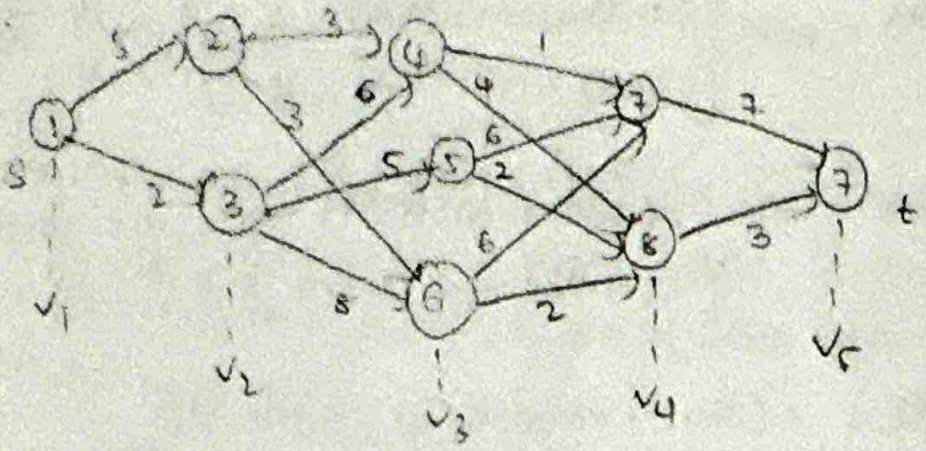


All pairs shortest path (Floyed marshall)

$$A^k[i,j] = \begin{cases} c[i,j] & k=0 \\ \min\{A^{k-1}[i,j] \\ \quad A^{k-1}[i,k] + A^{k-1}[k,j] & k \geq 1 \end{cases}$$

Procedure:

S-1: Optimal substructure of a shortest path- shortest path algorithms typically rely on the property that a shortest path between two vertices contains other shortest path within it (principle of optimality)

2. A recursive soln wher $c[i,j]$ is the cost matrix of the given graph

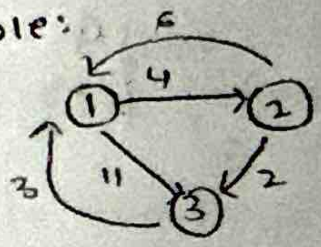
3. computing the cost adjacency matrices A^k where $[k=1,2,\ldots n]$ where n = no. of vertices

4. Finally An matrix gives the shortest distance from every vertex 'i' to every other matrix 'j'

Algorithm:

```
Algorithm Allpairs(cost[i,j])
{
for i:=1 to n do
{ for j:=1 to n do
    A[i,j] = cost[i,j];
}
for i:=1 to n do
{ for j:=1 to n do
  {
    for k:=1 to n do
        A^k[i,j] = min{ A^(k-1)[i,j], A^(k-1)[i,k] + A^(k-1)[k,j] };
  }
}
return A^k;
}
```

Example:



k → max on
n → no. of vertices.

$$A^0 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{ccc} 1 & 2 & 3 \end{array} \\ \begin{bmatrix} 0 & 4 & 11 \\ 6 & 0 & 2 \\ 3 & \infty & 0 \end{bmatrix} \end{array}$$

no, self loops
1-1, 2-2, 3-3 = 0

if k=1 : assume vertex 1 is intermediate vertex

$$A^1[2,3] = \min\begin{cases} A^{1-1}[2,3] \\ A^{1-1}[2,1] + A^{1-1}[1,3] \end{cases}$$

$$= \min\begin{cases} A^0[2,3] \\ A^0[2,1] + A^0[1,3] \end{cases} = \min\begin{cases} 2 \\ 6+11 \end{cases} = 2$$

$$A^1[3,2] = \min\begin{cases} A^{1-1}[3,2] \\ A^{1-1}[3,1] + A^{1-1}[1,2] \end{cases} = \min\begin{cases} \infty \\ 3+4 \end{cases} = 7$$

$$A^1 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{ccc} 1 & 2 & 3 \end{array} \\ \begin{bmatrix} 0 & 4 & 11 \\ 6 & 0 & 2 \\ 3 & 7 & 0 \end{bmatrix} \end{array}$$

let k=3 consider vertex 3 as intermediate node

$A^3[1,3]$ and $A^3[3,1]$

$$A^3[1,3] = \min \begin{cases} A^{3-1}[1,3] \\ A^{3-1}[1,3] + A^{3-1}[3,3] \end{cases} = \min \begin{cases} 11 \\ 4+1 = 6 \end{cases}$$

$$A^3[3,1] = \min \begin{cases} A^{3-1}[3,1] \\ A^{3-1}[3,3] + A^{3-1}[3,1] \end{cases} = \min \begin{cases} 3 \\ 1+6 = 3 \end{cases}$$

$$A^3 = \begin{bmatrix} 0 & 11 & 6 \\ 6 & 0 & 2 \\ 3 & 1 & 0 \end{bmatrix}$$

let k=4 consider vertex 4 as intermediate node

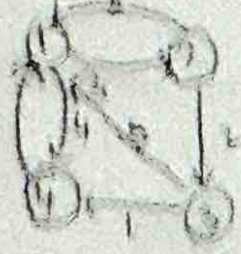
$$A^4[1,3] = \min \begin{cases} A^{4-1}[1,3] \\ A^{4-1}[1,4] + A^{4-1}[4,3] \end{cases} = \min \begin{cases} 6 \\ 3+1 = 4 \end{cases}$$

$$A^4[3,1] = \min \begin{cases} A^{4-1}[3,1] \\ A^{4-1}[3,4] + A^{4-1}[4,1] \end{cases} = \min \begin{cases} 3 \\ 2+3 = 5 \end{cases}$$

$$A^4 = \begin{bmatrix} 0 & 11 & 6 \\ 5 & 0 & 2 \\ 3 & 1 & 0 \end{bmatrix}$$

Analysis: TC = $O(n^3)$

Example-1:



$$A^0 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 6 & \infty & 2 \\ 6 & 0 & 2 & \infty \\ 5 & \infty & 0 & 1 \\ 3 & \infty & \infty & 0 \end{bmatrix} \end{array}$$

if k=1 Assume vertex 1 as intermediate node

$$A^1[3,4] = \min \begin{cases} A^{1-1}[3,4] \\ A^{1-1}[3,1] + A^{1-1}[1,4] \end{cases} = \min \begin{cases} 1 \\ 5+2 = 7 \end{cases}$$

$$A^1[4,3] = \min \begin{cases} A^{1-1}[4,3] \\ A^{1-1}[4,1] + A^{1-1}[1,3] \end{cases} = \min \begin{cases} \infty \\ 3+6 = 10 \end{cases}$$

$$A^1[2,3] = \min \begin{cases} A^{1-1}[2,3] \\ A^{1-1}[2,1] + A^{1-1}[1,3] \end{cases} = \min \begin{cases} 2 \\ 6+6 = 12 \end{cases}$$

$A^1[4,2] = \min \begin{cases} A^{1-1}[4,2] \\ A^{1-1}[4,1] + A^{1-1}[1,2] \end{cases} = \min \begin{cases} \infty \\ 7+\infty \end{cases} = \infty$

$A^1[2,3] = \min \begin{cases} A^{1-1}[2,3] \\ A^{1-1}[2,1] + A^{1-1}[1,3] \end{cases} = \min \begin{cases} 2 \\ 1+\infty \end{cases} = 2$

