UNIT-3

Knapsack problem

Greedy method - An algorithm that always takes the best immediate, or local, solution while finding an answer.

→ Greedy algorithms find the overall, or globally optimal solutions for some optimization problems, but may find less-than optimal solutions for some instances of other problems.

Some of the applications of greedy method are

1) knapsack problem
2) Job sequencing with deadlines
3) Minimum cost spanning trees
4) prims algorithm
5) single source shortest path (Dijkstra's alg.)

1) Knapsack problem

objects	1	2	3	4	5	6	7	n = 7
profits	10	5	15	7	6	18	3	m = 15
weight	2	3	5	7	1	4	1	

Here, the weight of the bag is 15

→ Fill the bag with the objects such that profit is maximized and weight should not exceed 15

Objects	1	2	3	4	5	6	7
Profits	10	5	15	7	6	18	3
weight	2	3	5	7	1	4	1
P/w	5	1.3	3	1	6	4.5	3
x	1	2/3	1	0	1	1	1

15 - 1 = 14
14 - 2 = 12
12 - 4 = 8
8 - 5 = 3
3 - 1 = 2
2 - 2 = 0

$$\Sigma x_i w_i = 1\times2 + \frac{2}{3}\times3 + 1\times5 + 0\times7 + 1\times1 + 1\times4 + 1\times1$$

$$= 15$$

$\Sigma x_i p_i = 1 \times 10 + \frac{2}{3} \times 5 + 1 \times 15 + 1 \times 6 + 1 \times 18 +$
1×3

$= 54.6$

constraint - $\Sigma x_i w_i \le m$

objective - $\max \Sigma x_i p_i$

Job sequencing with deadlines

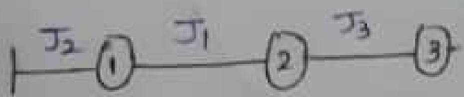
problem statement - In job sequencing problem the objective is to find a sequence of jobs which is completed within their deadlines and gives maximum profit

Job	J_1	J_2	J_3	J_4	J_5
Deadline	2	1	3	2	1
profit	60	100	20	40	20

→ The jobs have to be sorted according to their profit in decending order

After sorting

Job	J_2	J_1	J_4	J_3	J_5
Deadline	1	2	2	3	1
profit	100	60	40	20	20



Explanation of job selection

Total profit $= 100 + 60 + 20$
$= 180$

Algorithm

step1 :- Arrange all jobs in decreasing order of profit -(nlogn)

step2 :- For each job (mi) do linear search to find particular slot in array of size(n)

n = maximum deadline

m = Total jobs - n^2

Time complexity = $O(n^2)$

Algorithm for knapsack problem

→ for i = 1 to n
    calculate profit/weight

→ sort objects in decreasing order of p/w ratio

→ for i = 1 to n
    if M > 0 and $w_i \le M$
        $M = M - w_i$;
        $P = P + P_i$;
    else break
    if (M > 0)
        $P = P + P_i \left(\frac{M}{w_i}\right)$.

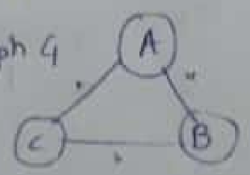
Time complexity -
O(nlogn)

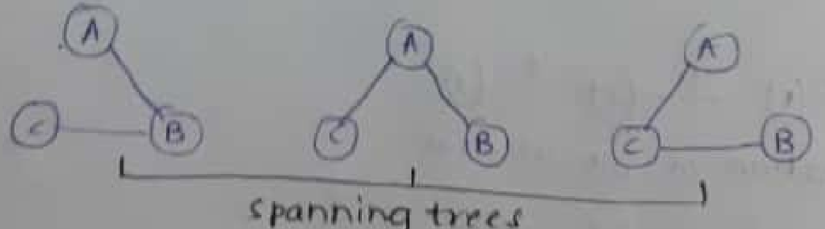
Minimum cost spanning tree

spanning tree - A spanning tree is a subset of graph G, which has all the vertices covered with minimum possible no. of edges

→ Hence, a spanning tree doesn't have cycles and it cannot be disconnected

ex: Graph G





spanning trees

properties of a spanning tree

→ A connected graph G can have more than one spanning tree

→ All possible spanning trees for graph G have same no. of edges and vertices

→ spanning tree doesn't have any cycle

→ spanning tree cannot be disconnected

Applications

- civil network planning
- computer network routing protocol
- cluster analysis

Minimum spanning tree - In a weighted graph, a minimum spanning tree is a spanning tree that has minimum weight than all other spanning trees of the same graph.

MST Algorithms

→ kruskal's Algorithm

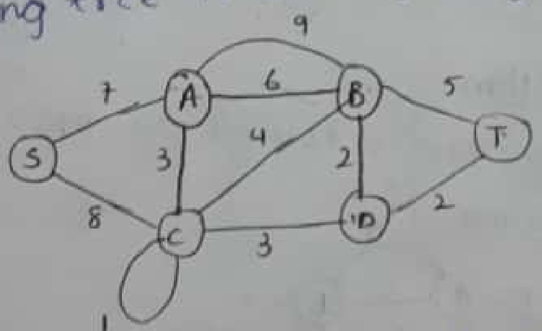
→ prims Algorithm.

→ In real world the weight may be measured as distance, traffic load etc.

