UNIT-3

DYNAMIC PROGRAMMING

General method-multistage graphs-all pair shortest path algorithm-Optimal Binary Search trees-0/1 knapsack Problem-Reliability design-traveling salesman problem

BASIC SEARCH AND TRAVERSAL TECHNIQUES

The techniques-and/or graphs-bi_connected components-depth first search- -breadth first search.

DYNAMIC PROGRAMING

- ➢ The idea of dynamic programming is thus quit simple: avoid calculating the same thing twice, usually by keeping a table of known result that fills up a sub instances are solved.
- ➢ Divide and conquer is a top-down method.
- ➢ When a problem is solved by divide and conquer, we immediately attack the complete instance, which we then divide into smaller and smaller sub-instances as the algorithm progresses.
- ➢ Dynamic programming on the other hand is a bottom-up technique.
- ➢ We usually start with the smallest and hence the simplest sub- instances.
- ➢ By combining their solutions, we obtain the answers to sub-instances of increasing size, until finally we arrive at the solution of the original instances.
- ➢ The essential difference between the greedy method and dynamic programming is that the greedy method only one decision sequence is ever generated.
- ➢ In dynamic programming, many decision sequences may be generated. However, sequences containing sub-optimal sub-sequences can not be optimal and so will not be generated.

MULTISTAGE GRAPH

1. A multistage graph G = (V,E) is a directed graph in which the vertices are portioned into $K \geq 2$ disjoint sets Vi, $1 \leq i \leq k$.
2. In addition, if < u,v > is an edge in E, then u < = Vi and V ∑ Vi+1 for some i, $1 \leq i < k$.
3. If there will be only one vertex, then the sets Vi and Vk are such that [Vi]=[Vk] = 1.
4. Let ‘s’ and ‘t’ be the source and destination respectively.

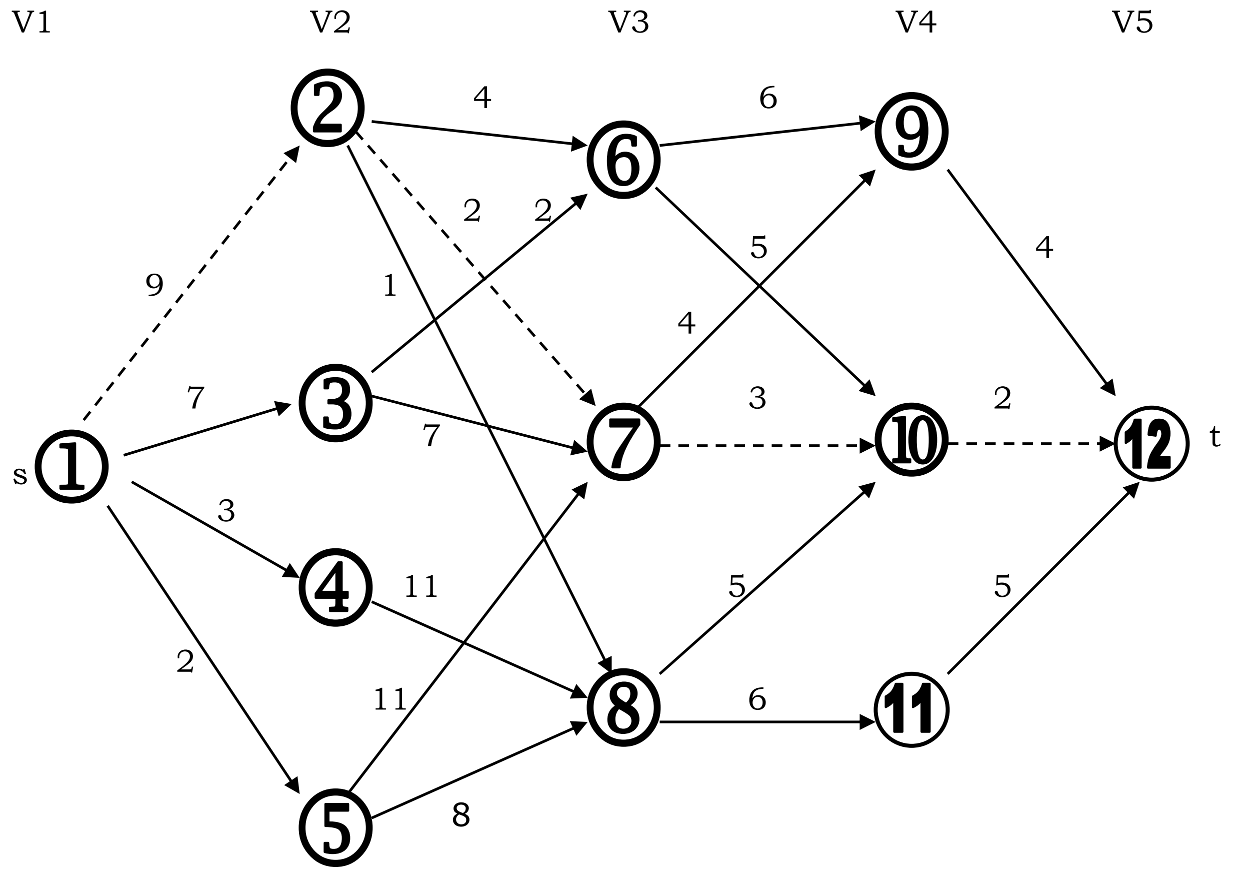
5. The cost of a path from source (s) to destination (t) is the sum of the costs of the edger on the path.
6. The *MULTISTAGE GRAPH* problem is to find a minimum cost path from 's' to 't'.
7. Each set Vi defines a stage in the graph. Every path from 's' to 't' starts in stage-1, goes to stage-2 then to stage-3, then to stage-4, and so on, and terminates in stage-k.
8. This *MULISTAGE GRAPH* problem can be solved in 2 ways.

   a) Forward Method.
   b) Backward Method.

FORWARD METHOD

1. Assume that there are 'k' stages in a graph.
2. In this *FORWARD* approach, we will find out the cost of each and every node starling from the 'k' th stage to the 1st stage.
3. We will find out the path (i.e.) minimum cost path from source to the destination (i.e,) [Stage-1 to Stage-k].

PROCEDURE:



- Maintain a cost matrix cost (n) which stores the distance from any vertex to the destination.
- If a vertex is having more than one path, then we have to choose the minimum distance path and the intermediate vertex, which gives the minimum distance path, will be stored in the distance array 'D'.
- In this way we will find out the minimum cost path from each and every vertex.
- Finally cost(1) will give the shortest distance from source to destination.
- For finding the path, start from vertex-1 then the distance array D(1) will give the minimum cost neighbor vertex which in turn give the next nearest vertex and proceed in this way till we reach the Destination.
- For a 'k' stage graph, there will be 'k' vertex in the path.
- In the above graph V1…V5 represent the stages. This 5 stage graph can be solved by using forward approach as follows,

STEPS: -		**DESTINATION, D**
Cost (12)=0	$\longrightarrow$	D (12)=0
Cost (11)=5	$\longrightarrow$	D (11)=12
Cost (10)=2	$\longrightarrow$	D (10)=12
Cost (9)=4	$\longrightarrow$	D (9)=12

- For forward approach,

$$\boxed{\text{Cost}(i,j) = \min_{\substack{l \in V_{i+1} \\ (j,l) \in E}} \{C(j,l) + \text{Cost}(i+1,l)\}}$$