$A^1[3,2] = \min \begin{cases} A^{1-1}[3,2] \\ A^{1-1}[3,1] + A^{1-1}[1,2] \end{cases} = \min \begin{cases} \infty \\ 5+3 \end{cases} = 8$

$$A^1 = \begin{bmatrix} 0 & 3 & \infty & 2 \\ 1 & 0 & 2 & 10 \\ 5 & 8 & 0 & 1 \\ 7 & 10 & \infty & 0 \end{bmatrix}$$

If $k=2$: assume vertex 2 as intermediate node

$A^2[1,3] = \min \begin{cases} A^{2-1}[1,3] \\ A^{2-1}[1,2] + A^{2-1}[2,3] \end{cases} = \min \begin{cases} \infty \\ 3+2 \end{cases} = 5$

$A^2[3,1] = \min \begin{cases} A^{2-1}[3,1] \\ A^{1}[3,2] + A^{1}[2,1] \end{cases} = \min \begin{cases} 5 \\ 8+1 \end{cases} = 5$

$A^2[1,4] = \min \begin{cases} A^{1}[1,4] \\ A^{1}[1,2] + A^{1}[2,4] \end{cases} = \min \begin{cases} 2 \\ 3+10 \end{cases} = 2$

$A^2[4,1] = \min \begin{cases} A^{1}[4,1] \\ A^{1}[4,2] + A^{1}[2,1] \end{cases} = \min \begin{cases} 7 \\ 8+10 \end{cases} = 7$

$A^2[3,4] = \min \begin{cases} A^{1}[3,4] \\ A^{1}[3,2] + A^{1}[2,4] \end{cases} = \min \begin{cases} 2 \\ 8+10 \end{cases} = 2$

$A^2[4,3] = \min \begin{cases} A^{1}[4,3] \\ A^{1}[4,2] + A^{1}[2,3] \end{cases} = \min \begin{cases} \infty \\ 10+2 \end{cases} = 12$

$$A^2 = \begin{bmatrix} 0 & 3 & 5 & 2 \\ 8 & 0 & 2 & 10 \\ 5 & 8 & 0 & 1 \\ 7 & 10 & 12 & 0 \end{bmatrix}$$

If k=3: assume vertex 3 as intermediate

$A^3[1,2] = \min \begin{cases} A^2[1,2] \\ A^2[1,3]+A^2[3,2] \end{cases} = \min \begin{cases} 3 \\ 5+8 \end{cases} = 3$

$A^3[2,1] = \min \begin{cases} A^2[2,1] \\ A^2[2,3]+A^2[3,1] \end{cases} = \min \begin{cases} 8 \\ 2+5 \end{cases} = 7$

$A^3[4,1] = \min \begin{cases} A^2[4,1] \\ A^2[4,3]+A^2[3,1] \end{cases} = \min \begin{cases} 7 \\ 12+5 \end{cases} = 7$

$A^3[1,4] = \min \begin{cases} A^2[1,4] \\ A^2[1,3]+A^2[3,4] \end{cases} = \min \begin{cases} 2 \\ 5+1 \end{cases} = 2$

$A^3[2,4] = \min \begin{cases} A^2[2,4] \\ A^2[2,3]+A^2[3,4] \end{cases} = \min \begin{cases} 10 \\ 2+1 \end{cases} = 3$

$A^3[4,2] = \min \begin{cases} A^2[4,2] \\ A^2[4,3]+A^2[3,2] \end{cases} = \min \begin{cases} 10 \\ 12+8 \end{cases} = 10$

$$A^3 = \begin{bmatrix} 0 & 3 & 5 & 2 \\ 7 & 0 & 2 & 3 \\ 5 & 8 & 0 & 1 \\ 7 & 10 & 12 & 0 \end{bmatrix}$$

If k=4: assume vertex 4 as intermediate

$A^4[1,2] = \min \begin{cases} A^3[1,2] \\ A^3[1,4]+A^3[4,2] \end{cases} = \min \begin{cases} 3 \\ 2+10 \end{cases} = 3$

$A^4[2,1] = \min \begin{cases} A^3[2,1] \\ A^3[2,4]+A^3[4,1] \end{cases} = \min \begin{cases} 7 \\ 3+7 \end{cases} = 7$

$A^4[1,3] = \min \begin{cases} A^3[1,3] \\ A^3[1,4]+A^3[4,3] \end{cases} = \min \begin{cases} 5 \\ 2+12 \end{cases} = 5$

$A^4[3,1] = \min \begin{cases} A^3[3,1] \\ A^3[3,2]+A^3[4,1] \end{cases} = \min \begin{cases} 5 \\ 11+1 \end{cases} = 5$

$A^4[2,3] = \min \begin{cases} A^4[2,3] \\ A^4[2,4]+A^4[4,3] \end{cases} = \min \begin{cases} 2 \\ 3+1 \end{cases} = 2$

$A^4[3,2] = \min \begin{cases} A^4[3,2] \\ A^4[3,4]+A^4[4,2] \end{cases} = \min \begin{cases} 8 \\ 1+10 \end{cases} = 8$

$$A^4 = \begin{bmatrix} 0 & 3 & 5 & 2 \\ 7 & 0 & 2 & 3 \\ 5 & 8 & 0 & 1 \\ 7 & 10 & 12 & 0 \end{bmatrix}$$

shortest: find ∞ pairs [no path for them]

example: [1,3) → calculate for this via 2,4

[3,2] → via 1,4

[4,3]

0/1 knapsack

A thief robbing a store finds n items the ith item, give profit of P_i rupees and weights w_i whose p_i and w_i are integers. He wants to take as valuable a load as possible but he can carry at most w kgs in his knapsack for some integer w. This is called 0/1 knapsack problem because each item must either be taken or left behind.

→ the thief cannot take a fractional amount of an item or take an item more than once

procedure:

1. Generate set S^i where the set contains possible elements (P_i, w_i) that can be added to the set.
2. A recursive solution: initially, the bag contains no items so that $S^0 = \{(0,0)\}$ for $i=0$

$$S_1^i = S^{i-1} + \{(P_i, w_i)\}$$

$$S^i = S^{i-1} + S_1^{i-1} \text{ (merge)}$$

purging rule/dominance rule:

If S^i has a pair (P_i, w_i) and (P_j, w_j) and $P_i < P_j$ but $w_i \geq w_j$ then the pair (P_i, w_i) is discarded from the set S^i. This process is repeated after every generation of S^i where $i=1$ to n.

→ generating the sets, $S^1, S^2 \cdots S^n$ using the above formula.

→ after generating S^n, select the element (P_k, W_k) such that $W_k \leq$ capacity of the bag and P_k is greater than all sets if $(P_k, W_k) \in S^n$ and $(P_k, W_k) \notin S^{n-1}$ then $x_n = 1$ otherwise $x_n = 0$

→ If $x_n = 1$ find another element (P_j, W_j) such that $P_j = P_k - P_n$ and $W_j = W_k - W_n$

check if (P_j, W_j) belongs to S^{n-1} and it does not belong to S^{n-2} then $x_{n-1} = 1$ otherwise

this process repeats until we find all x_i value where $i = 1$ to n

Example: consider knapsack example where,

$n = 3$ $(w_1, w_2, w_3) = (2, 3, 4)$, $(P_1, P_2, P_3) = (1, 2, 5)$

$M = 6$; (weight of knapsack, W or M)

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

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

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

$S_1^2 = \{(0+2, 0+3), (1+2, 2+3)\} = \{(2,3), (3,5)\}$

$S_2 = \{(0,0), (1,2), (2,3), (3,5)\}$

$S_3^3 = \{(0+5, 0+4), (1+5, 2+4), (2+5, 3+4), (3+5, 5+4)\}$

$S_3^3 = \{(5,4), (6,6), (7,7), (8,9)\}$

(7,7) → overweight

$S_3 = \{(0,0), (1,2), (2,3), (3,5), (5,4), (6,6)\}$

(3,5) < (5,4) → purging rule (remove)

$S_3 = \{(0,0), (1,2), (2,3), (5,4), (6,6)\}$

after purging rule is applied final soln is S^3

→ (6,6) is (P_k, W_k) with max profit and weight $\leq$ knapsack weight.

