variable i & j to mark start & end position respectively
5) When i is less than or equal to j follow:
- Increment i until arr[i] is greater than pivot.
- Decrement j until arr[j] < pivot
- If i ≤ j swap element at arr[i] & arr[j]
- Recursively call quick sort in Left & Right subarrays
- End function.

Bubble Sort

1) Start bubble_sort function that accepts an input array as parameter
2) Set a function swap to call whenever we need to swap values Inside the function swap values using temp variable
3) Iterate the array & compare first element with other elements. If any smaller element is found call swap function. Compare 1 with i+1.
4) End inner loop.
5) Repeat same inner lop with other elements till all number larger are placed at last
6) End function.

Insertion

1) Start function taking array as parameter
2) Iterate over each element starting from second ie i=1 upto last
3) let current ele = key & j = key - 1
4) While j is greater than or equal to 0 move j to next position
5) Place key at j+1
6) Repeat 2 to 5
7) End func.

Scanned with OKEN Scanner

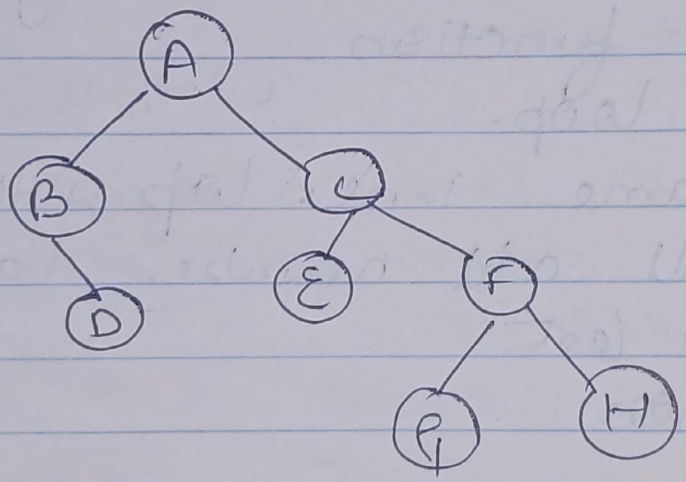
Binary Tree Traversal

is a rooted tree or ordered tree in which every node has 2 ~~ne~~ chi

Inorder ~~L M R~~ Left Mid Right

Preader Root left Right

Post ~~R~~eft Right Root



2 In B D A E C G F H

1 Pre A B D C E F G H

3 Post D B ~~R~~ E G H F C A

Jenny Nadam

Expression Trees

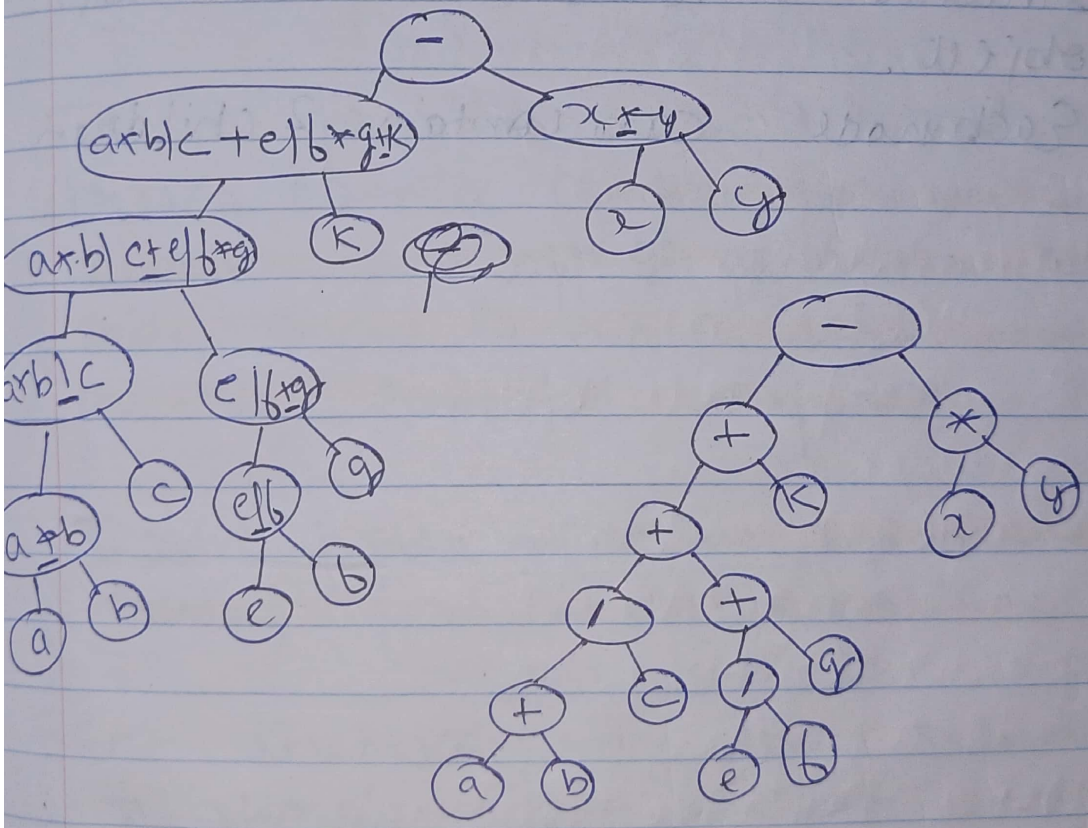
Binary Exp Tree (from Infix)