Cost(8) = min {C (8,10) + Cost (10), C (8,11) + Cost (11) }
= min (5 + 2, 6 + 5)
= min (7,11)
= 7
cost(8) =7 =>D(8)=10

cost(7) = min(c (7,9)+ cost(9),c (7,10)+ cost(10))
(4+4,3+2)
= min(8,5)
= 5
cost(7) = 5 =>D(7) = 10

cost(6) = min (c (6,9) + cost(9),c (6,10) +cost(10))
= min(6+4 , 5 +2)
= min(10,7)
= 7
cost(6) = 7 =>D(6) = 10

cost(5) = min (c (5,7) + cost(7),c (5,8) +cost(8))

```
           = min(11+5 , 8 +7)
           = min(16,15)
           = 15
cost(5)    = 15 =>D(5) = 18

cost(4)    = min (c (4,8) + cost(8))
           = min(11+7)
           = 18
cost(4)    = 18 =>D(4) = 8

cost(3)    = min (c (3,6) + cost(6),c (3,7) +cost(7))
           = min(2+7 , 7 +5)
           = min(9,12)
           = 9
cost(3)    = 9 =>D(3) = 6

cost(2)    = min (c (2,6) + cost(6),c (2,7) +cost(7) ,c (2,8)  +cost(8))
           = min(4+7 , 2+5 , 1+7 )
           = min(11,7,8)
           = 7

cost(2)    = 7 =>D(2) = 7


cost(1)    = min (c (1,2)+cost(2) ,c (1,3)+cost(3) ,c (1,4)+cost(4) ,c(1,5)+cost(5))
            = min(9+7 , 7 +9 , 3+18 , 2+15)
            = min(16,16,21,17)
            = 16
cost(1)    = 16 =>D(1) = 2
```

$\dashrightarrow$ The path through which you have to find the shortest distance.

(i.e.)

$$\begin{tikzcd} \textcircled{1} \arrow[r] & \textcircled{2} \arrow[r] & \textcircled{7} \arrow[r] & \textcircled{10} \arrow[r] & \textcircled{12} \end{tikzcd}$$

Start from vertex - 2

```
D ( 1)  =  2
D ( 2)  =  7
D ( 7)  = 10
D (10) = 12
```

So, the minimum –cost path is,

$$\begin{tikzcd} \textcircled{1} \arrow[r, "9"] & \textcircled{2} \arrow[r, "2"] & \textcircled{7} \arrow[r, "3"] & \textcircled{10} \arrow[r, "2"] & 12 \end{tikzcd}$$

∴ The cost is 9+2+3+2+=16

ALGORITHM: FORWARD METHOD

Algorithm FGraph (G,k,n,p)

```
// The i/p is a k-stage graph G=(V,E) with 'n' vertex.
// Indexed in order of stages E is a set of edges.
// and c[i,j] is the cost of<i,j>,p[1:k] is a minimum cost path.
{
    cost[n]=0.0;
     for j=n-1 to 1 step-1 do
  {
     //compute cost[j],
    // let 'r' be the vertex such that <j,r> is an edge of 'G' &
   // c[j,r]+cost[r] is minimum.

   cost[j] = c[j+r] + cost[r];
    d[j] =r;
   }
  // find a minimum cost path.

       P[1]=1;
       P[k]=n;
       For j=2 to k-1 do
       P[j]=d[p[j-1]];
}
```

ANALYSIS:

The time complexity of this forward method is $O(|V| + |E|)$

BACKWARD METHOD

- If there are 'K' stages in a graph using back ward approach. we will find out the cost of each & every vertex starting from 1st stage to the kth stage.
- We will find out the minimum cost path from destination to source (i.e.,) from stage k to stage 1.

PROCEDURE:

1. It is similar to forward approach, but differs only in two or three ways.
2. Maintain a cost matrix to store the cost of every vertices and a distance matrix to store the minimum distance vertex.
3. Find out the cost of each and every vertex starting from vertex 1 up to vertex k.

4. To find out the path star from vertex 'k', then the distance array D (k) will give the minimum cost neighbor vertex which in turn gives the next nearest neighbor vertex and proceed till we reach the destination.

STEP:

Cost(1) = 0 => D(1)=0
Cost(2) = 9 => D(2)=1
Cost(3) = 7 => D(3)=1
Cost(4) = 3 => D(4)=1
Cost(5) = 2 => D(5)=1

Cost(6) =min(c (2,6) + cost(2),c (3,6) + cost(3))
=min(13,9)

cost(6) = 9 =>D(6)=3

Cost(7) =min(c (3,7) + cost(3),c (5,7) + cost(5) ,c (2,7) + cost(2))
=min(14,13,11)

cost(7) = 11 =>D(7)=2

Cost(8) =min(c (2,8) + cost(2),c (4,8) + cost(4) ,c (5,8) +cost(5))
=min(10,14,10)

cost(8) = 10 =>D(8)=2

Cost(9) =min(c (6,9) + cost(6),c (7,9) + cost(7))
=min(15,15)

cost(9) = 15 =>D(9)=6

Cost(10)=min(c(6,10)+cost(6),c(7,10)+cost(7)),c (8,10)+cost(8)) =min(14,14,15)
cost(10)= 14 =>D(10)=6

Cost(11) =min(c (8,11) + cost(8))

cost(11) = 16 =>D(11)=8

cost(12)=min(c(9,12)+cost(9),c(10,12)+cost(10),c(11,12)+cost(11))
=min(19,16,21)

cost(12) = 16 =>D(12)=10

PATH:

Start from vertex-12

    D(12) = 10
    D(10) = 6
    D(6) = 3
    D(3) = 1

So the minimum cost path is,

$1 \xrightarrow{7} 3 \xrightarrow{2} 6 \xrightarrow{5} 10 \xrightarrow{2} 12$

The cost is 16.

ALGORITHM : BACKWARD METHOD

Algorithm BGraph (G,k,n,p)

```
// The I/p is a k-stage graph G=(V,E) with 'n' vertex.
// Indexed in order of stages E is a set of edges.
// and c[i,J] is the cost of<i,j>,p[1:k] is a minimum cost path.
{
     bcost[1]=0.0;
      for j=2 to n do
   {
      //compute bcost[j],
     //   let 'r' be the vertex such that <r,j> is an edge of 'G' &
    //   bcost[r]+c[r,j] is minimum.

    bcost[j] = bcost[r] + c[r,j];
     d[j] =r;
    }
  // find a minimum cost path.

        P[1]=1;
        P[k]=n;
        For j= k-1 to 2 do
        P[j]=d[p[j+1]];
}
```

ALL PAIR SHORTEST PATH

❖ Let G=<V,E> be a directed graph 'V' is a set of nodes and 'E' is the set of edges.

❖ Each edge has an associated non-negative length.

❖ We want to calculate the length of the shortest path between each pair of nodes.