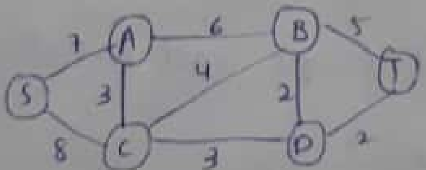
prims algorithm

prim's algorithm is to find minimum cost spanning tree it uses greedy method

~~step~~

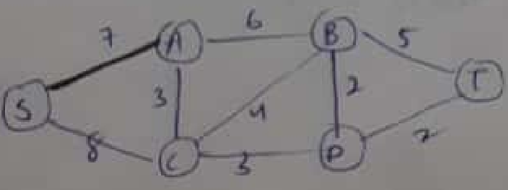


step1 - Remove all loops and parallel edges

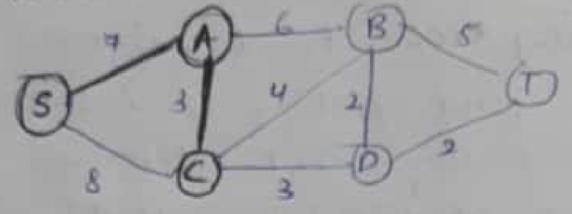


→ Incase of parallel edges keep one has least cost and remove other

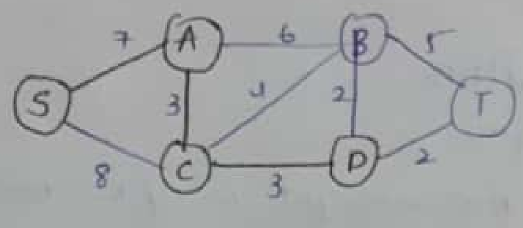
step2 - choose any one of the node as root node

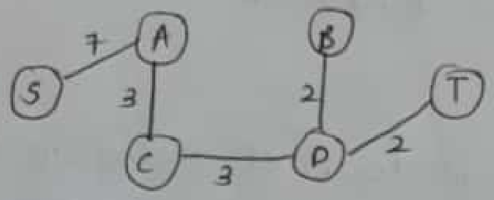


step3 - check outgoing edges and select one with least cost and continue the process till spanning tree contains all the vertices



explain each step in detail

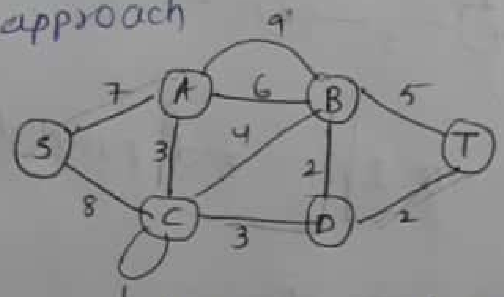




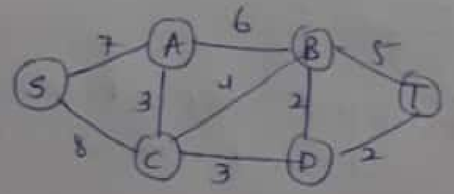
MST

kruskal's algorithm

→ kruskal's algorithm is to find the MCST using greedy approach



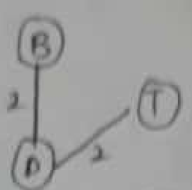
step1 - Remove all loops and parallel edges



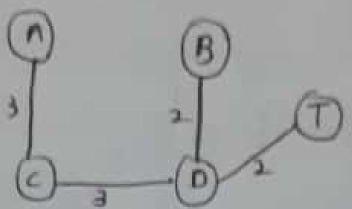
step2 - Arrange all the edges in increasing order of their weight

B,D	D,T	C,D	A,C	B,C	B,T	A,B	A,S	C,S
2	2	3	3	4	5	6	7	8

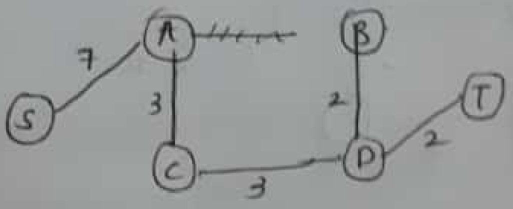
step 3- Add the edges which has least weight



continue adding edges acc to their weights in increasing order such that a spanning tree is formed



→ we avoid the edges that cost 4,5,6 because it makes a circuit in the graph. It violates the rules of spanning tree



single source shortest path problem (or)
Dijkstra's algorithm

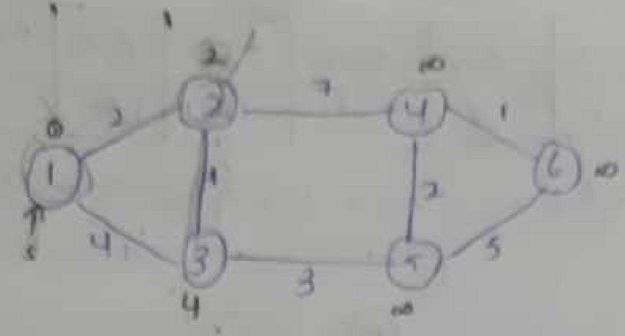
→ Dijkstra's algorithm is used to find minimum cost spanning tree

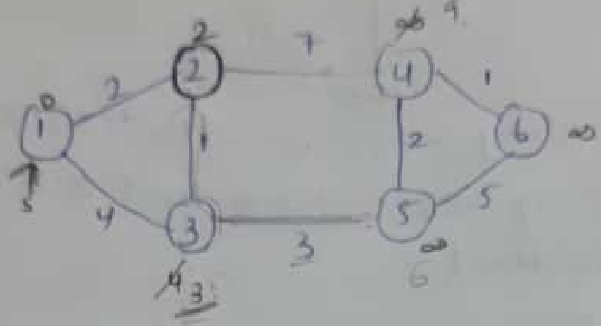
→ This algorithm updates the values of the node by the following rules