^	R - L
*/	L - R
-+	L - R

a * b / c + e / b * g + K - x * y

Root.

Root will be less precedence with coming in last, e -.



Jese + or - Ki sabse kam hoti h value to hum last wala +/- Root bnaenge (i.e with least value).

From postfix

a+b/c+e/f*g+k-x+y

by stack.

A binary search tree is a binary tree That can be defined by a linked da structure in which a node is an object.

Each node can contain 2 children

Scanned with OKEN Scanner

Dynamic Programming

It is a technique that breaks the problem into subproblems & saves the result for future purpose so that we do not need to compute Result again. It is mainly an optimization over plain Recursion.

Whenever we see Recursive calls using same input, we can optimize it using Dynamic programming.

It is a problem solving technique that breaks down a complex problem into smaller overlapping subproblems solved only once & reused when needed avoiding repeated computing

It solve proble in bottom up manner starting from smallest subproblem

Ex- Knapsack, travelling salesman, & shortest path problem, etc.

C[1,1] = C[0,1] + C[0,0]
= ∞(NaN) + 1
= 1

C[2,5] = C[1,5] & C[1,4]

Binomial Coefficient

A Binomial Coefficient is an alg
expression that contains that co
terms $(a+b)$ let.
Such as
$(a+b)^2 = 1a^2 + 1b^2 + 2ab$
where no. that appear as coe
of terms in binomial expressio
Binomial Coefficient.

Binomial Coefficient of n, K
^nC_K or $C(n,K)$ $\binom{n}{x}$

Mathematically ⇒

$$^nC_K = \frac{n!}{K!(n-K)!}$$

$$^5C_3 = \frac{5!}{3!(5-3)!} = \frac{5!}{3!(2!)} =$$

$$5 \times 2 = 10$$

Using Dynamic Programming

$^5C_3$ $n = 5$ $K = 3$

Scanned with OKEN Scanner

i = 0 to 5
j = 0 to min(i, K)

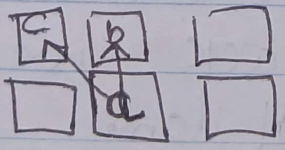
i ↓ K →	0	1	2	3
0	1	X		
1	1	1	X	
2	1	2	1	X
3	1	3	3	1
4	1	4	6	4
5	1	5	10	10

1) In this as soon as (i) j > K(i) stop there Wherever K values one > than n value put across

2) Fill entire first column with 1

3) Diagonal blow X fill with 1 (black)

4) Any value will be computed using above value & left diagonal value Sum of them is value at current position



a = b + c.