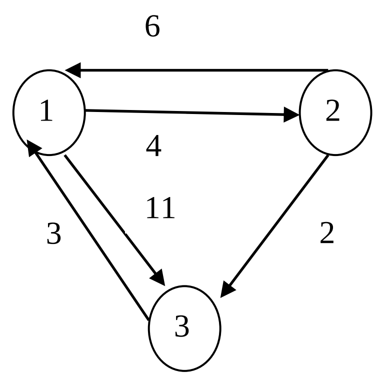
- ❖ Suppose the nodes of G are numbered from 1 to n, so V={1,2,...N},and suppose G matrix L gives the length of each edge, with L(i,j)=0 for i=1,2...n,L(i,j)>=for all i & j, and L(i,j)=infinity, if the edge (i,j) does not exist.
- ❖ The principle of optimality applies: if k is the node on the shortest path from i to j then the part of the path from i to k and the part from k to j must also be optimal, that is shorter.
- ❖ First, create a cost adjacency matrix for the given graph.
- ❖ Copy the above matrix-to-matrix D, which will give the direct distance between nodes.
- ❖ We have to perform N iteration after iteration k. the matrix D will give you the distance between nodes with only (1,2...,k) as intermediate nodes.
- ❖ At the iteration k, we have to check for each pair of nodes (i,j) whether or not there exists a path from i to j passing through node k.

$$\boldsymbol{A^k(i,j) = min\{A^{k-1}(i,j), A^{k-1}(i,k) + A^{k-1}(k-1,j)\}} \qquad \mathbf{k \geq 1}$$

$$\boldsymbol{A^k(i,j) - Length\ of\ the\ shortest\ path\ from\ i\ to\ j.}$$

Let G=(V,E) is a directed graph with 'N' Vertices. Let cost be the cost of the adjacency matrix for G such that cost(i,i)=0 (1≤i≤n) then cost(i,j) is the length or cost of the edge <i,j>. If <i,j> exists the edge belongs to E(G) and cost(i,j)=α, if i≠j and <i,j> $\notin$ E(G)

Example:



Matrix Representation:

$$\begin{bmatrix} 0 & 4 & 11 \\ 6 & 0 & 2 \\ 3 & \alpha & 0 \end{bmatrix}$$

$A^0 =$

$$\begin{bmatrix} 0 & 4 & 11 \\ 6 & 0 & 2 \\ 3 & \alpha & 0 \end{bmatrix}$$

The path of the vertices through vertex-1 is given as
$A^1 =$

$$\begin{bmatrix} 0 & 4 & 11 \\ 6 & 0 & 2 \\ 3 & 7 & 0 \end{bmatrix}$$

The path of the vertices through vertex-2 is given as
$A^2 =$

$$\begin{bmatrix} 0 & 4 & 6 \\ 6 & 0 & 2 \\ 3 & 7 & 0 \end{bmatrix}$$

The path of the vertices through vertex-3 is given as
$A^3 =$

$$\begin{bmatrix} 0 & 4 & 6 \\ 5 & 0 & 2 \\ 3 & 7 & 0 \end{bmatrix}$$

- At 1st iteration we have to check the each pair(i,j) whether there is a path through node 1.if so we have to check whether it is minimum than the previous value and if I is so than the distance through 1 is the value of d1(i,j).at the same time we have to solve the intermediate node in the matrix position p(i,j).

ALGORITHM :

```
Function AllPair (L[1..r,1..r]):array[1..n,1..n]
array D[1..n,1..n]

D = L
For   k = 1 to n do
For   i  = 1 to n do
For   j  = 1 to n do
D [ i , j ] = min (D[ i, j ], D[ i, k ] + D[ k, j ]
Return D
```

ANALYSIS:

This algorithm takes a time of O(n^3)

OPTIMAL BINARY SEARCH TREES

If n-identifiers exists, every internal node represents a P and every external node represents a point where an unsuccessful search may terminate.

The cost of searching an identifier a_i is P(i)*level(a_i)
The cost of unsuccessful search which terminates in E_i is q(i)*(level(E_i)-1)
Therefore the cost of the binary search tree is

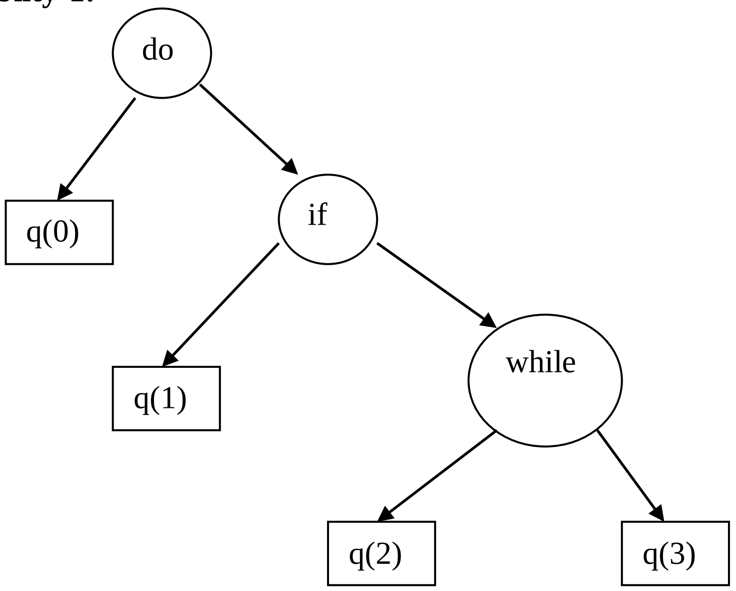
$$\sum_{1\le i\le n} P(i) * level(a_i) + \sum_{0\le i\le n} q(i) * (level(E_i) - 1)$$

Criterion for an optimal tree (principle of optimality): Each optimal binary search tree is composed of a root and (at most) two optimal subtrees, the left and the right. Or, stated in another way: in an optimal tree, all the subtrees must also be optimal with respect to the keys that they contain.

Example: Find OBST for a set {a1,a2,a3}={do,if,while} if P(1)=0.5,P(2)=0.1,P(3)=0.05 and q(0)=0.15, q(1)=0.1,q(2)=0.05,q(3)=0.05

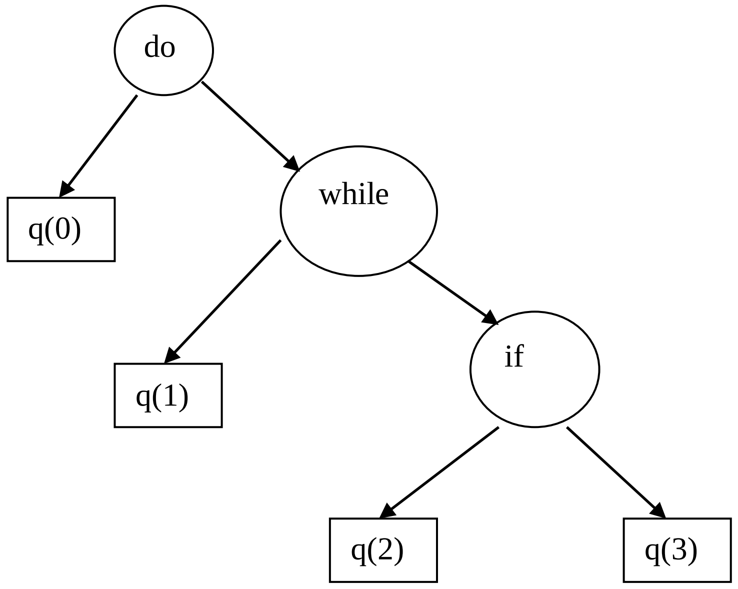
Solution:
The possible binary trees are many.