$(6,6) \in S_3^3 \Rightarrow x_3 = 1$

$(P_k, P_3, W_k - W_3) = (6-5, 6, 1) = (1, 2)$

$(1,2) \in S_1^1 \Rightarrow x_1 = 1$

$(1-1, 2-2) = (0,0) \Rightarrow x_2 = 0$

$(x_1, x_2, x_3) = (1, 0, 1)$ with max profit of 6.

algorithm DKP(P, W, n, M)

{
$S^0 = \{(0,0)\}$;
for i := 1 to n do
{
$S_1^{i-1} = S^{i-1} + (P_i, W_i)$;
S^i = merge - purge(S^{i-1}, S_1^i);
}
(P_x, W_x) := last pair in S^{i-1}
$(P_y, W_y) := (P + P_n, W + W_n)$ where W is the largest w in S^{n-1}
// traverse back for $x_n, x_{n-1} \ldots x_1$ such that $W + W_n <= M$;
if $(P_x > P_y)$ then
$x_n = 0$;
else
$x_n = 1$;
trackback for $x_{n-1} \ldots x_1$;
}

TC = $O(2^n)$

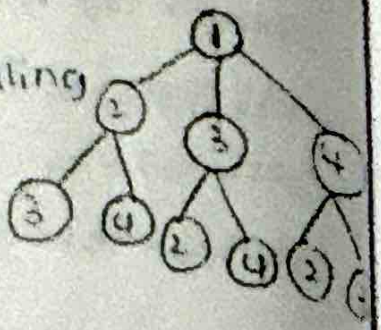
1. solve, $W = 5$, $(W_1, W_2, W_3, W_4) = (2, 1, 3, 2)$
$(P_1, P_2, P_3, P_4) = (12, 10, 20, 15)$, $n = 4$

2. solve, $n = 4$, $M = 3$
$(W_1, W_2, W_3) = (1, 2, 2)$, $(P_1, P_2, P_3) = (18, 16, 6)$

using dynamic programming

Travelling sales person problem.

There are n cities, start travelling from a point and travel all cities within mincost and end the travel at samepoint



$$g(i,s) = \begin{cases} c(i,j) & \text{if } s = \phi \\ \min\{c(i,j) + g(j, s-\{j\})\} & \text{if } s \neq \phi \end{cases}$$

Problem statement:

It is a minimization problem which is used to find the route travelled by sales man. starting from 1 vertex and visiting each vertex exactly once and returning back to starting vertex. The objective of this problem is to minimizing travelling cost of sales person.

Procedure:

1. derive the shortest path between the vertices, contains other shortest path within it

2. Recursive soln

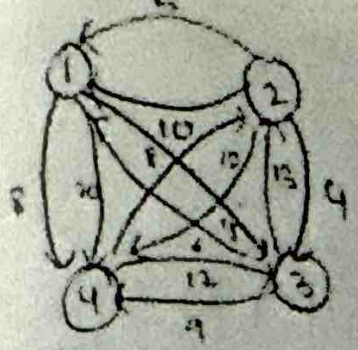
$$g(i,s) = \begin{cases} c(i,j) & \text{if } s = \phi \\ \min\{c(i,j) + g(j, s-\{j\})\} & \text{if } s \neq \phi \end{cases}$$

where $c(i,j)$ → cost matrix of given graph.

3. computing route until all vertices are added to sets

4. Finally, calculate $g(1,s)$ where set s contains all vertices other than starting vertex which gives optimal cost of travelling.

Example:

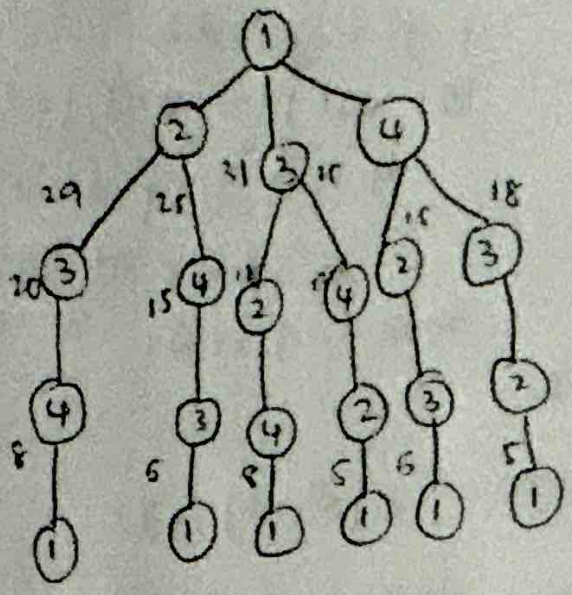


	1	2	3	4
1	0	10	15	20
2	5	0	9	10
3	6	13	0	12
4	8	8	9	0

$g(2,\phi) = C[2,1] = 5$

$g(3,\phi) = C[3,1] = 6$

$g(4,\phi) = C[4,1] = 8$



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

$\min\{c[2,3] + g(3,\phi)\}$

$= \min\{9+6\} = 15$

$g(2,\{4\}) = \min\{c[2,4] + g(4,\phi)\} = 10+8 = 18$

$g(3,\{2\}) = \min\{c[3,2] + g(2,\phi)\} = 13+5 = 18$

$g(3,\{4\}) = \min\{c[3,4] + g(4,\phi)\} = 12+8 = 20$

$g(4,\{2\}) = \min\{c(4,2) + g(2,\phi)\} = 8+5 = 13$

$g(4,\{3\}) = \min\{c(4,3) + g(3,\phi)\} = 9+6 = 15$

$g(2,\{3,4\}) = \min\begin{cases} c(2,3) + g(3,\{4\}) \\ c(2,4) + g(4,\{3\}) \end{cases} = \min\begin{cases} 9+20 \\ 10+15 \end{cases} = 25$

$g(3,\{2,4\}) = \min\begin{cases} c(3,2) + g(2,\{4\}) \\ c(3,4) + g(4,\{2\}) \end{cases} = \min\begin{cases} 13+18 \\ 12+13 \end{cases} = 25$

$g(4,\{2,3\}) = \min\begin{cases} c(4,2) + g(2,\{3\}) \\ c(4,3) + g(3,\{2\}) \end{cases} = \min\begin{cases} 8+15 \\ 9+18 \end{cases} = 23$

$g(1,\{2,3,4\}) = \min\begin{cases} c[1,2] + g(2,\{3,4\}) \\ c[1,3] + g[3,\{2,4\}] \\ c[2,4] + g(4,\{2,3\}) \end{cases} = \begin{cases} 10+25 \\ 15+25 \\ 20+23 \end{cases} = 35$

$1 \to 2 \to 4 \to 3 \to 1 \to 35$

Algorithm TSP_DP(c)