But in exam we need to show with formula so calculating by formula C[1,1] - C[0,1] + C[0,0]
= X(NaN) + 1
= 1

~~C[2,5] = C[1,5] + C[1,4]~~

Ex Calculate C[5,2) by Formula

$$C[5,2] = C[4,2] + C[4,1]$$
$$= 6 + 4$$
$$= 10$$

For easy calculation use 4 steps
But ACC to formula we solve by this way.

0/1 Knapsack

A Knapsack is like a bage or container. In this we try to fill Knapsack with more weight & less cost, however unlike fractional Knapsack we cannot add a part of item.

Either pick the item (1) completely or leave the item (0)

Scanned with OKEN Scanner

If we solve this problem without dynamic prog. we need to calculate 16 time as for 4 items

Ex- W 3 4 6 5 W=8
P 2 3 1 4 n=4

i = no of weights given, but 0 will include

i ↓ W →	0	1	2	3	4	5	6	7	8
(P,W) 0	0	0	0	0	0	0	0	0	0
(2,3) 1	0	0	0	2 (Profit)	2	2	2	2	2
(3,4) 2	0	0	0	2	3	3	3	3	3
(4,5) 3	0								
(1,6) 4	0								

fill first Row & first colum with 0.

in> When with W=3 (0,1,2) can't be filled so write 0, in W=3, 1 can be filled with 2 as Profit.
in W=4 only 1 item of 3 can but so write 2 & follow same for others. We cannot add 2 times.

$$K(i, C) = \max[(p_i + K(i-1, C-w_i)), K(i-1, C)]$$

$i = 0$ or $C = 0 \Rightarrow 0$

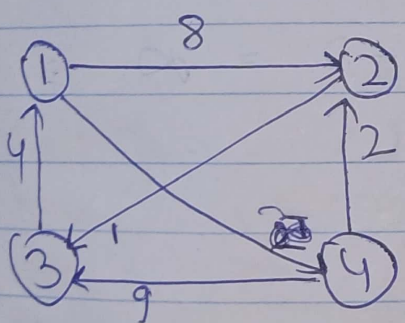
When $w_i > C$ $K(i-1, C)$.

All Pair Shortest Path (Floyd Warshall)

It is a classic algorithmic problem that aims to find shortest path between all pair of vertices of directed or undirected graph.
Algo uses a dynamic prog table to store intermediate Results.
for an array i*j , we initialize a 2d array of size n*n where distance [i][j] respresent shortest distance

Complexity - $VE \log V$

Working



Directed Weighted Graph

we make Distance Matrix, first Do for Direct Edge

D_0 =

	1	2	3	4
1	0	8	∞	2
2	∞	0	1	∞
3	4	∞	0	∞
4	∞	2	9	0

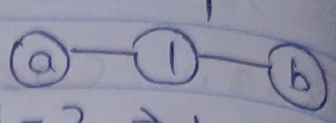
if direction is not there or we can't reach directly Write ∞.

Scanned with OKEN Scanner

D^1 =

	1	2	3	4
1	0	8	∞	1
2	∞	0	1	∞
3	4	12	0	6
4	∞	2	9	0

Includes via 1



1-2 ⇒ 1-1, 1-2

Same

↳ First Row & Column as 1 is already included

Take values from D^0.

for going from 2-3 by including 1 we can go as 2-1 and then 1-3 but 2-1 there is no arrow that gives so we directly go from 2-3

For going from 2-4

2-1, 1-4

⇓

no edge = ∞ ⇒ 2 ∞+2 = ∞

So ∞ is ans

⇒ 3-2 3-1, 1-2

∞ > 4 8 = 12 So take 12

⇒ 3-4 3-1, 1-4

∞ > 4 + 1 = 5 Take 6

⇒ 4-2 4-1, 1-2 So take 2

2 < ∞

⇒ 4-3 4-1, 1-3 Take 9

9 < ∞ + ∞

Scanned with OKEN Scanner

But we can also include 1 as we have matrix for it

via node 2

D_2 =

	1	2	3	4
1	0	8	(9)	1
2	∞	0	1	∞
3	4	12	0	5
4	∞	2	(3)	0

First step fill Second column & second row as D_1.