Possiblity-1:



Cost=(0.5*1+0.1*2+0.05*3)+(0.15*1+0.1*2+0.05*3+0.05*3)=1.50

Possiblity-2:



Cost=(0.5*1+0.05*2+0.1*3)+(0.15*1+0.1*2+0.05*3+0.05*3)=0.9+0.65=1.7

Possiblity-3:
Possiblity-4:

:

:

:

We get many possible binary search trees and out of which first tree is having the minimum cost i.e., the optimal binary search tree.

Example1: Compute w(i,j), c(i,j), r(i,j) for the identifier set n = 4, ($a1$,$a2$,$a3$,$a4$) = {do,if,int,while} with P(1:4) = {3,3,1,1} and q = {2,3,1,1,1}. Using the r (i,j)s construct the optimal binary search tree.

Solution:

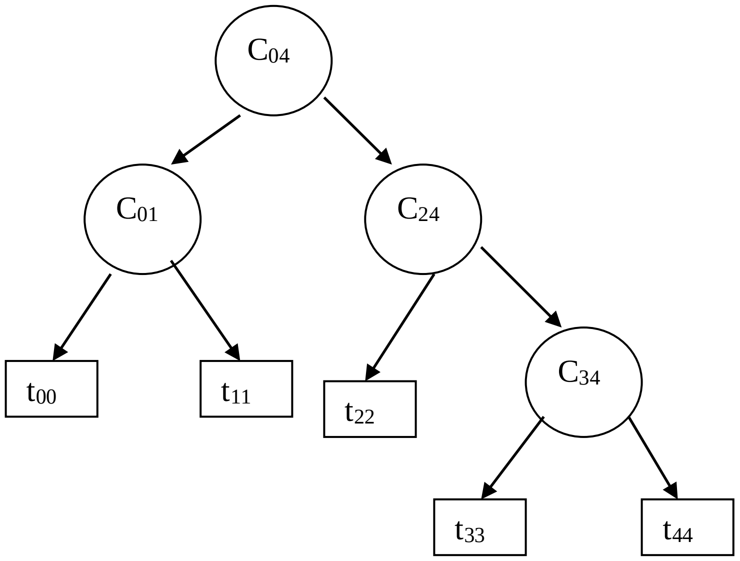
$$w(i,j) = P(j) + q(j) + w(i,j-1)$$

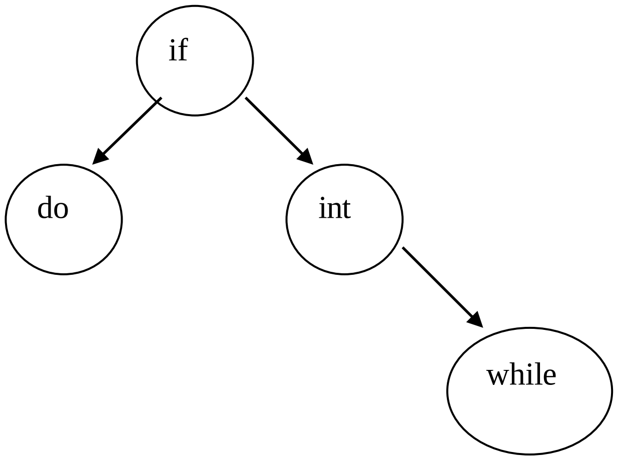
$$cost(i,j) = \min_{i<k\le j}\{cost(i,k-1) + cost(k,j)\} + w(i,j)$$

w(i,i)=q(i), C(i,i)=r(i,i)=0

The 'K' value which gives minimum cost is chosen as 'r'.

	0	**1**	**2**	**3**	**4**
j-i=0	W(0,0)=2 C(0,0)=0 r(0,0)=0	W(1,1)=3 C(1,1)=0 r(1,1)=0	W(2,2)=1 C(2,2)=0 r(2,2)=0	W(3,3)=1 C(3,3)=0 r(3,3)=0	W(4,4)=1 C(4,4)=0 r(4,4)=0
j-i=1	W(0,1)=8 C(0,1)=8 r(0,1)=1	W(1,2)=7 C(1,2)=7 r(1,2)=2	W(2,3)=3 C(2,3)=3 r(2,3)=3	W(3,4)=3 C(3,4)=3 r(3,4)=4	
j-i=2	W(0,2)=12 C(0,2)=19 r(0,2)=1	W(1,3)=9 C(1,3)=12 r(1,3)=2	W(2,4)=5 C(2,4)=8 r(2,4)=3		
j-i=3	W(0,3)=14 C(0,3)=25 r(0,3)=2	W(1,4)=11 C(1,4)=19 r(1,4)=2			
j-i=4	W(0,4)=16 C(0,4)=32 r(0,4)=2				





This algorithm takes a time of $O(n^3)$.

0/1 KNAPSACK PROBLEM

- This problem is similar to ordinary knapsack problem but we may not take a fraction of an object.

- We are given ' N ' objects with weight W_i and profits P_i where i varies from l to N and also a knapsack with capacity ' M '.

- The problem is, we have to fill the bag with the help of ' N ' objects and the resulting profit has to be maximum.

- Formally, the problem can be started as, maximize $\sum_{i=l}^{n} X_i P_i$

  subject to $\sum_{i=l}^{n} X_i W_i \leq M$

- Where X_i are constraints on the solution $X_i \in \{0,1\}$. X_i is required to be 0 or 1. If the object is selected then the unit is 1. If the object is rejected than the unit is 0. That is why it is called as 0/1, knapsack problem.

- Time complexity is O(nW).n is the number of items and W is the weight of the knapsack.

- For X_n , two states are possible

  - If X_n is included in the knapsack then the remaining capacity is M- W_n and profit earned is P_n.
  - If X_n is not included in the knapsack then capacity remains M and no profit is earned.

Therefore, we have to select a state such that it maximizes the result. This can be shown as

$$f_n(M)=\text{Max}\ \{ f_{n-1}(x),\ f_{n-1}(M-W_n)+P_n\}$$

When $f_n(M)$—maximizing profit function for object 'n' where capacity is M.
This can be generalized as

$$f_i(y)=\text{Max}\ \{ f_i(y),\ f_{i-1}(y-W_i)+P_i\}$$

We solve 0/1 knapsack problem using the above equation.

By generating set of ordered pairs (P,W) where P= $f_i(y)$, W=y_i
The set S^{i+1} is computed from S^i, where

$$S_1^i = \{(P,W) | (P - P_i), (W - W_i) \epsilon S^i\}$$

S^{i+1} is obtained by merging and purging the order pairs of S^i and S_1^i . Purging is done by applying dominance rule.

Dominance Rule: If S^{i+1}continas two ordered pairs (P_j,W_j), (P_k,W_k) such that $P_j<P_k$ and $W_j>W_k$ then the ordered pair (P_j,W_j) is discarded.

Example: Given n=3 and M=6, (P1,P2,P3)=(1,2,5), (W1,W2,W3)=(2,3,4). Find the optimal solution for 0/1 knapsack.