{

g(i,{}) := c(i,1);

for s := 1 to n-1 do

for all subsets of {1,2,...,n} of size s and containing i

if g(i,s) := ∞

for all j ∈ s and j ≠ 1

g(i,s) = min{c[i,j] + g(j,s-{j})}

∀ j ∈ s and i ≠ j

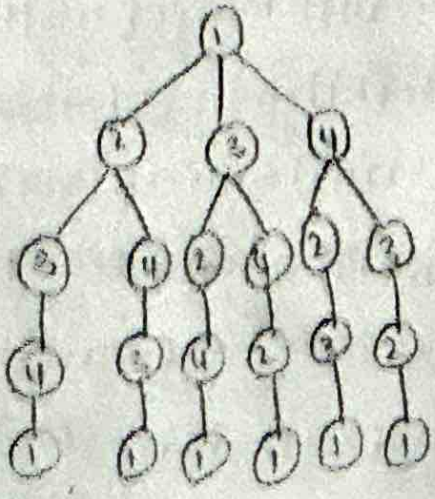
return g(1,{2,...,n});

}

Analysis: $O(2^n n^2)$ → TC

construct optimal travelling sales person tour using dynamic programming for given data

	1	2	3	4
1	0	10	9	3
2	5	0	6	2
3	9	6	0	7
4	7	8	5	0



g(2,∅) = c(2,1) = 5

g(3,∅) = c(3,1) = 9

g(4,∅) = c(4,1) = 7

g(2,{3,4}) = c(2,3) + g(3,∅)

= 6 + 9 = 15

g(2,{4,3}) = c[2,4] + g(4,∅) = 2 + 7 = 9

g(3,{2,4}) = c[3,2] + g(2,∅) = 6 + 5 = 11

g(3,{4,2}) = c[3,4] + g(4,∅) = 7 + 9 = 16

g(4,{2,3}) = c[4,2] + g(2,∅) = 8 + 5 = 13

g(4,{3,2}) = c[4,3] + g(3,∅) = 5 + 9 = 14

$g(2,\{3,4\}) = \min\begin{cases} c[2,3] + g(3,\{4\}) \\ c[2,4] + g(4,\{3\}) \end{cases} = \begin{cases} 6+10 \\ 2+10 \end{cases} = 16$

$g(3,\{4\},4) = \min\begin{cases} c[3,4] + g(4,\{\}) \\ c[3,4] + g(4,\{2\}) \end{cases} = \begin{cases} 6+9 \\ 7+8 \end{cases} = 15$

$g(4,\{2,3\}) = \min\begin{cases} c[4,2] + g(2,\{3\}) \\ c[4,3] + g(3,\{2\}) \end{cases} = \begin{cases} 8+15 \\ 5+11 \end{cases} = 16$

$g(1,\{2,3,4\}) = \min\begin{cases} c[1,2] + g(2,\{3,4\}) \\ c[1,3] + g(3,\{2,4\}) \\ c[1,4] + g(4,\{2,3\}) \end{cases}$

$= \begin{cases} 10+16 \\ 9+15 \\ 8+16 \end{cases} = \begin{cases} 26 \\ 24 \\ 19 \end{cases}$

$= 19.$

$1 \to 4 \to 3 \to 2 \to 1$

$3+5+6+5 = 19$

Binary search tree:

A BST is a binary tree either tree is empty or non empty, if it is non empty

1. All left side elements of the root node should be less than the root node.
2. All the right side elements of root node should be greater than the root node.
3. Both right and left subtrees should also satisfy BST property

Optimal Binary search Tree:

Given a sequence $K = \{k_1, k_2 \cdots k_n\}$ of n distinct keys in sorted order, i.e, $k_1 < k_2 < k_3 < \cdots < k_n$. and we wish to build a BST from these keys such that the cost of BST is minimum.

For each key k_i we have a probability that search will be successful, some searches may be for values not in set K and we also have n+1 dummy keys which are $e_0, e_1, \ldots e_n$ representing values not in K in particular, e_0 represents all values less than k_1, and e_n represents all values greater than k_n.

Dummy key e_i represents all values blw k_i and k_{i+1}

For each dummy key e_i have a probability q_i that a search will correspond to e_i

no. of possible BST can be constructed for n keys is $\frac{2n C_n}{n+1}$

Procedure:

cost of BST is calculated as

$$\sum_{i=1}^{n} p_i * level(a_i) + \sum_{i=0}^{n} q_i * (level(e_i) - 1)$$

1. structure of an optimal BST, construct an optimal soln to the problem from optimal soln to the sub-problems

Given keys k_i to k_j, one of these keys k_r so that $i \le r \le j$ will be the root of an optimal subtree containing these keys, the left sub-tree of root k_r will contain the keys from k_i to k_{r-1} and right subtree will contain the keys k_{r+1} to k_j and dummy keys of left subtree will be e_{i-1} to e_{r-1} and dummy keys of right subtree will be e_r to e_j

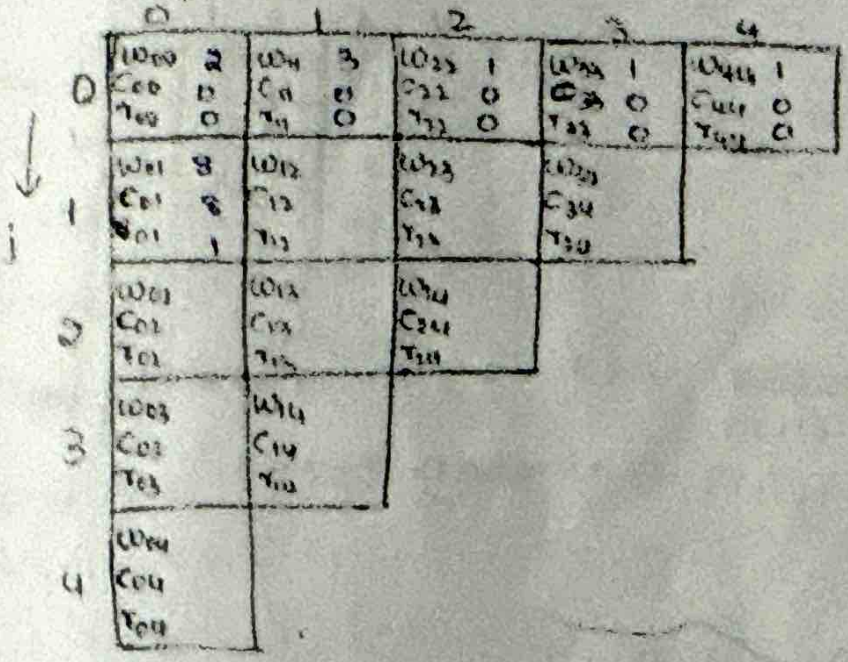
1. recursive soln

we define the cost of an optimal soln recursively in terms of optimal soln to sub problem for the optimal BST problem. we can define weight $w(i,j)$, cost $c[i,j]$, root $r(i,j)$

$$w(i,j) = q_i$$
$$c[i,i] = 0$$
$$r(i,i) = 0$$

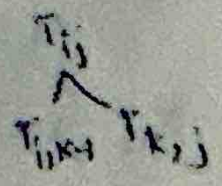
$$i = j \mid \begin{cases} w(i,j) = w(i,j-1) + p_j + q_j \\ c(i,j) = \min_{i<k\le j} \{c[i,k-1] + c[k,j]\} + w(i,j) \\ r(i,j) = k \end{cases}$$

2. computing the expected search cost of an optimal BST by using tabular & bottom up approaches.