a) 1-3 = 1-2, 2-3 | 1-4 1-2 2-4

∞ > 8+1 = 9 | 1 < ∞

Take 9 | Take 1

⇒ 3-1 3-2 2-1 | ⇒ 3-4 3-2 2-4

4 < ∞ + ∞ | (as in D1) 5 < ∞

Compare values from previous Matrix

b) 4-1 4-2 1-2 | 4-3 4-2 2-1

∞ = 2 ∞ | 9 > 2 1 = 3 Take 3

D_3 =

	1	2	3	4
1	0	8	9	1
2	(5)	0	1	(6)
3	4	12	0	5
4	(7)	2	3	0

Copy third Row & third Colum

c) 1-2 1-3 3-2 | 1-3 1-3 3-4

8 < ∞ | 1 ∞

d) 2-1 2-3 3-1 | 2-4 2-3 3-4 Stuff

∞ > 1 + 4 = 5 | ∞ 1 ∞ = ✓ 5

e) 4-1 4-3 3-1] But | = 1+5 = 6 But we can go from 3-1-4 = 4+1 = 5

∞ > 3+4 = 7 ✓

For smaller we can go 4-3 ki jagah 4-2-3 = 3

4-2-3

⇒ 4-2 4-3 3-2

2 < 3 + 12

3-1-2

Dy	1	2	3	4
1	0	③	④	1
2	⑤	0	1	6
3	4	⑦	0	5
4	7	2	3	0

Copy same 4th row & column

via 4 but can include 1-2-3 as we have made matrix for them

⇒ 1-2 1-4 4-2

8 > 1 + 2 = 3

⇒ 1-3 1-4 4-3

4-2-3

as 03 - 9 > 1 + 3 = 4

⇒ 2-1 2-4 4-1

2-3 3-4 4-3

1 ⇒ last already

5 < ∞

⇒ 3-1 3-4 4-1

4 < 5

3-1-4

3-2 3-4 4-2

12 > 5 2 = 7

Dy gives all pair shortest path

Floyd- Warshall is used for finding shortest distance through matrices

Algorithm

1) Create a 2D array wist distance [] [] of size n×n, where n is no. of vertices in graph.
2) Initialize distance[] [] array with weights of edges & if there is no direct edge, mark it as infinity.
3) For each vertex K starting from 1 to n
   If distance from i to j is through vertex K is shorter than current distance update distance [i] [j] to distance [i] [K] + distance [K] [j].
4) Repeat this for all other values of K

Matrix Chain Multiplication

A · B · C · D
A = 13×5
B = 5×89
C = 89×3
D = 3×34

M	1	2	3	4
1	0	5785	1530	2856
2	X	0	1335	1845
3	X	X	0	9078
4	X	X	X	0

M = Size of Matrix.
M[1,1] M[2,2] M[3,3] M[4,4] = 0

Scanned with OKEN Scanner

last no. should be equal to first of next element

2) M[1,2] M[2,3] M[3,4]
A B B C C D

= 13×5×5×89 one time
= 13×5×89
= 5785

5×89×89×3
= 5×89×3
= 1335

89×3×34
89×3×34
= 9078

3) Jo ikatha n uske bich wala nhi lena

M[1,3]

A·(B·C) or (A·B)·C
13×5 5×89 89×3 13×5 5×89 89×3

= m[1,1] + m[2,3] + 13×5×3
A BC A·B·C

0 + 1335 + 195
= 1530

m[1,2] + m[3,3] +
13×89×3
= 5789 + 0 + 3471
= 9256

Take Min value
So m[1,3] = 1530

4) m[2,4]
B + (C·D)
~~[2020] + m~~
5×89 (89×3 3×34)
m[2,2], m[3,4] +
5×89×34
= 24208

(B·C) + D
5×89 89×3 3×34
m[2,3] + m[4,4] +
5×3×34
= 1845

Take m[2,4] = 1845

5) m[1,4]

... have 3 combinations

A . BCD

13x5 (13x5x89 89x3 3x34) so 13x5x34

1 x x

m[1,4]

A . BCD

m[1,1] +

m[2,4] +

13x5x34

= 4055

AB CD

m[1,2] +

m[3,4] +

13x89x34

= 54,201

ABC . D

m[1,3] +

m[4,4] +

13x 3x34

= 2856

AB CD

13 5x89 89x3 3x34

x x x

ABC . D

So min is 2856 so m[1,4] = 2856

formula

$$\min[i,j] = \min\{m(i,K) + m(K+1,j) + d_{i-1} * d_K * d_j\}$$

So 2856 no. of Multiplication Required to Nutiply AB CD.