Solution:

S^0 ={(0,0)}

$S_1^0 = \{P_1 + 0, W_1 + 0)\}$
={(1+0,2+0)}={(1,2)}
S^1 ={(0,0),(1,2)}
Apply dominance rule for purging

$S_1^1 = \{(2 + 0,3 + 0), (2 + 1,3 + 2)\}$
={(2,3),(3,5)}
S^2= {(0,0),(1,2),(2,3),(3,5)}
Apply dominance rule for purging

$S_1^2 = \{(5 + 0,4 + 0), (5 + 1,4 + 2), (5 + 2,4 + 3), (5 + 3,4 + 5)\}$
={(5,4),(6,6),(7,7),(8,9)}
S^3= {(0,0),(1,2),(2,3),(3,5), (5,4),(6,6),(7,7),(8,9)}
(7,7) , (8,9) are discarded, as maximum capacity is only 6.
Apply dominance rule for purging. (3,5) is discarded using purging (3<5,5>4) as we need maximum profit.
Hence, S^3= {(0,0),(1,2),(2,3), (5,4),(6,6)}
Select the largest ordered pair i.e., (6,6)

X_3=(6,6) ϵS^3
(6,6) $\notin S^2$
Therefore, X_3=1

This implies the knapsack is filled now and the profit is 6. For next ordered pair, remove 3rd object profit and weight.

i.e., (6-5,6-4)=(1,2)

The next ordered pair is (1,2)

as adding (5,4) is giving (6,6)

$X_2=(1,2) \in S^2$
$(1,2) \in S^1$
Therefore, $X_2=0$. It cannot be placed.

The next ordered pair is (1,2)
$X_1=(1,2) \in S^1$
$(1,2) \notin S^0$
Therefore, $X_1=1$.

The optimal solution is (x1,x2,x3)=(1,0,1)

Exercise: Generate the sets S^i $0\leq i\leq 4$
(P1,P2,P3,P4)=(2,5,8,1), (W1,W2,W3,W4)=(10,15,16,9).

RELIABILITY DESIGN

The problem is to design a system that is composed of several devices connected in series. Let r_i be the reliability of device D_i (that is r_i is the probability that device i will function properly) then the reliability of the entire system is $\prod r_i$. Even if the individual devices are very reliable (the r_i's are very close to one), the reliability of the system may not be very good. For example, if n = 10 and $r_i = 0.99$, i < i < 10, then $\prod r_i = .904$.
Hence, it is desirable to duplicate devices. Multiple copies of the same device type are connected in parallel. If stage i contains mi copies of device D_i. Then the probability that all m_i have a malfunction is $(1 - r_i)^{mi}$. Hence the reliability of stage i becomes $1 – (1 - r\ )^{mi}$. The reliability of stage 'i' is given by a function $\Phi i\ (m_i)$.
Our problem is to use device duplication. This maximization is to be carried out under a cost constraint. Let c_i be the cost of each unit of device i and let c be the maximum allowable cost of the system being designed.
We wish to solve:
Maximize

$$\prod_{1\leq i\leq n} \varphi_i(m_i)$$

Subject to

$$\sum_{1\leq i\leq n} C_i m_i < C$$

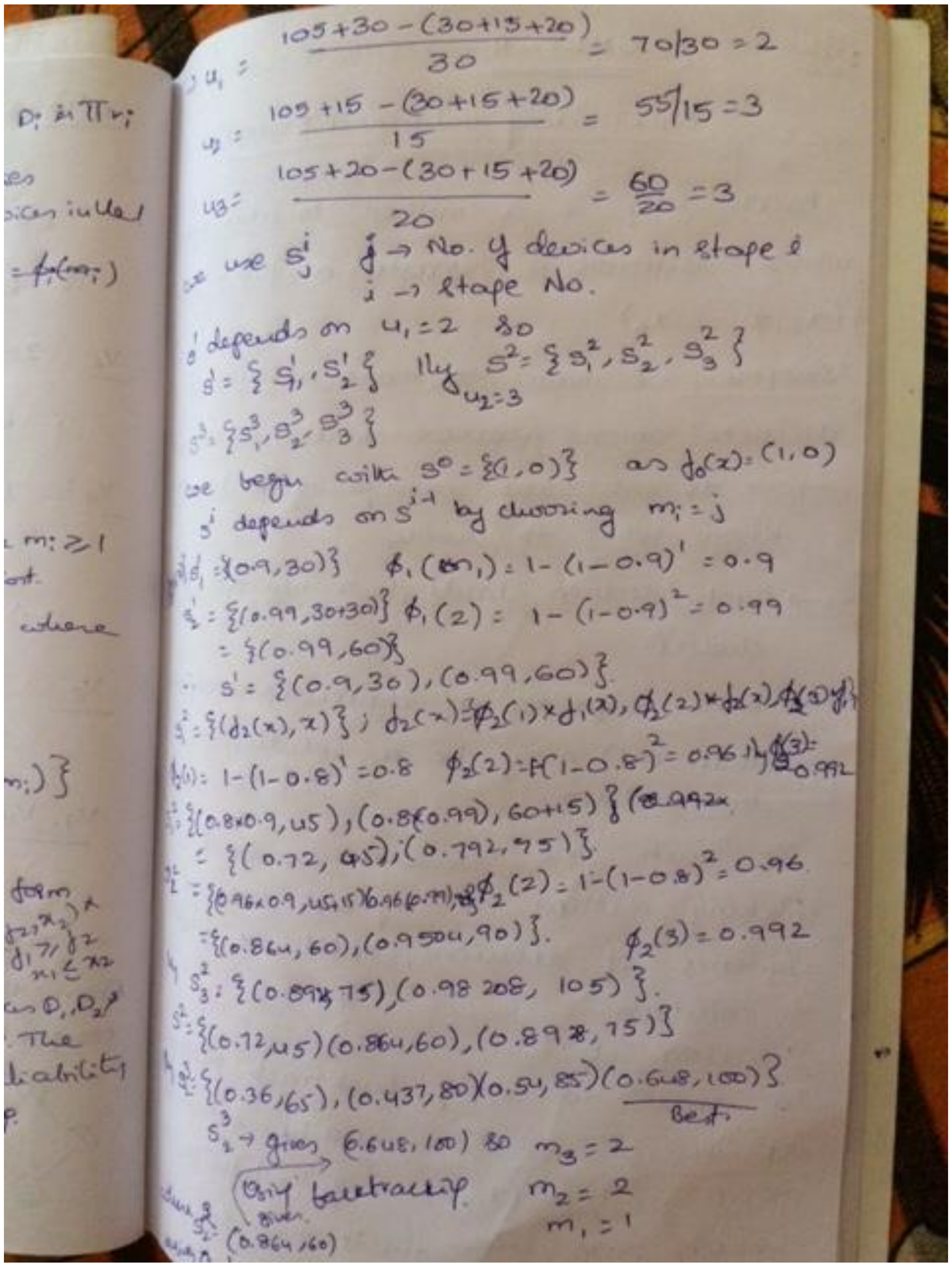
$m_i \geq 1$

mi > 1 and interger, 1 < i < n

Example 1:

Design a three stage system with device types D1, D2 and D3. The costs are $30, $15 and $20 respectively. The Cost of the system is to be no more than $105. The reliability of each device is 0.9, 0.8 and 0.5 respectively.

Solution:



$u_1 = \frac{105+30-(30+15+20)}{30} = 70/30 = 2$

$u_2 = \frac{105+15-(30+15+20)}{15} = 55/15 = 3$

$u_3 = \frac{105+20-(30+15+20)}{20} = \frac{60}{20} = 3$

use S^i_j $j \rightarrow$ No. of devices in stage i

$i \rightarrow$ stage No.