4. construct an optimal soln

consider the 0th column 4th row specify the root that is k_1 then k^{th} element from the given list is a root node. if T_{ij} is the tree with root k, the tree is subdivided in $T_{i,k-1}$ and right subtree is T_{kj}, the process will be repeated until T_{ij} is reached, where $i=j$ so at this condition tree will become leaf node.



$w_{01} = w_{00} + p_1 + q_1$

$= 0 + 3 + 5 = 8$

$c_{01} = \{ i < k \leq j \}$

$\{ i=0, j=1 \Rightarrow k=1 \}$

$= c[0,0] + c[1,1] + w(0,1)$

$= 0 + 0 + 8 = 8$

$r_{01} = k = 1$

$w_{02} = w_{01} + p_2 + q_2$

$= 8 + 3 + 1 = 12$

$c_{02} = \{ i < k \leq j \}$

$\{ i=0, j=2 \Rightarrow k = 1, 2 \}$

$= c[0,0] + c[1,2] + w(0,2) \quad 0 + 19 = 1 + 12 = 19 \Rightarrow k = 1$

$c[0,1] + c[2,2] + w(0,2) \quad 8 + 0 + 12 = 20 \Rightarrow k = 2$

$r_{02} = k = 1$

$w_{13} = w_{11} + p_2 + q_2 = 3 + 3 + 1 = 7$

$c_{13} = \{ i < k \leq j \}$

$\{ i=1, j=3 \Rightarrow k = 2, 3 \}$

$= c[1,1] + c[2,3] + w(1,3)$

$= 0 + 0 + 7 = 7$

$r_{13} = k = 2$

$w_{23} = w_{22} + p_3 + q_3 = 1 + 1 + 1 = 3$

$c_{23} = \{ i < k \leq j \Rightarrow i = 2, j = 3 \Rightarrow k = 3 \}$

$= c[2,2] + c[3,3] + w(2,3) = 0 + 0 + 3 = 3$

$r_{23} = k = 3$

	0	1	2	3	4
0	8 0 0	3 0 0	1 0 0	1 0 0	1 0 0
1	8 8 1	1 1 2	3 3 3	3 3 4	
2	12 19 1	9 13 2	6 8 3,4		
3	14 25 2	11 19 2			
4	16 32 2				

$w_{34} = w_{33} + P_4 + Q_4 = 1 + 1 + 1 = 3$

$c_{34} = 0$ $\{3 < k \leq 4\} \Rightarrow i = 3,\ j = 4,\ k = 4$

$= c[3,3] + c[4,4] + w(3,4) = 0 + 0 + 3 = 3$

$r_{34} = k = 4$

$w_{13} = w_{12} + P_3 + Q_3 = 7 + 1 + 1 = 9$

$c_{13} = i < k \leq j \Rightarrow i = 1,\ j = 3 \Rightarrow k = 2, 3$

$k=2 = c[1,1] + c[2,3] + w(1,3) = 0 + 3 + 9 = 12 \rightarrow k = 2$

$k=3 = c[1,2] + c[3,3] + w(1,3) = 7 + 0 + 9 = 16$ } 12

$r_{13} = k = 2$

$w_{24} = w_{23} + P_4 + Q_4 = 3 + 1 + 1 = 5$

$c_{24} = i < k \leq j \Rightarrow i = 2,\ j = 4 \Rightarrow k = 3, 4$

$k=3 \Rightarrow c[2,2] + c[3,4] = 0 + 3 = 3 \Rightarrow 3 + w(2,4)$ } 8

$k=4 \Rightarrow c[2,3] + c[4,4] = 3 + 0 = 3 \quad = 3 + 5 = 8$

$r_{24} = k = 3, 4$

$w_{02} = w_{01} + P_2 + Q_2 = 12 + 1 + 1 = 14$

$c_{02} = i < k \leq j \Rightarrow i = 0,\ j = 2 \Rightarrow k = 1, 2$

$k=1 = c[0,0] + c[1,2] = 0 + 12 = 12$

$k=2 \Rightarrow c[0,1] + c[2,2] = 8 + 3 = 11$ } $11 + w(0,2)$ $(k=2)$

$= 11 + 14 = 25$

$r_{02} = k = 2$

$w_{14} = w_{13} + P_4 + Q_4 = 9 + 1 + 1 = 11$

$c_{14} \Rightarrow i < k \leq j \Rightarrow i = 1,\ j = 4 \Rightarrow k = 2, 3, 4$

$k=2 \Rightarrow c[1,1] + c[2,4] = 0 + 8 = 8$

$k=3 \Rightarrow c[1,2] + c[3,4] = 7 + 3 = 10$

$k=4 \Rightarrow c[1,3] + c[4,4] = 12 + 0 = 12$ } $k=2$

$8 + w(1,4) = 8 + 11 = 19$

$W_{04} = W_{03} + P_4 + Q_4 = 14 + 1 + 1 = 16$

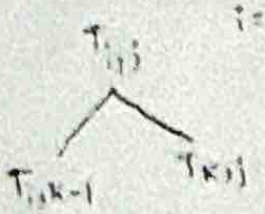
$C_{04} \Rightarrow i < k \leq j \Rightarrow i = 0, j = 4 \Rightarrow k = 1, 2, 3, 4$

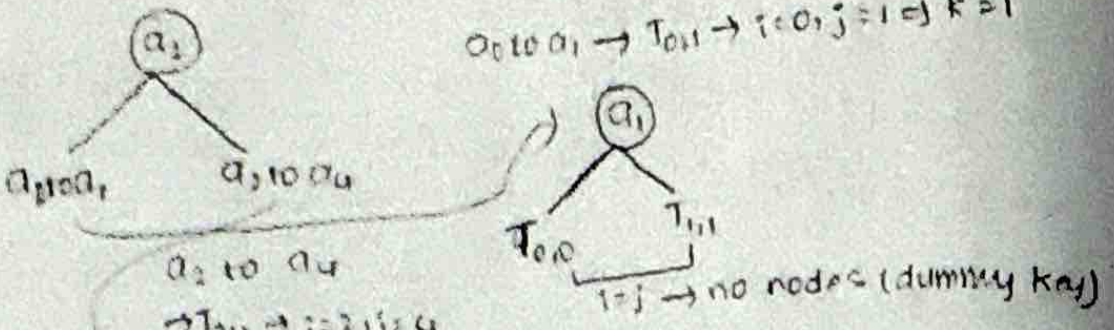
$k=1 \Rightarrow c[0,0] + c[1,4] = 0 + 19 = 19$
$k=2 \Rightarrow c[0,1] + c[2,4] = 2 + 8 = 16$
$k=3 \Rightarrow c[0,2] + c[3,4] = 19 + 3 = 22$
$k=4 \Rightarrow c[0,3] + c[4,4] = 25 + 0 = 25$