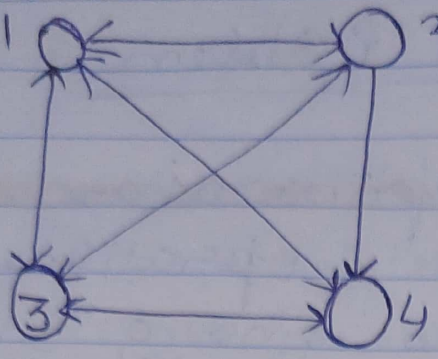
Scanned with OKEN Scanner

Travelling Saleman Problem

TSP is a classic optimization problem in computer science & operations Research. It involves finding shortest possible Route that allows a salesman to visit a set of cities exactly once & Return to starting city.

The salesman finds the shortest rotute to complete tour. The distance b/w each pair of cities is known and the goal is to minimize the total distance travelled

Scanned with OKEN Scanner



	1	2	3	4
1	0	10	15	20
2	5	0	25	10
3	15	30	0	5
4	15	10	20	0

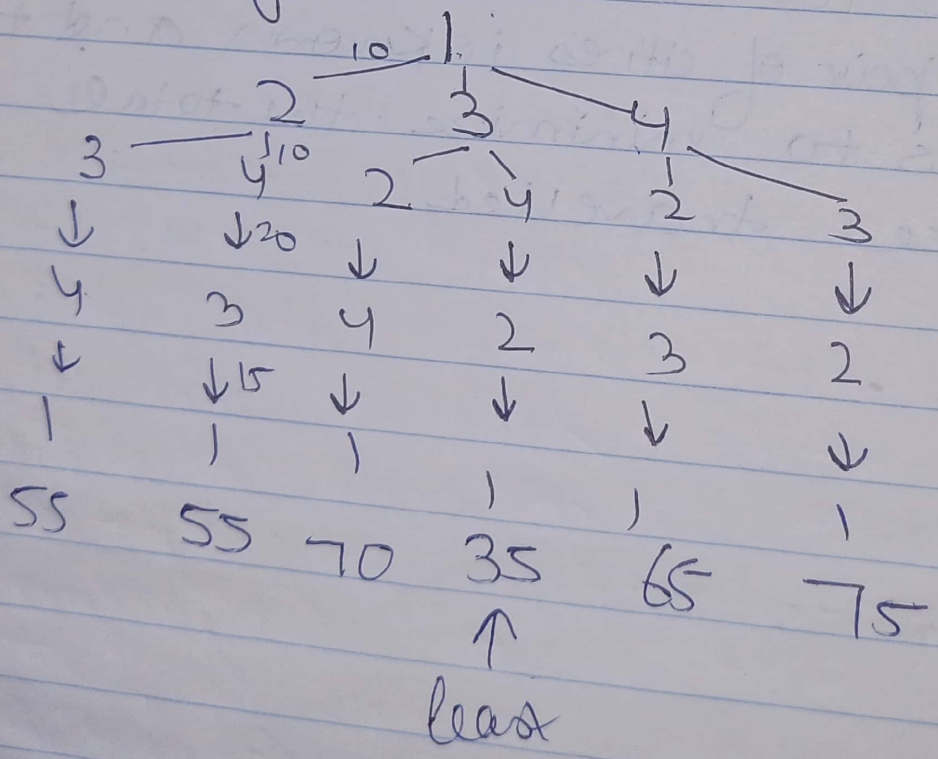
According to Greedy

1 → 2 → 4 → 3 → 1

10 10 20 15

= 55

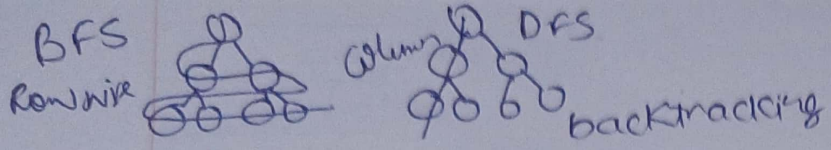
Brute force



Algo

1) Travelling salesman problem take a graph G(V, E) as input & give other graph (say G) as output which record the path through which salesman travel.
2) Algo begin by sorting edges in graph from least to largest distance
3) First edge is selected with least distance and one of the vertex say A is taken as origin.
4) Then similarly amongst other edges one is chosen, B as origin with least dist.