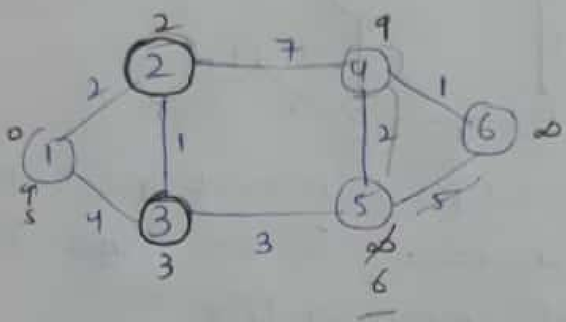
if $(d[u] + c(u,v) \le d[v])$

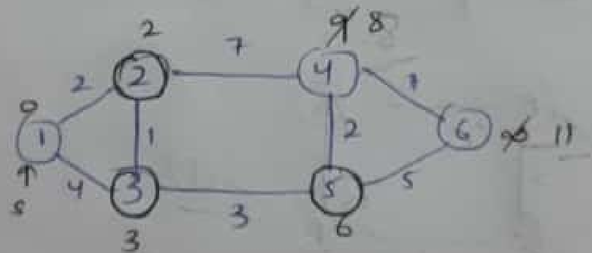
$$d[v] = d[u] + c(u,v)$$

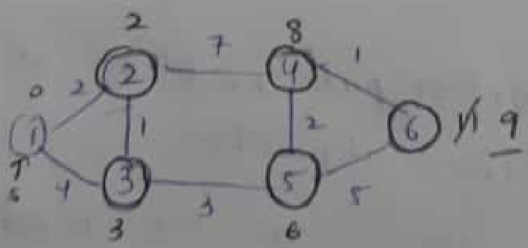
This process is called as relaxation







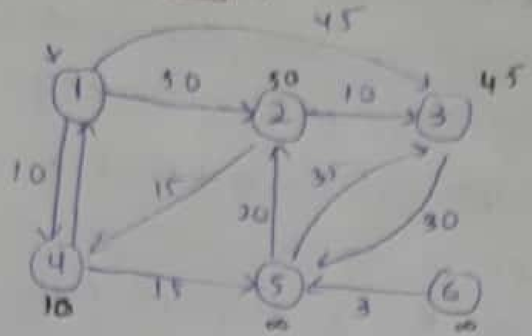




vertices	distances d[v]
2	2
3	3
4	8
5	6
6	9

→ Time complexity
worst case - $O(n^2)$

Using directed graph



selected vertex	distance 2	3	4	5	6
4	50	45	10	∞	∞
5	50	45	10	25	∞
2	45	45	10	25	∞
3	45	45	10	25	∞
6	45	45	10	25	∞

Drawback - It does not work for the cycles with negative weights
→ consume much time.

Dynamic programming

Dynamic programming is both a mathematical optimization method and a computer programming method. The method was developed by Richard bellman in the 1950's and found applications in numerous fields.

Some of the applications of dynamic programming is → Multistage graph
→ 0/1 knapsack problem
→ Travelling sales person problem
→ Reliability design
→ optimal binary search tree

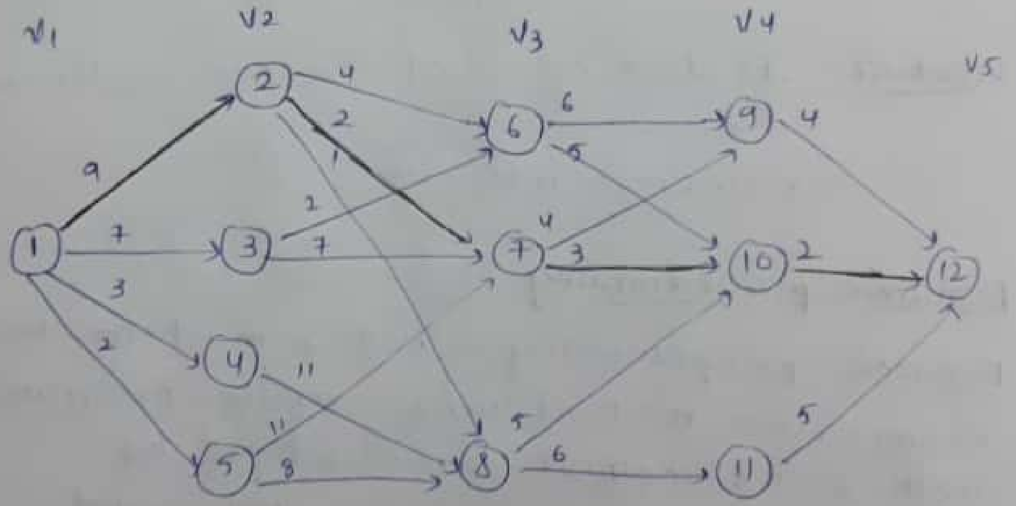
Multistage graph

Multistage problem is an optimization problem done using dynamic programming

→ Multistage graph G = (V, E, W) is a weighted directed graph in which vertices are partitioned into $k \geq 2$ disjoint sub sets $V = \{V_1, V_2 \ldots V_k\}$

→ The algorithm operates in the backward direction i.e, it starts from end (last vertex) and proceeds in backward direction

→ The goal of multistage graph problem is to find minimum cost path from source to destination.