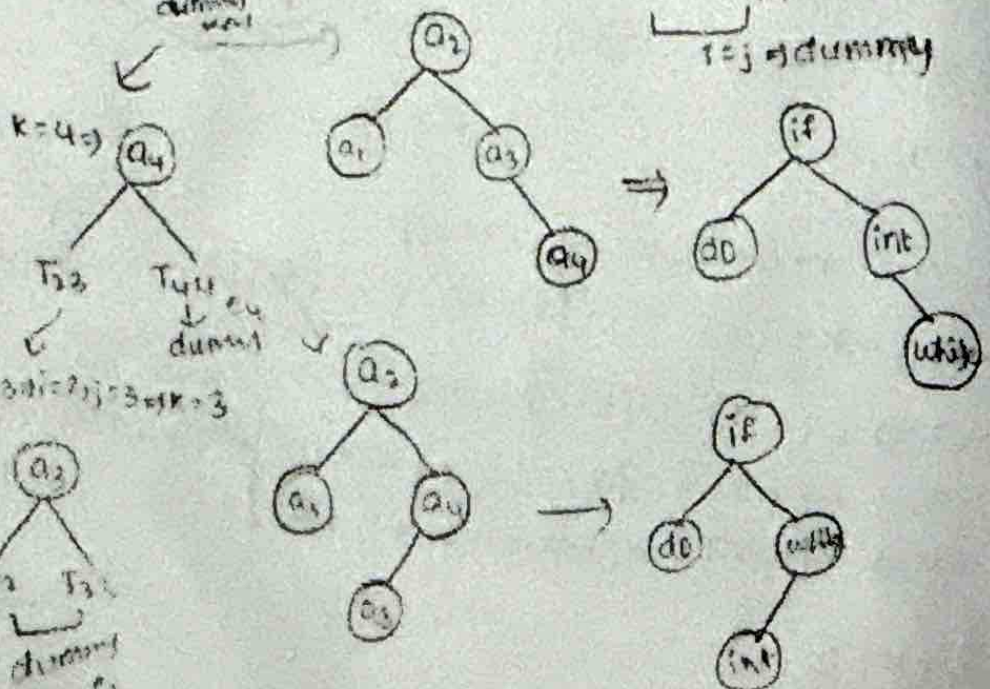
$k=2$
$16 + w(0,4)$
$= 16 + 16$
$= 32$

$r_{04} = k = 2$

$i=0, j=4 \Rightarrow k=2$







Algorithm:

```
Algorithm OBST (P, Q, n)
{
  for i := 0 to n-1 do {
      w[i,i] := q[i];
      r[i,i] := 0;
      c[i,i] := 0;
  }
  for m := 2 to n do {
      for i := 0 to n-m do {
          j := i+m;
          w[i,j] := w[i,j-1] + p[j] + q[j];
          k := Find(c,r,i,j);
          c[i,j] = c[i,k-1] + c[k+j] + w[i,j];
          r[i] = k;
      }
      write (c[0,n], w[0,n], r[0,0]);
  }
Algorithm Find(c,r,i,j){
    min := ∞;
    for( m := r[i,j+1] to r[i+1,j] do
        if (c[i,m-1] + c[m,j] < min) then {
            min := c[i,m-1] + c[m,j];
            l := m;
        }
    return l;
}
```

Analysis:
computation of each c[i,j] requires to find the minimum of m quantities, hence each such ~~cost of i,j~~ c[i,j] can be computed in O(m)

The total time for all c[i,j] is $O(m) \cdot O(n-m)$
i.e → $O(nm-m^2)$

Construct the optimal BST for the given instance $n = 4, \{a_1, a_2, a_3, a_4\} = \{$ count, float, if, while$\}$ i.e., while$\} = \{p_1, p_2, p_3, p_4\} = \{2, 1, 3, 3\}$ and $\{q_0, q_1, q_2, q_3, q_4\} = \{2, 3, 2, 3, 1\}$

	0	1	2	3	4
	2	3	2	3	1
0	0	0	0	0	0
	0	0	0	0	0
	3	5	8	7	
1	1	6	8	1	
	1	2	3	4	
	10	10	12		
2	14	16	17		
	1	3	3		
	16	16			
3	21	24			
	2	3			
	20				
4	23				
	3				

$w_{01} = w_{00} + p_1 + q_1$

$= 2 + 2 + 3 = 7$

$c_{01} = \min_{0 < k \le 1} \{c[0,0] + c[1,1]\} + w(0,1)$

$k = 1 \Rightarrow c[0,0] + c[1,1] + w(0,1)$

$= 0 + 0 + 7 = 7$

$r_{01} = 1$

$w_{12} = w_{11} + p_2 + q_2$

$= 3 + 1 + 2 = 6$

$c_{12} \Rightarrow \{1, 1\}, \{2, 2\}$

$c[1,1] + c[2,2] + w(1,2)$

$= 0 + 0 + 6 = 6$

$r_{12} = k = 2$

$w_{23} = w_{22} + p_3 + q_3 = 2 + 3 + 3 = 8$

$c_{23} \Rightarrow k = 3$

$c[2,2] + c[3,3] + w(2,3) = 0 + 0 + 8 = 8$

$r_{23} = 3$

$w_{34} = w_{33} + p_4 + q_4 = 3 + 3 + 1 = 7$

$c_{34} \Rightarrow k = 4$

$c[3,3] + c[4,4] + w(3,4) = 0 + 0 + 7 = 7$

$r_{34} = k = 4$

$w_{02} = w_{01} + p_2 + q_2 = 7 + 1 + 2 = 10$

$c_{02} \Rightarrow k = 1, 2$

$k = 1 \Rightarrow c[0,0] + c[1,2] = 0 + 6 = 6$

$k = 2 \Rightarrow c[0,1] + c[2,2] = 7 + 0 = 7$

$\left.\begin{matrix} k=1 \\ 6 + 10 = 16 \end{matrix}\right\}$

$r_{02} = 1$

$W_{13} = W_{12} + P_2 + Q_3 = 6 + 3 + 3 = 12$

$C_{13} \Rightarrow k = 2, 3$

$k=2 \Rightarrow c[1,1] + c[2,3] = 0 + 8 = 8$ } $k=3$
$k=3 \Rightarrow c[1,2] + c[3,3] = 6 + 0 = 6$ } $6+12 = 18$

$r_{13} = 3$

$W_{24} = W_{23} + P_4 + Q_4 = 8 + 3 + 1 = 12$

$C_{24} \Rightarrow k = 3, 4$

$k=3 \Rightarrow c[2,2] + c[3,4] = 0 + 1 = 1$ } $k=3$
$k=4 \Rightarrow c[2,3] + c[4,4] = 8 + 0 = 8$ } $1 + 12 = 19$

$r_{24} = 3$

$W_{03} = W_{02} + P_3 + Q_3 = 10 + 3 + 3 = 16$

$C_{03} \Rightarrow k = 1, 2, 3$

$k=1 \Rightarrow c[0,0] + c[1,3] = 0 + 18 = 18$ } $k=2$
$k=2 \Rightarrow c[0,1] + c[2,3] = 1 + 8 = 15$ } $15 + 16 = 31$
$k=3 \Rightarrow c[0,2] + c[3,3] = 16 + 0 = 16$

$r_{03} = 2$

$W_{14} = W_{13} + P_4 + Q_4 = 18 + 3 + 1 = 16$

$C_{14} \Rightarrow k = 2, 3, 4$

$k=2 \Rightarrow c[1,1] + c[2,4] = 0 + 19 = 19$ } $k=3$
$k=3 \Rightarrow c[1,2] + c[3,4] = 6 + 7 = 13$ } $13 + 16 = 29$
$k=4 \Rightarrow c[1,3] + c[4,4] = 18 + 0 = 18$