S^1 depends on $u_1 = 2$ so

$S^1 = \{S^1_1, S^1_2\}$ Ily $S^2 = \{S^2_1, S^2_2, S^2_3\}$ $u_2 = 3$

$S^3 = \{S^3_1, S^3_2, S^3_3\}$

we begin with $S^0 = \{(1,0)\}$ as $f_0(x) = (1,0)$

S^i depends on S^{i-1} by choosing $m_i = j$

$S^1_1 = \{(0.9, 30)\}$ $\phi_1(m_1) = 1-(1-0.9)^1 = 0.9$

$S^1_2 = \{(0.99, 30+30)\}$ $\phi_1(2) = 1-(1-0.9)^2 = 0.99$

$= \{(0.99, 60)\}$

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

$S^2_1 = \{(f_2(x), x)\}$; $f_2(x) = \{\phi_2(1) \times f_1(x), \phi_2(2) \times f_2(x), \phi_2(3) f_1\}$

$\phi_2(1) = 1-(1-0.8)^1 = 0.8$ $\phi_2(2) = 1-(1-0.8)^2 = 0.96$ Ily $\phi_2(3) = 0.992$

$S^2_1 = \{(0.8 \times 0.9, 45), (0.8(0.99), 60+15)\}$

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

$S^2_2 = \{(0.96 \times 0.9, 45+15), (0.96(0.99), \ldots$ $\phi_2(2) = 1-(1-0.8)^2 = 0.96$

$= \{(0.864, 60), (0.9504, 90)\}$. $\phi_2(3) = 0.992$

Ily $S^2_3 = \{(0.8928, 75), (0.98208, 105)\}$.

$S^2 = \{(0.72, 45) (0.864, 60), (0.8928, 75)\}$

Ily $S^3_2 = \{(0.36, 65), (0.437, 80) (0.54, 85) \underline{(0.648, 100)}\}$

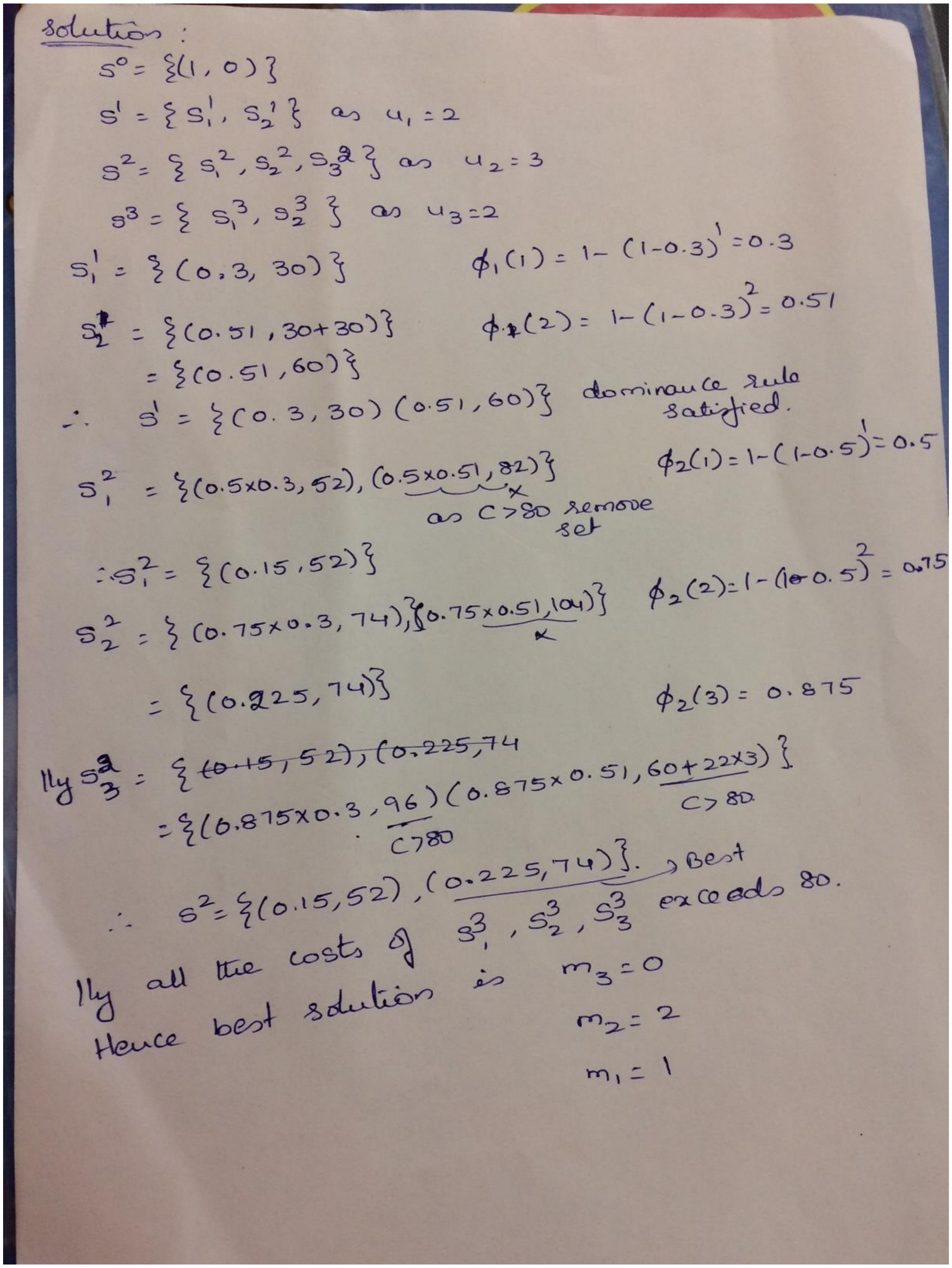
Best

$S^3_2 \rightarrow$ gives $(0.648, 100)$ so $m_3 = 2$

Using backtracking $m_2 = 2$

S^2_2 given $(0.864, 60)$ $m_1 = 1$

Example2: Consider three stages of a system with r1=0.3, r2=0.5, r3=0.2, c1=30, c2=22, c3=30. Where the total cost of the system is C=80 and u1=2, u2=3, u3=2 find the reliability design.



solution :

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

$S^1 = \{S^1_1, S^1_2\}$ as $u_1 = 2$

$S^2 = \{S^2_1, S^2_2, S^2_3\}$ as $u_2 = 3$

$S^3 = \{S^3_1, S^3_2\}$ as $u_3 = 2$

$S^1_1 = \{(0.3, 30)\}$ $\qquad \phi_1(1) = 1-(1-0.3)^1 = 0.3$

$S^1_2 = \{(0.51, 30+30)\}$ $\qquad \phi_1(2) = 1-(1-0.3)^2 = 0.51$

$= \{(0.51, 60)\}$

$\therefore$ $S^1 = \{(0.3, 30)(0.51, 60)\}$ dominance rule satisfied.

$S^2_1 = \{(0.5\times0.3, 52), (\underbrace{0.5\times0.51, 82}_{\times})\}$ $\qquad \phi_2(1) = 1-(1-0.5)^1 = 0.5$

as $C > 80$ remove set

$\therefore S^2_1 = \{(0.15, 52)\}$

$S^2_2 = \{(0.75\times0.3, 74), \{\underbrace{0.75\times0.51, 104}_{\times})\}$ $\qquad \phi_2(2) = 1-(1-0.5)^2 = 0.75$

$= \{(0.225, 74)\}$ $\qquad \phi_2(3) = 0.875$

IIly $S^2_3 = \{$ ~~(0.15, 52), (0.225, 74~~ $(0.875\times0.51, \underbrace{60+22\times3}_{C>80})\}$

$= \{(0.875\times0.3, \underbrace{96}_{C>80})(\ldots$

$\therefore$ $S^2 = \{(0.15, 52), (\underbrace{0.225, 74}))\}.$ $\rightarrow$ Best

IIly all the costs of S^3_1, S^3_2, S^3_3 exceeds 80.