V	1	2	3	4	5	6	7	8	9	10	11	12
cost	16	7	9	18	15	7	5	7	4	2	5	0
d	2 (or) 3	7	6	8	8	10	10	10	12	12	12	12

cost (5, 12) = 0

cost (4, 9) = 4

cost (4, 10) = 2

cost (4, 11) = 5

$cost(3,6) = \min\{c(6,9) + cost(4,9),$
$c(6,10) + cost(4,10)\}$

$\min\{6+4, 5+2\}$

$\min\{10, 7\}$

$cost(3,6) = 7$

$cost(3,7) = \min\{c(7,9) + cost(4,9),$
$c(7,10) + cost(4,10)\}$

$= \min\{4+4, 3+2\}$

$\min\{8, 5\}$

$cost(3,7) = 5$

$cost(3,8) = \min\{c(8,10) + cost(4,10)\}$
$c(8,11) + cost(4,11)\}$

$\min\{5+2, 6+5\}$

$\min\{7, 11\}$

$cost(3,8) = 7$

$cost(2,2) = \min\{c(2,6) + cost(3,6),$
$c(2,7) + cost(3,7),$
$c(2,8) + cost(3,8)\}$

$= \min\{4+7, 2+5, 1+7\}$

$= 7$

$cost(2,2) = 7$

$cost(2,3) = \min\{c(3,6) + cost(3,6),$
$c(3,7) + cost(3,7)\}$

$= \min\{2+7, 7+5\}$

$= 9$

$cost(2,3) = 9$

min(

cost(2,4) = min{c(4,8) + cost(3,8)}

$= 11 + 7$

$= 18$

cost(2,5) = min{c(5,7) + cost(3,7)}
c(5,8) + cost(3,8)}

$= \min\{11 + 5, 8 + 7\}$

$\min\{16, 15\}$

$= 15$

cost(2,5) = 15

cost(1,1) = min{c(1,2) + cost(2,2),
c(1,3) + cost(2,3),
c(1,4) + cost(2,4),
c(1,5) + cost(2,5)}

$\min\{9+7, 7+9, 3+18, 2+15\}$

$\min\{16, 16, 21, 17\}$

$= 16$

cost(1,1) = 16

Formula - $\underset{\text{stage} \downarrow \ \downarrow \text{vertex no}}{\text{cost}(i,j)} = \min\{c(i,l) + \text{cost}(i+1,l)$

$<j,l> \in E$

$l \in V_{i+1}$

→ solving by sequence of decisions by data taken

d(1,1) = 2 path 1-2-7-10

d(2,2) = 7

d(3,7) = 10

d(4,10) = 12

0/1 knapsack problem

→ In the dynamic programming we have n items each with an associated weight and value.

→ The objective is to fill the knapsack with items such that we have a maximum profit without exceeding the weight limit of knapsack

→ since it is 0/1 knapsack we cannot break an item and fill the knapsack.

constraint - $\sum w_i x_i \leq m$

objective - $\max \sum p_i x_i$

$m = 8$ $\quad p = \{1, 2, 5, 6\}$

$n = 4$ $\quad w = \{2, 3, 4, 5\}$

P	w
1	2
2	3
5	4
6	5

V	0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0	0
1	0	0	1	1	1	1	1	1	1
2	0	0	1	2	2	3	3	3	3
3	0	0	1	2	5	5	6	7	7
4	0	0	1	2	5	6	6	7	8

$v[i, w] = \max\{v[i-1, w], v[i-1, w-w[i]] + p[i]\}$

$v[4,1] = \max\{v[3,1], v[3, 1-5] + 6\}$

$v[4,5] = \max\{v[3,5], v[3, 5-5] + 6\}$

$= \max\{5, 0+6\}$

$= 6$

$v[4,6] = \max\{v[3,6], v[3, 6-5] + 6\}$

$\max\{6, 6\} = 6$

$v[4,7] = \max[v[3,7], v[3, 7-5] + 6\} = 7$

$v[4,8] = \max\{v[3,8], v[3, 8-5] + 6\} = 8$

x_1	x_2	x_3	x_4
0	1	0	1

→ Max profit - 8 [Not present in other rows 3, 2, 1]

8 - 6 (4^{th} row profit) = 2

→ 2 present in 3^{rd} rows as well as 2^{nd} so, profit 2 is not due to 3^{rd} row so don't include x_3, include x_2 if 2 is not present in 1^{st} row.

2 - 2 = 0

→ 0 is present in row 2 as well as row 1 so it is due to row 1 so don't include x_1

sets method

$P = \{1, 2, 5, 6\}$ $m = 8$, $n = 4$

$w = \{2, 3, 4, 5\}$

(P, w)

$S^0 = \{(0,0)\}$

$S^0_1 = \{(1,2)\}$

$S^1 = \{(0,0)\ (1,2)\}$

$S^1_1 = \{(2,3)\ (3,5)\}$

$S^2 = \{(0,0)\ (1,2)\ (2,3)\ (3,5)\}$

$S^2_1 = \{(5,4)\ (6,6)\ (7,7)\ (8,9)\}$

$S^3 = \{(0,0)\ (1,2)\ (2,3)\ \cancel{(3,5)}\ (5,4)\ (6,6)\ (7,7)\ \cancel{(8,9)}\}$