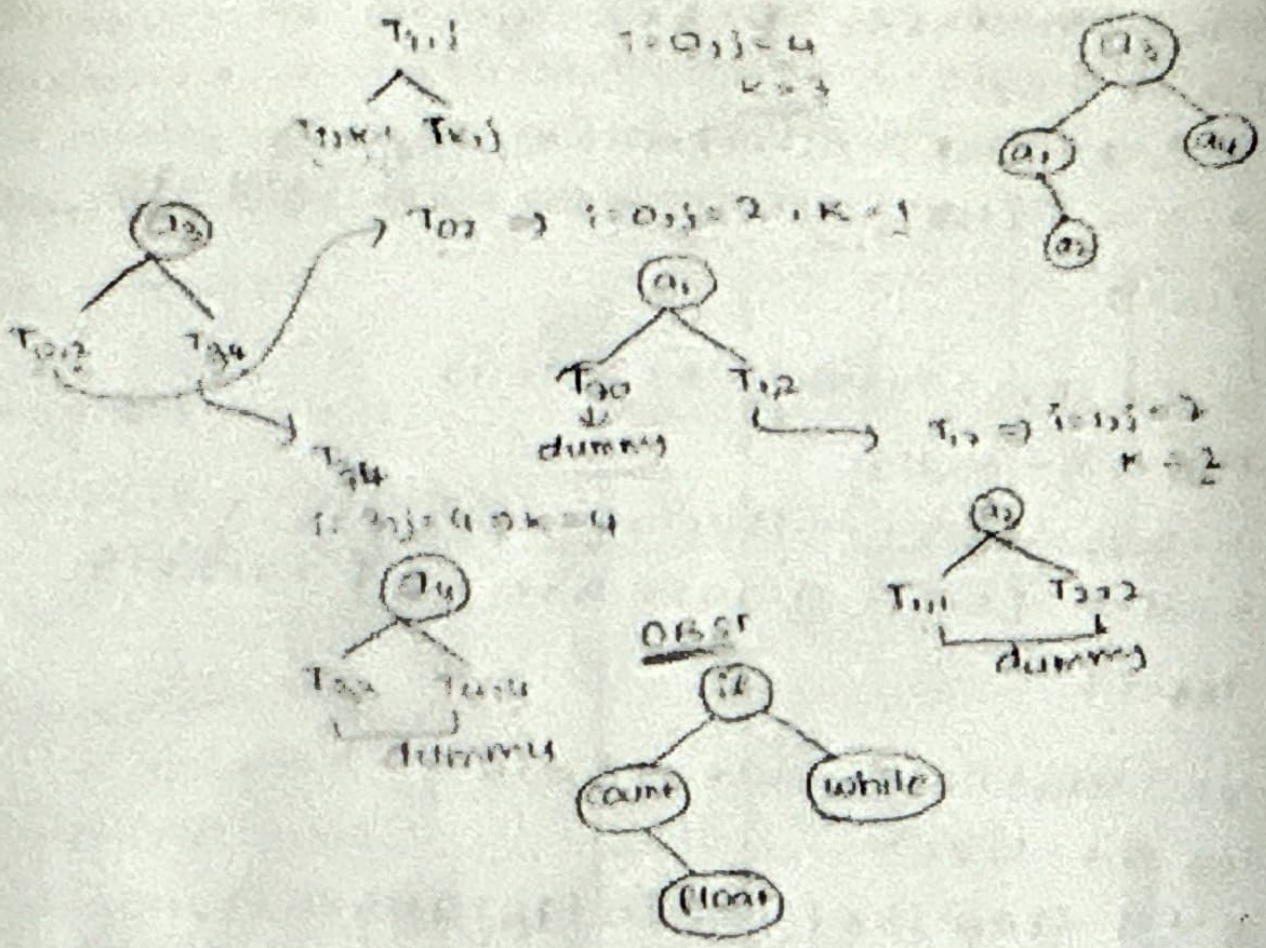
$r_{14} = 3$

$W_{04} = W_{03} + P_4 + Q_4 = 16 + 3 + 1 = 20$

$C_{04} \Rightarrow k = 1, 2, 3, 4$

$k=1 \Rightarrow c[0,0] + c[1,4] = 0 + 31 = 31$ } $k=3$
$k=2 \Rightarrow c[0,1] + c[2,4] = 7 + 19 = 26$ } $23 + 20 = 43$
$k=3 \Rightarrow c[0,2] + c[3,4] = 16 + 7 = 23$
$k=4 \Rightarrow c[0,3] + c[4,4] = 31 + 0 = 31$

$r_{04} = 3$



Reliability Design:

Reliability design problem is to design a system which is composed of several devices connected in a series. Let r_i be the reliability of the device D_i then the reliability of the entire system is $\prod r_i$. Our problem is to use device duplication to maximise reliability and cost constraint

Procedure

1. Reliability design problem typically rely on the property that less reliable devices are more duplicated than the more reliable devices.

2. Recursive call:

To setup a system within given cost by buying the devices, how many copies of devices each is needed to build a system within the

given cost to achieve max reliability

- each device should take must once, so that

$$Ec_i = (c_1 + c_2 + \dots + c_n)$$

$$\text{remaining} = C - Ec_i$$

(remaining ← system cost; Ec_i ← current devices cost)

- The remaining amount we can spend to buy the multiple copies of devices.
- How many copies of each device that we can buy will be calculated by using

$$m_i = \left\lfloor \frac{C - Ec_i}{c_i} \right\rfloor + 1$$

(m_i ← upperbound; +1 ← the device is already taken once; floor ← copies the device)

↳ represents the max. no. of copies a device can take

- The reliability of device is measured when it is repeated maximum no. of times

$$\phi_i(j) = (1-(1-r_i)^j)$$

$$j = 1, 2, \dots m_i$$

- Generate the set S^i where the set contains the possible elements $\in \{(r,c)\}$ that can be added to the system initially when no devices are added to the system.

$S^0 = \{(1,0)\}$ → initial

Reliabilities to be multiplied, costs are added.

so, $r=1, c=0$ initially

$$S_j^i = S^{i-1} + (Q_i(m_i) + \text{if } 1)$$

- (r_j, c_j) and (r_j, c_j) if $r_j < r_j$ and $c_i \geq c_j$ then remove (r_i, c_i) → purging rule / dominance rule.

Design a 3 stage system with device type

D	C	r
D_1	$30	0.9
D_2	$15	0.8
D_3	$20	0.5

$C = \$105$

$\sum C_i = 30+15+20 = 65$, $C - \sum C_i = 105 - 65 = 40$

$$UB_i = \left\lfloor \frac{C - \sum C_i}{C_i} \right\rfloor + 1$$

$UB_1 = \frac{40}{30} + 1 = 1.33 + 1 \approx 2$

$UB_2 = \frac{40}{15} + 1 = 2.4 + 1 \approx 3$

$UB_3 = \frac{40}{20} + 1 = 2 + 1 = 3$

When no devices is added initially

$S^0 = \{(1,0)\}$ → reliability, costs are multiplied; costs are added

consider device D_1 with $UB_1 = 2$

S_1^1 = D_1 is used only once

$= \{(1*0.9, 0+30)\} = \{(0.9, 30)\}$

S_2^1 = D_1 is repeated twice

$= \{(1-(1-r_1)^2, c_1+c_1)\}$

$= \{(1-(1-0.9)^2, 30+30)\}$

$S_2^1 = \{0.99, 60)\}$

$S^1 = \{$merge S_1^1 & $S_2^1\}$

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

consider D_2 with $UB_2 = 3$

$S_1^2 = \{(0.9 \times 0.8 \quad 80+15), (0.99 \times 0.8, 60+15)\}$

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

$S_2^2 = \{0.9(1-(1-0.8)^2), 80+15+15), (0.99(1-(1-0.8)^2), 60+15+15)\}$

$= \{(0.864, 80), (0.9504, 90)\}$

↳ total budget increased

$S_3^2 = \{0.9(1-(1-0.8)^3), 80+15+15+15), (0.99(1-(1-0.8)^3), 60+15+15+15)\}$

$S_3^2 = \{(0.8928, 95), (0.98208, 105)\}$

↳ discard because there is one more item to be added and the budget increases

$S^2 = \{(0.72, 45), (0.792, 75), (0.864, 80), (0.8928, 95)\}$

↓ discard (marginal) $(x_1 < x_2, c_1 > c_2)$

$S^2 = \{(0.72, 45), (0.864, 80), (0.8928, 95)\}$

consider D_3 with $UB_3 = 3$

$S_1^3 = \{(0.72 \times 0.5, 45+20), (0.864 \times 0.5, 80+20), (0.8928 \times 0.5, 95+20)\}$