Hence best solution is $m_3 = 0$

$m_2 = 2$

$m_1 = 1$

TRAVELLING SALESMAN PROBLEM

- Let G(V,E) be a directed graph with edge cost c_{ij} is defined such that $c_{ij} > 0$ for all i and j and $c_{ij} = \propto$,if $<i,j> \notin$ E.
  Let |V| =n and assume n>1.
- The traveling salesman problem is to find a tour of minimum cost.
- A tour of G is a directed cycle that include every vertex in V.
- The cost of the tour is the sum of cost of the edges on the tour.
- The tour is the shortest path that starts and ends at the same vertex i.e., 1.

APPLICATION:

1. Suppose we have to route a postal van to pick up mail from the mail boxes located at ‘n’ different sites.
2. An n+1 vertex graph can be used to represent the situation.
3. One vertex represents the post office from which the postal van starts and returns.
4. Edge <i,j> is assigned a cost equal to the distance from site ‘i’ to site ‘j’.
5. The route taken by the postal van is a tour and we are finding a tour of minimum length.
6. Every tour consists of an edge <1,k> for some k $\in$ V-{} and a path from vertex k to vertex 1.
7. The path from vertex k to vertex 1 goes through each vertex in V-{1,k} exactly once.
8. The function which is used to find the path is

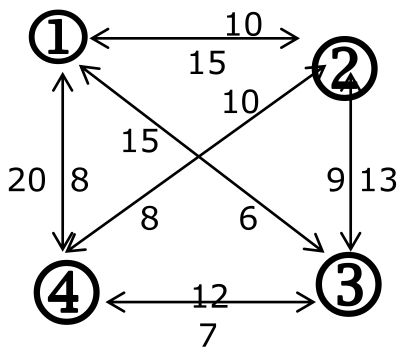
   $g(1,V-\{1\}) = \min\{ c_{ij} + g(j,s-\{j\})\}$

9. g(i,s) be the length of a shortest path starting at vertex i, going through all vertices in S and terminating at vertex 1.
10. The function g(1,v-{1}) is the length of an optimal tour.

STEPS TO FIND THE PATH:

1. Find $g(i,\Phi) = c_{i1}$, $1 <= i < n$, hence we can use equation(2) to obtain g(i,s) for all s to size 1.
2. That we have to start with s=1,(ie) there will be only one vertex in set ‘s’.
3. Then s=2, and we have to proceed until |s| <n-1.

Example1:



Cost matrix

$$\begin{vmatrix} 0 & 10 & 15 & 20 \\ 5 & 0 & 9 & 10 \\ 6 & 13 & 0 & 12 \\ 8 & 8 & 9 & 0 \end{vmatrix}$$

$$\begin{tikzcd} g(i,s) \arrow[r] \arrow[d] & \text{set of nodes/vertex have to visited.} \\ \text{starting position} \end{tikzcd}$$

$g(i,s) = \min\{c_{ij} + g(j, s-\{j\})$

STEP 1:

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

$\min\{10+25,15+25,20+23\}$
$\min\{35,35,43\}$
$=35$

STEP 2:

$g(2,\{3,4\}) = \min\{c_{23}+g(3\{4\}),c_{24}+g(4,\{3\})\}$

$\min\{9+20,10+15\}$
$\min\{29,25\}$
$=25$

$g(3,\{2,4\}) =\min\{c_{32}+g(2\{4\}),c_{34}+g(4,\{2\})\}$

$\min\{13+18,12+13\}$
$\min\{31,25\}$
$=25$

$g(4,\{2,3\}) = \min\{c_{42}+g(2\{3\}),c_{43}+g(3,\{2\})\}$

$\min\{8+15,9+18\}$
$\min\{23,27\}$
$=23$

STEP 3:

1. $g(3,\{4\}) = \min\{c_{34} + g\{4,\Phi\}\}$
   $12+8 = 20$

2. g(4,{3}) = min{c_{43} +g{3,Φ}}
   9+6 =15

3. g(2,{4}) = min{c_{24} +g{4,Φ}}
   10+8 =18

4. g(4,{2}) = min{c_{42} +g{2,Φ}}
   8+5 =13

5. g(2,{3}) = min{c_{23} +g{3,Φ}}
   9+6=15

6. g(3,{2}) = min{c_{32} +g{2,Φ}}
   13+5=18

STEP 4:

g{4,Φ} =c_{41} = 8

g{3,Φ} =c_{31} = 6

g{2,Φ} =c_{21} = 5

$|s|$=0

i =1 to n.

g(1,Φ) = c_{11} => 0

g(2,Φ) = c_{21} => 5

g(3,Φ) = c_{31} => 6

g(4,Φ) = c_{41} => 8

$|s|$=1

i =2 to 4

g(2,{3}) = c_{23} + g(3,Φ)
= 9+6 =15

g(2,{4}) = c_{24} + g(4,Φ)
= 10+8 =18

$g(3,\{2\}) = c_{32} + g(2,\Phi)$
$= 13+5 = 18$

$g(3,\{4\}) = c_{34} + g(4,\Phi)$
$= 12+8 = 20$

$g(4,\{2\}) = c_{42} + g(2,\Phi)$
$= 8+5 = 13$

$g(4,\{3\}) = c_{43} + g(3,\Phi)$
$= 9+6 = 15$

$|s|=2$

$i \neq 1$, $1 \in s$ and $i \in s$.

$g(2,\{3,4\}) = \min\{c_{23}+g(3\{4\}),c_{24}+g(4,\{3\})\}$
$\min\{9+20,10+15\}$
$\min\{29,25\}$
$=25$

$g(3,\{2,4\}) = \min\{c_{32}+g(2\{4\}),c_{34}+g(4,\{2\})\}$
$\min\{13+18,12+13\}$
$\min\{31,25\}$
$=25$

$g(4,\{2,3\}) = \min\{c_{42}+g(2\{3\}),c_{43}+g(3,\{2\})\}$
$\min\{8+15,9+18\}$
$\min\{23,27\}$
$=23$
$|s|=3$

$g(1,\{2,3,4\})=\min\{c_{12}+g(2\{3,4\}),c_{13}+g(3,\{2,4\}),c_{14}+g(4,\{2,3\})\}$
$\min\{10+25,15+25,20+23\}$
$\min\{35,35,43\}$
$=35$

optimal cost is 35

the shortest path is,

$g(1,\{2,3,4\}) = c_{12} + g(2,\{3,4\})$ => 1->2

$g(2,\{3,4\}) = c_{24} + g(4,\{3\})$ => 1->2->4

$g(4,\{3\}) = c_{43} + g(3\{\Phi\})$ => 1->2->4->3->1

so the optimal tour is **1 → 2 → 4 →3 → 1**

BASIC SEARCH AND TRAVERSAL TECHNIQUE

DEFINING GRAPH:

A graphs g consists of a set V of vertices (nodes) and a set E of edges (arcs). We write G=(V,E). V is a finite and non-empty set of vertices. E is a set of pair of vertices; these pairs are called as edges. Therefore, V(G) is read as V of G, is a set of vertices and E(G),read as E of G is a set of edges.