Acc to dominants rule P↑ w↑

At (3,5) (5,4) P↑ w↓ so cancel the one with less profit

$S^3_1 = \{(6,5)\ (7,7)\ (8,8)\ (11,9)\ \cancel{(12,11)}\ \cancel{(13,12)}\}$

$S^4 = \{(0,0)\ (1,2)\ (2,3)\ (5,4)\ (6,6)\ (7,7)\ (6,5)\ (8,8)\}$

$(8,8) \in S^4$

but $(8,8) \notin S^3$, $\therefore x_4 = 1$

$(8-6, 8-5) = (2,3)$

→ $(2,3) \in S^3$ $\therefore x_3 = 0$

and $(2,3) \in S^2$

→ $(2,3) \in S^2$

but $(2,3) \notin S^1$ $\therefore x_2 = 1$

$(2-2, 3-3) = (0,0)$

→ $(0,0) \in S^1$ &

$(0,0) \in S^0$ $\therefore x_1 = 0$

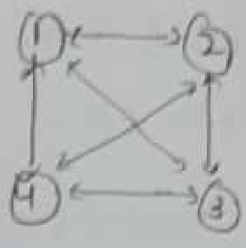
Travelling sales person problem

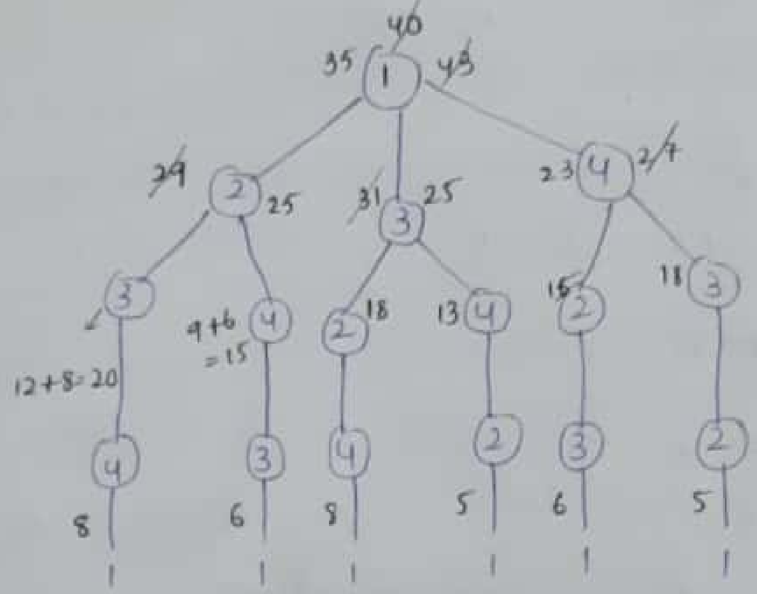
→ Travelling sales person problem is based on real life scenario, whereas a salesman from a company has to start from his own city and visit all the assigned cities exactly once and return to his home till the end of the day

i.e., we are given a set of cities and distance between them, the goal is to find the shortest path that visits every city once exactly and returns to the starting point

→ It can be done using brute-force method and dynamic programming

$$
\begin{array}{c|cccc}
 & 1 & 2 & 3 & 4 \\
\hline
1 & \multicolumn{4}{c}{} \\
2 & \multicolumn{4}{c}{} \\
3 & \multicolumn{4}{c}{} \\
4 & \multicolumn{4}{c}{}
\end{array}
\quad
\begin{bmatrix} 0 & 10 & 15 & 20 \\ 5 & 0 & 9 & 10 \\ 6 & 13 & 0 & 12 \\ 8 & 8 & 9 & 0 \end{bmatrix}
$$





→ Brute-Force approach

path 1 – 2 – 4 – 3 – 1

Dynamic programming

$$g(1,\{2,3,4\}) = \min_{k \in \{2,3,4\}} \{ c_{1k} + g(k, \{2,3,4\} - \{k\}) \}$$

$$g(i,S) = \min \{ c_{ik} + g(k, S - \{k\}) \}$$

c_{i1}

$g(2,\phi) = 5$

$g(3,\phi) = 6$

$g(4,\phi) = 8$

$g(2,\{3\}) = 15$

$g(2,\{4\}) = 18$

$g(3,\{2\}) = 18$

$g(3,\{4\}) = 20$

$g(4,\{2\}) = 13$

$g(4,\{3\}) = 15$

$g(2,\{3,4\}) = 25$

$g(3,\{2,4\}) = 25$

$g(4,\{2,3\}) = 23$

$$g(1,\{2,3,4\}) = \min \{ c_{12} + g(2,\{3,4\}),\; c_{13} + g(3,\{2,4\}),\; c_{14} + g(4,\{2,3\}) \} = 35$$

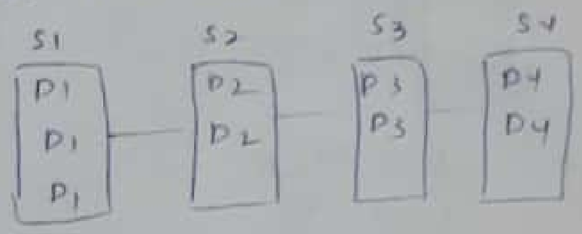
Reliability design

In reliability the problem is to design a system that is composed of several devices connected in series

If we imagine r_1 is the reliability of device Then reliability of function can be given by Πr_1

If there are n devices reliability is Πr_i

To design a system with good condition and high reliability we can take set of same copies and connect them in parallel



$r_1 = 0.9$

$1 - r_1 = 1 - 0.9 = 0.1$

$(1-r_1)^3 = 0.001$ (not working)

$1-(1-r_1)^3 = (0.999)$ (working)

$\Sigma c_i = 30 + 15 + 20 = 65$

D_i	c_i	r_i	u_i
D1	30	0.9	1
D2	15	0.8	3
D3	20	0.5	3

$c = 105$

remaining

$c - \Sigma c_i = 105 - 65 = 40$

$u_i \quad \left\lfloor \frac{c - \Sigma c_i}{c_i} \right\rfloor + 1$ (upper bound)

For device 1 - $\frac{40}{30} + 1$
$= 2$

For device 2 - $\frac{40}{15} + 1 = 3$

For device 3 - $\frac{40}{20} + 1 = 3$

Scanned with CamScanner

(R, C)

$S^0 = \{(1,0)\}$

consider D1

device copy

$S_1^1 = \{\boxed{(0.9, 30)}\}$ $\quad 1-(1-r_1)^2$

$S_2^1 = \{\boxed{(0.99, 60)}\}$ $\quad 1-(1-0.9)^2$

$1-(0.1)^2$

$1-0.01$

$= 0.99$

$S^1 = \{(0.9, 30), \boxed{(0.99, 60)}\}$

consider D2

$S_1^2 = \{(0.72, 45), (0.792, 75)\}$

$S_2^2 = \{\boxed{(0.864, 60)}, (0.9504, 90)\}$

$S_3^2 = \{(0.8928, 75), (\text{—}, 105)\}$

$1-(1-r_2)^2$

$1-(1-0.8)^2$

$1-(0.2)^2$

$1-0.04 = 0.96$

$c = 30$

$1-(1-r_2)^3$

$1-(1-0.8)^3$

$1-(0.2)^3 = 1-0.008$

$= 0.992$

$c = 45$

$S^2 = \{(0.72, 45), (0.792, 75), \boxed{(0.864, 60)}, (0.8928, 75)\}$ $\quad \because$ Dominants rule