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

$S_2^3 = \{(0.72(1-(1-0.5)^2), 45+20+20), (0.864(1-(1-0.5)^2), 80+20+20), (0.8928(1-(1-0.5)^2), 95+20+20)\}$

$= \{(0.54, 85), (0.6048, 100), (0.66960, 115)\}$

discarded 115 > 101

$S_3^3 = \{(0.72(1-(1-0.5)^3), 45+20+20+20), (0.864(1-(1-0.5)^3), 80+20+20+20), (0.8928(1-(1-0.5)^3), 95+20+20+20)\}$

$S_3^3 = \{(0.63, 105), (0.756, 120), (0.7812, 135)\}$

discard discard

$S^3 = \{(0.72,105), (0.648,100), (0.594,75), (0.36,65),$
$(0.432,60), (0.414,45), (0.54,85),$
$(0.648,100), (0.63,105)\}$

(not discarded)

$S^3 = \{(0.72,105), (0.648,100), (0.594,75), (0.36,65),$
$(0.432,60), (0.54,85), (0.648,100)\}$

$S^3 = \{(0.36,65), (0.432,60), (0.54,85), (0.648,100),$
$(0.63,105)\}$

max cost = 105

max reliability in below 105

$(0.648,100) \in S_3^2$

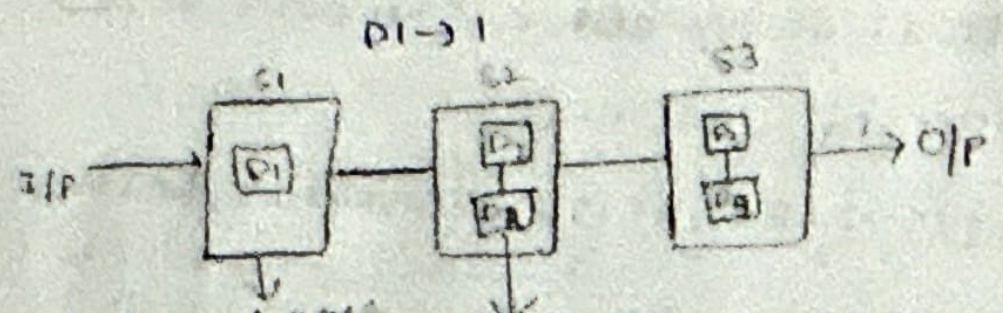
D3 → 2

$(7, 100-40) = (7,60) \in S_2^1$

D2 → 2

$(7, 60-(15+15)) = (7,30) \in S_1^1$

D1 → 1



For the given values of reliabilities and costs the above system can be made with reliability 0.648 and system cost 100.

- Backtracking to design a system: After generating coln S^3 select the pair (0.648,100) that has max reliability within system cost when compared to other pairs coln set S^3.

so, $(0.648, 100) \in S_3^2$ then device 3 is duplicated twice

so, $\boxed{D3 = 2}$.

subtract the device 3 cost from the cost achieved from solution

$$100 - (20+20) = 60$$

D3=2 times.

60th set with 60 system cost belongs to $S_1^2 \Rightarrow (1, 60) \in S_1^2$, i.e, $(0.869, 60)$, i.e, D2 duplicated twice $\Rightarrow \boxed{D2 = 2}$

$$60 - (15+15) = 30$$

D2 2 times.

select the pair with cost = 30 from S_1' & S_1'

$(0.9, 30) \in S_1' \Rightarrow$ i.e, D1 duplicate only once

$\boxed{D1 = 1}$

3/5/24 Unit - 4 : Backtracking & Branch and Bound

Backtracking:

It is an algorithm design technique which uses brute force technique to find the desired output

- The term backtracking suggest that if the current soln is not suitable for the given problem then go back chooses the another alternative to the desired output.
- The idea behind the backtracing approach is that it searches for the soln to a problem among all the available options initially we start from one possible option. If the problem is solved with that option. then we return the soln else we backtrack and select another option from the remaining availability of options.
- There also might be a case where none of the options available for the given problem and hence we understand backtracking won't give any soln to the particular problem.
- Backtracking uses recursive approach because the process of finding soln from various options available is repeated recursively until we don't find the soln or we reach the final stage. So we can conclude backtracking at every step eliminates those choices that cannot give us the soln and proceed those choices that have the potential of taking us to the soln.

Terminology:

In backtracking technique, we will solve the problem in efficient way compared to other methods like greedy method,

the soln is based on finding one or more vectors that maximizes or minimises or satisfy a criteria function.

$P(x_1, x_2, \ldots x_n)$ – criteria function

- Form a soln at any point seems not promising, ignore it. all possible solns and required cer. of all constraints which are divided into two types of constraints

1. **Explicit constraints:** these are the rules that restricts each x_i to take on values only from a given set $\{0, 1\}$ (satisfied). x_i value should be either '1' or '0' → '0' (not satisfied) represents unselection, '1' → selection

2. **Implicit constraints:** These are the rules that determine which of the tuples in the soln space satisfies the criteria function.

Solution space:
All the tuples that satisfy explicit constraints defined in for a particular instance of a problem is called as soln space.

Problem state:
The state that is defined by all the nodes within the tree organisation is known as problem state.

Solution state:
These are the problem states for which the path from root to leaf node defines a tuple in the soln space. All soln states are indicated with square node.

* **State space tree:**
It is a set of paths from root node to other nodes and is a tree organisation of the solns of the given problem as soln space.

Answer state
The soln state s for which the path from root node to s defines a tuple that is a member of the set of solutions of the given problem.

* Live node:
A node which has been generated but whose children had not yet been generated

E-node [Expanded-node]
The live node whose children are currently being generated is called E-node

Dead-node:
It is a generated node that is either not to be expanded further or one for which all of its children has been generated.

General method of backtracking:
Basic idea of backtracking is to build up a vector one component at a time and to test whether the vector being formed has any chance of success.

Major advantage of this algorithm approach is that we can realise the fact that the partial vector generated does not lead to an optimal soln, in such situations that vector can be ignored

Backtracking is a depth first search with some bounding functions. all soln's using pabacktracking are required to satisfy a complex set of constraints that constraints may be implicit or explicit.

```
Algorithm Backtrack(k) {
// on entering the first k-1 values from x(1) to x(k-1)
// the col vector x[1:m] has been assigned
// x[] is global variables
for (each x[k] ∈ T(x[1], ... x[n]) do
{
  if (Bk(x[1], x[2], ... x[k]) ≠ 0) then
  {
    if (x[1], x[2], ... x[k] is a path to an
        answer node)
      then write(x[1:k]);
    if x[k] then Backtrack(k+1));
  }
}
```

Applications:

Backtracking is an algorithm design technique that can easily solve large instances of combinational problems.

1. N-Queen's problem.
2. Sum of subset problem
3. Graph-colouring problem
4. Hamiltonian cycle.

Q N-Queen's problem

N=1 to 5.