5) Continue the process with further nodes making sure that there are no cycles in output and after travelling all nodes it reaches back to origin A.
6) The shortest path with least cost is identified.

Scanned with OKEN Scanner



Graph Traversal

The process of visiting & exploring a graph for processing a graph.

Every ~~graph~~ tree is a graph.

DFS Depth first search

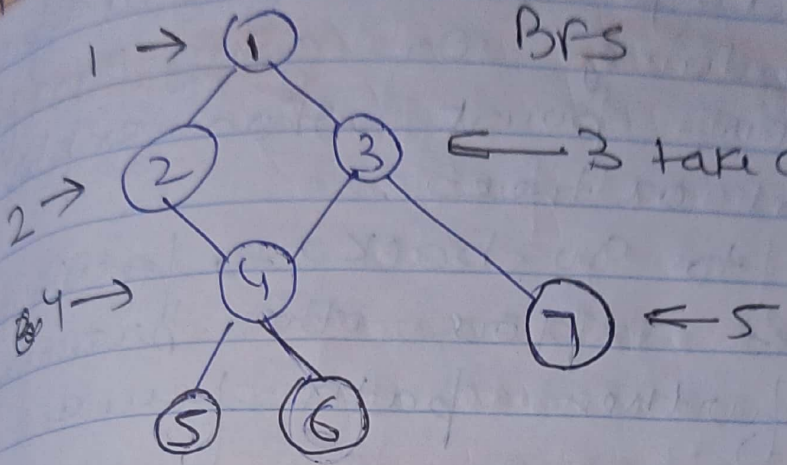
It is a graph traversal technique that explores or visits all the ~~vertex~~ vertices of graph in depth ward motion before backtracting. It starts with a chosen vertex & explores all vertex in depth & if not found backtracks.

It uses a stack. Often implemented through Recursion
Ex - detecting cycles, finding connected components, etc.

BFS Breadth first search.

Graph traversal technique that explores or visits a vertex in breadthward motion, starting from a chosen vertex. It explores vertices in layers

visiting all neighbours before going to next level. it uses a queue (FIFO)



BFS

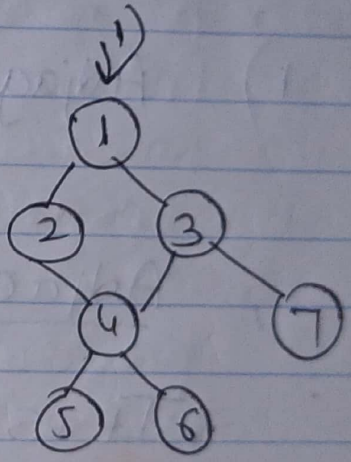
BFS (queue)

1) 2) 3) 4)

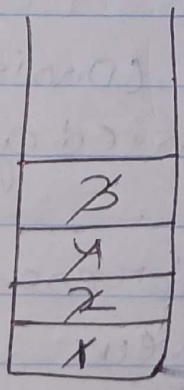
1	2	3	4	7	5	6

1 2 3 4 7 5 6.

→ BFS.



DFS.



write here ufor, when not in use put in stack.