$1-(1-r_3)^2$

$= 1-(1-0.5)^2$

$= 0.75$

$c = 40$

$1-(1-r_3)^3$

$= 0.875$

$c = 60$

$S_1^3 = \{(0.36, 65), (0.432, 80), (0.4464, 95)\}$

$S_2^3 = \{(0.54, 85), \boxed{(0.648, 100)}\}$

not possible cost exceeds

$S_3^3 = \{(0.63, 105), (\text{—}, 120), (\text{—}, 135)\}$

$S_3 = \{(0.36, 55), (0.432, 80), (0.4464, 95), (0.54, 85), (0.648, 100), (0.63, 105)\}$

$\because$ Dominants rule

Maximum reliability of system is

(0.648, 100)

R C

D1 - 2 copies

D2 - 2 copies

D3 - 2 copies

optimal binary search tree

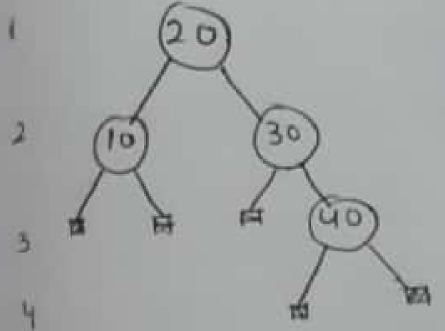
In computer science, an optimal binary search tree, sometimes called a weight-balanced binary tree

→ It is a binary search tree which provides the smallest possible search time for given sequence of accesses

→ optimal BST's are generally divided into two types: static and dynamic

BST –

keys		10		20		30		40	
P_i		.1		.2		.1		.2	
q_i	.1		.05		.15		.05		.05
	q_0		q_1		q_2		q_3		q_4



$1\times0.2 + 2\times0.1 + 2\times0.1 + 3\times0.2$
$+ 2\times0.1 + 2\times0.05 + 2\times0.15 +$
$3\times0.05 + 3\times0.05$
$= 2.1$

$$cost[0,n] = \sum_{1\le i\le n} P_i * level(a_i) + \sum_{0\le i\le n} q_i * (level(E_i)-1)$$

OBST

$$c[i,j] = \min_{i<k\le j} \{ c[i,k-1] + c[k,j] \} + w[i,j]$$

$0<k<3$

$j-i=0$ $c[0,0]$ $c[1,1]$ $c[2,2]$ $c[3,3]$
$j-i=1$ $c[0,1]$ $c[1,2]$ $c[2,3]$
$j-i=2$ $c[0,2]$ $c[1,3]$
$j-i=3$ $c[0,3]$

ex: $w[0,2] = q_0 + P_1 + q_1 + P_2 + q_2$

$$w[i,j] = w[i,j-1] + P_j + q_j$$

$w[0,3] = q_0 + P_1 + q_1 + P_2 + q_2 + P_3 + q_3$

$w[0,3] - w[0,2] + P_3 + q_3$

0 1 2 3 4

keys = { 10, 20, 30, 40 }

p_i = { 3, 3, 1, 1 }

q_i = { 2, 3, 1, 1, 1 }

$w_{00} = 2$ $c_{00} = 0$ $r_{00} = 0$	$w_{11} = 3$ $c_{11} = 0$ $r_{11} = 0$	$w_{22} = 1$ $c_{22} = 0$ $r_{22} = 0$	$w_{33} = 1$ $c_{33} = 0$ $r_{33} = 0$	$w_{44} = 1$ $c_{44} = 0$ $r_{44} = 0$
$w_{01} = 8$ $c_{01} = 8$ $r_{01} = 1$	$w_{12} = 7$ $c_{12} = 7$ $r_{12} = 2$	$w_{23} = 3$ $c_{23} = 3$ $r_{23} = 3$	$w_{34} = 3$ $c_{34} = 3$ $r_{34} = 4$	
$w_{02} = 12$ $c_{02} = 19$ $r_{02} = 1$	$w_{13} = 9$ $c_{13} = 12$ $r_{13} = 2$	$w_{24} = 5$ $c_{24} = 8$ $r_{24} = 3$		
$w_{03} = 14$ $c_{03} = 25$ $r_{03} = 2$	$w_{14} = 11$ $c_{14} = 19$ $r_{14} = 2$			
$w_{04} = 16$ $c_{04} = 32$ $r_{04} = 2$				

$w[i,j] = w[i,j-1] + p_j + q_j$

$w[0,0] = w[0,-1]$ – not found same as q_0

$c[i,j] = \min\limits_{i<k\le j} \sum [c[i,k-1] + c[k,j]] + w[i,j]$

$c[0,0] = \min\limits_{0<k=0} \sum$ no value so – 0

$w[0,1] = w[0,0] + p_1 + q_1 = 2 + 6 = 8$

$w[1,2] = w[1,1] + p_2 + q_2 = 3 + 3 + 1 = 7$

$w[2,3] = w[2,2] + p_3 + q_3 = 1 + 1 + 1 = 3$

$w[3,4] = w[3,3] + p_4 + q_4 = 1 + 1 + 1 = 3$

$c[0,1] = \min\limits_{0<k\le 1} \left[c[0,0],\ c[1,1] \right] + w[0,1]$

$= 0 + 8$

simlly $c[1,2] = 7$ $\quad c[2,3] = 3 \quad c[3,4] = 3$

$w[0,2] = w[0,1] + p_2 + q_2$

$= 8 + 3 + 1$

slly $w[1,3] = 9 \qquad w[2,4] = 5$

$c[0,2] = \min\limits_{0<k\le 2} \{ c[0,0] + c[1,2],\ c[0,1] + c[2,2] \} + w[0,2]$