1 2 4 5 6 3 7 → DFS.

jab 1 se 2 p gye to put 1 in stack after going to 5 niche koi option nhi h to back track to 4 again then go to 6

Back Tracking

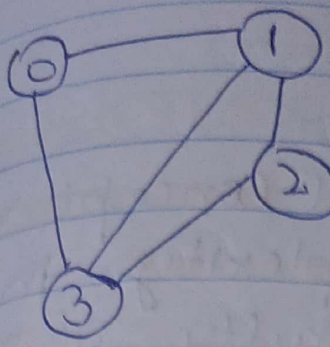
Backtracking in DFS refers to a process of undoing or going back from a decision point when exploring a graph or tree.
It Allows to go back to previous node & explore other branches if no further path found

Representation of Graph in Compuer

1) Adjagcency Matrix
2) Adjacency List.

Adjacency Matrix

It is a matrix consisting of 1 & 0
1 for having an edge & 0 for no direct edge.
It is a way to represent connection or relationship between vertices in graph
It is a nxn matrix
Each Row & colum corresponds to an edge in Matrix.
Ex - If edge from 2-3 exist mark



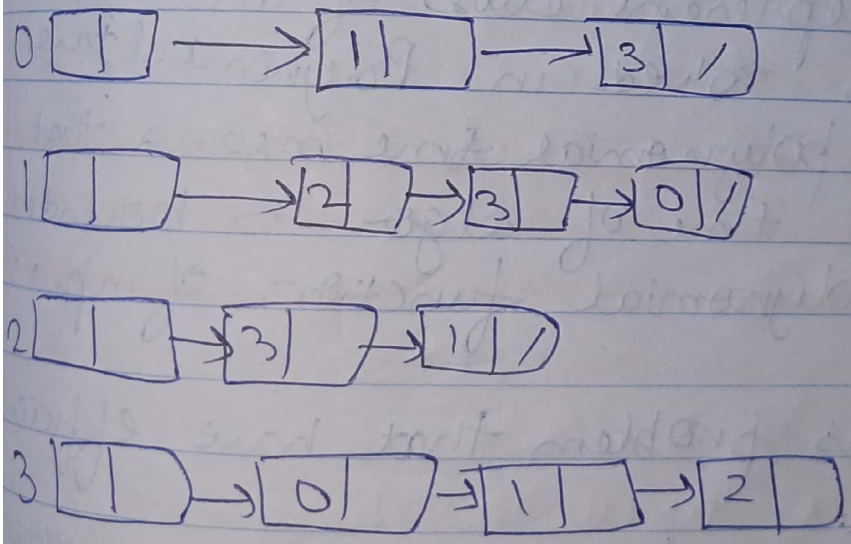
	0	1	2	3
0	0	1	0	1
1	1	0	1	1
2	0	1	0	1
3	1	1	1	0

Adjacency list

Uses linked for for each vertex
Another way to Represent relation or connection b/w vertics in a graph.
Use linklist to store connections.

Construct one link list with relation of one vertex.

with same graph as above



NP Completeness

It is a fundamental concept in computational Complexity that deals with the difficulty of solving Computational problems.
It refers to a class of problems known as NP - complete problems which are considered to be among most challenging problems.

P (Polynomial Time)
A problem is said to be Polynomial Problem (P) if it can be solved in polynomial time using deterministic algo
i.e. $O(n^k)$ where k is constant
P represent class of decision problems solved in Polynomial time where polynomial time means that running time of algo is bounded by a polynomial function of input size

- P contains problem that have efficient solution
  Ex - linear search, Binary search, Merge sort

NP - Nondeterministic Polynomical Time
NP represents the class of decision problems

for which a solution can be verified in polynomial time finding the solution might be

This include problem which only know algo require a no. of steps which increases exponentially with size of problem.

Ex- for some ques, there is no way to find ans quick but if ans is given na can verify it in given time. This is NP. , Ex- Suldoku

NP Hard

The problem that can not be solved in polynomial time are NP hard Problem
ex- Subset Sum problem.
They are most difficult to solve in terms of computational Complexity
They may not necessarily be verifiable in polynomial time as NP-complete problems.

NP Complete class

are decision problem with class NP but has special property that it is in