$\underline{12}$

$\{ \underline{0+7},\ 8+0 \} + 12$

$= 19$

$c[1,3] = \min\limits_{1<k\le 3} \{ c[1,1] + c[2,3],\ c[1,2] + c[3,3] \} + w[1,3]$

$\underline{2,3}$

$\{ \underline{0+3},\ 7+0 \} + 9$

$= 12$

$c[2,4] = \min \{ c[2,2] + c[3,4],\ c[2,3] + c[4,4] \} + w[2,4]$

$\underline{3,4}$

$\{ 0+3,\ 3+0 \} + 5$

$= 8$

$c[0,3] = \{ c[0,1] + c[2,3],\ c[0,2] + c[3,3] \} + w[0,3]$

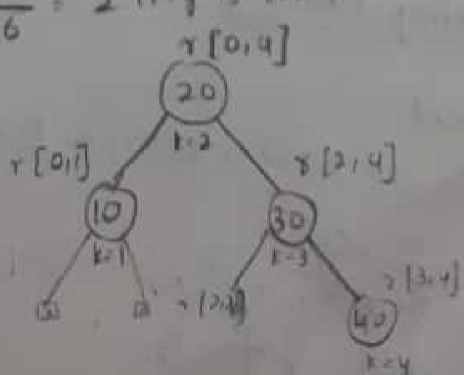
$2,1$

$\{ 8+3,\ 19+0 \} + 14$

$= 25$

creating OBST

$\text{cost} = \dfrac{32}{16} = 2$ (key is root)



UNIT-4

Backtracking

→ Backtracking strategy is used to solve the problems

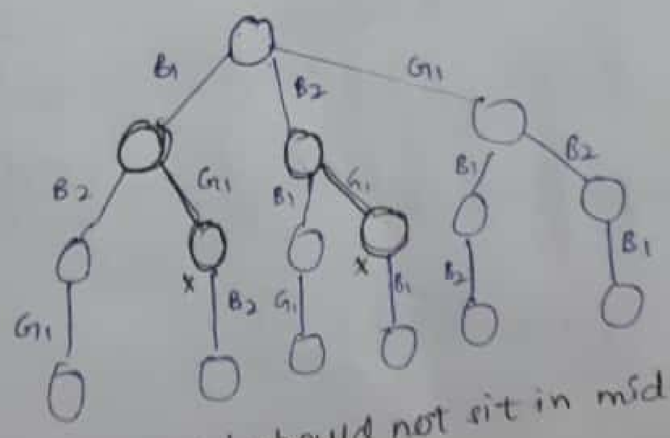
→ This strategy uses brute force approach
Brute-force approach says that for any given problem you should try out all the possible solutions and pick up desired solutions.

→ Backtracking is not an optimization problem

→ The solution is represented in the form of state space tree.

→ Backtracking is used when there are multiple solutions and we want to approach all the solutions.

state space tree B1 B2 G1 sitting arrangements



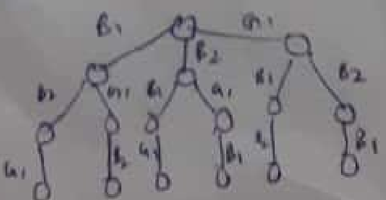
constraint - Girl should not sit in midle

killed the node

<u>Bounding function</u>

Branch and Bound

→ It is also same as backtracking but it follows BFS



N Queens problem

This problem is to find an arrangement of N queens on a chess board, such that no queen can attack any other queens on the board

The chess queens can attack in any direction as horizontal, vertical, and diagnal way.

Conditions

→ Two queens should not be in the same row
→ Two queens should not be in the same column
→ " " " " " same diagnol

No.of nodes possible

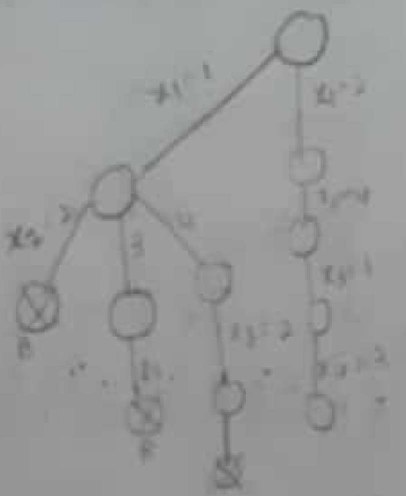
$$1 + \sum_{i=0}^{3} \left[\prod_{j=0}^{i} (N-i) \right]$$

	1	2	3	4
1		Q1		
2				Q2
3	Q3			
4			Q4	

Bounding function =
Not in same row
column
diagnol

solution 2,4,1,3

other solution
3,1,4,2



* Algorithm

Algorithm place(k, i)

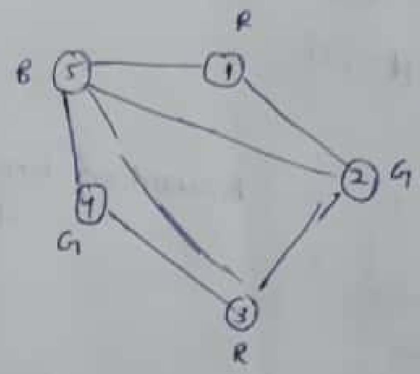
{ for j = 1 to k-1 do

if (x[j] =

Graph colouring

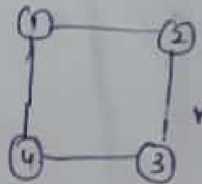
Graph coloring is the procedure of assignment of colors to each vertex of a graph G such that no adjacent vertices get same colour. The objective is to minimize the number of colors while coloring a graph.

R, G, B



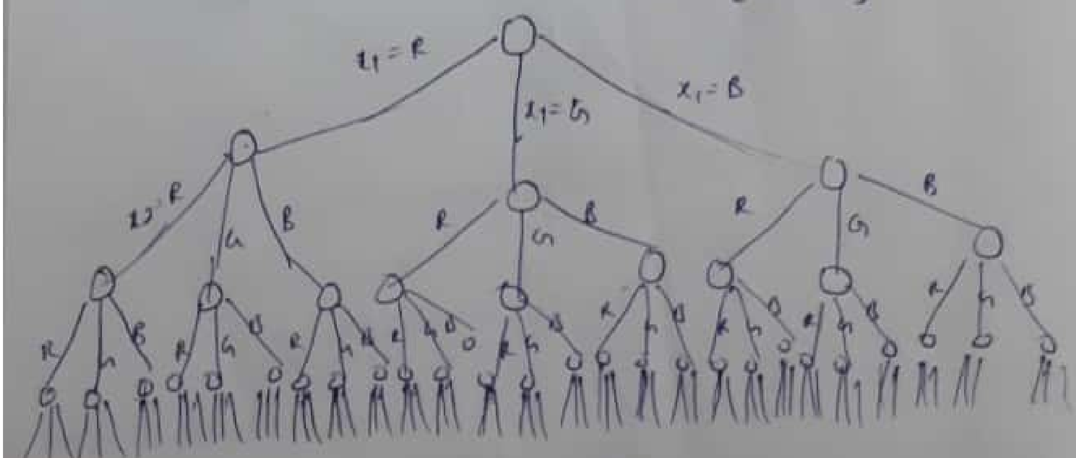
1	2	3	4	5
R	G	R	G	B
G	R	G	R	B

For the simple graph

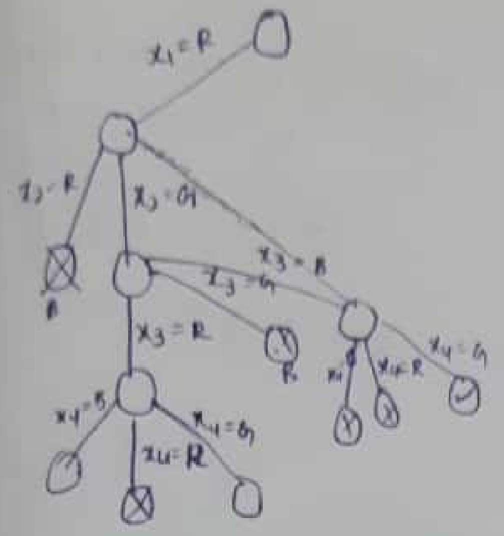


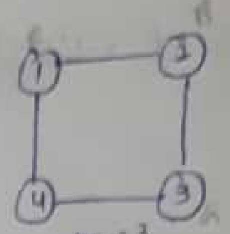
m = 3

{R, G, B}



Time complexity - c^{n+1}

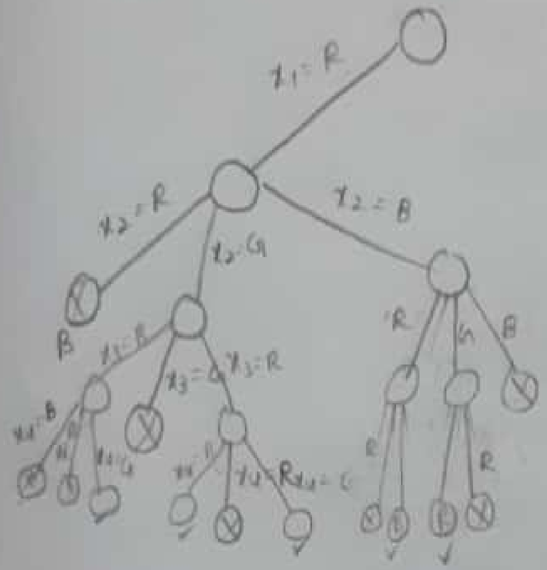




$m = 3$

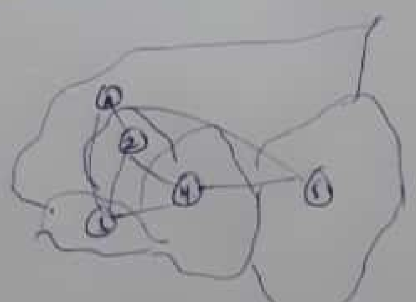
$\{R, G, B\}$

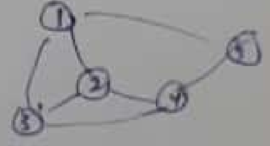
R, G, R, G
R, G, R, B
R, G, B, G



R, G, R, G
R, G, R, B
R, G, B, G
R, B, R, B
R, B, G, B

This problem is majorly used for printing the clours in the graph. To avoid keeping in the machine for more no. of times we use this problem





m-coloring decision problem → can be colored or not
m-coloring optimization problem → How may colours are req

Sum of subsets

$w[1:6] = \{5, 10, 12, 13, 15, 18\}$

$n = 6 \quad m = 30$

x

1	1	0	0	1	[illegible]

$$\sum_{i=1}^{k} w_i x_i + w_{k+1} \le m \quad - \text{ Bounding function}$$

$$\sum_{i=1}^{k} w_i x_i + \sum_{i=k+1}^{n} w_i > m$$