An edge e= (v, w) is a pair of vertices v and w, and to be incident with v and w.

A graph can be pictorially represented as follows,

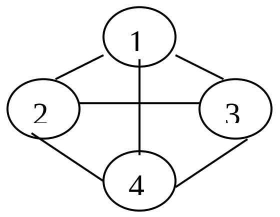


FIG: Graph G

We have numbered the graph as 1,2,3,4. Therefore, V(G)=(1,2,3,4) and E(G) = {(1,2),(1,3),(1,4),(2,3),(2,4)}.

BASIC TERMINOLOGIES OF GRAPH:

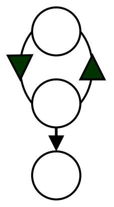
UNDIRECTED GRAPH:

An undirected graph is that in which, the pair of vertices representing the edges is unordered.

DIRECTED GRAPH:

An directed graph is that in which, each edge is an ordered pair of vertices, (i.e.) each edge is represented by a directed pair. It is also referred to as digraph.

DIRECTED GRAPH



COMPLETE GRAPH:

An n vertex undirected graph with exactly n(n-1)/2 edges is said to be complete graph. The graph G is said to be complete graph .

TECHNIQUES FOR GRAPHS:

- The fundamental problem concerning graphs is the reach-ability problem.
- In it simplest from it requires us to determine whether there exist a path in the given graph, G +(V,E) such that this path starts at vertex ‘v’ and ends at vertex ‘u’.
- A more general form is to determine for a given starting vertex v6 V all vertex ‘u’ such that there is a path from if it u.
- This problem can be solved by starting at vertex ‘v’ and systematically searching the graph ‘G’ for vertex that can be reached from ‘v’.
- We describe 2 search methods for this.

  i. Breadth first Search and Traversal.
  ii. Depth first Search and Traversal.

BREADTH FIRST SEARCH AND TRAVERSAL

Breadth First Search:

In Breadth first search we start at vertex v and mark it as having been reached. The vertex v at this time is said to be unexplored. A vertex is said to have been explored by an algorithm when the algorithm has visited all vertices adjacent from it. All unvisited vertices adjacent from v are visited next. There are new unexplored vertices. Vertex v has now been explored. The newly visited vertices have not been explored and are put onto the end of the list of unexplored vertices. The first vertex on this list is the next to be explored. Exploration continues until no unexplored vertex is left. The list of unexplored vertices acts as a queue and can be represented using any of the standard queue representations.

- In Breadth First Search we start at a vertex ‘v’ and mark it as having been reached (visited).
- The vertex ‘v’ is at this time said to be unexplored.
- A vertex is said to have been explored by an algorithm when the algorithm has visited all vertices adjust from it.
- All unvisited vertices adjust from ‘v’ are visited next. These are new unexplored vertices.
- Vertex ‘v’ has now been explored. The newly visit vertices have not been explored and are put on the end of a list of unexplored vertices.
- The first vertex on this list in the next to be explored. Exploration continues until no unexplored vertex is left.
- The list of unexplored vertices operates as a queue and can be represented using any of the start queue representation.

ALGORITHM:

```
Algorithm BFS (v)

// A breadth first search of 'G' is carried out.
// beginning at vertex-v; For any node i, visit.
// if 'i' has already been visited. The graph 'v'
// and array visited [] are global; visited []
// initialized to zero.
{ y=v; // q is a queue of unexplored 1visited (v)= 1
repeat
{ for all vertices 'w' adjacent from u do
    { if (visited[w]=0) then
       {Add w to q;
         visited[w]=1
           }
  }
if q is empty then return;// No delete u from q;
      } until (false)
}

algrothim : breadth first traversal
       algorithm BFS(G,n)
    {
     for i= 1 to n do
        visited[i] =0;
     for i =1 to n do
      if (visited[i]=0)then BFS(i)
   }
```

Here the time and space required by BFT on an n-vertex e-edge graph one O(n+e) and O(n) resp. if adjacency list and adjacency matrix is used then the bounds are $O(n^2)$ and O(n) resp.

DEPTH FIRST SEARCH

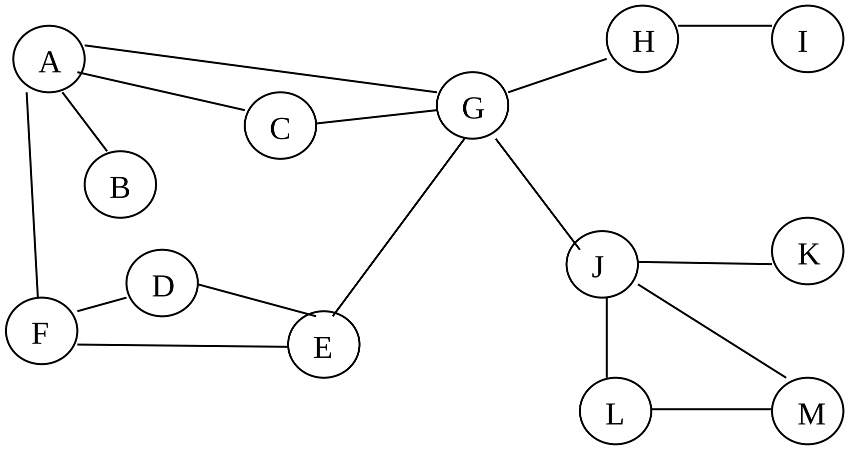
A depth first search of a graph differs from a breadth first search in that the exploration of a vertex v is suspended as soon as a new vertex is reached. At this time the exploration of the new vertex u begins. When this new vertex has been explored, the exploration of u continues. The search terminates when all reached vertices have been fully explored. This search process is best-described recursively.

```
Algorithm DFS(v)
{
visited[v]=1
for each vertex w adjacent from v do
{
If (visited[w]=0)then
DFS(w);
}
}
```

BI-CONNECTED COMPONENT AND DFS

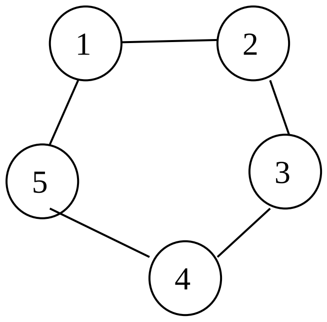
Articulation Point: An articulation point in a connected graph is a vertex that if deleted, would break the graph into two or more pieces (Connected components)

Considering the below graph



A,G,H,J are the articulation points for the above graph.

Bi-connected graph: A graph with no articulation point is said to be bi-connected graph. A graph is bi-connected iff any vertex is deleted, the graph remains connected.



Bi-connected Component: A bi-connected component of a graph is a maximal bi-connected subgraph. A graph which is not bi-connected can divide into bi-connected components set of nodes mutually accessible via two distinct paths.

From the above graph {A,G,E,F},{J,L,M},{K,I,B} are bi-connected components.

How to find articulation point:

1. Find the depth first spanning tree T for graph G.
2. Add back edges represented as --→ in T. Back edges are the edges which have no connection in DFS but there is an edge existing in original graph.
3. Finding Articulation point Nodes
    a. If ‘i’ is the root of T and has atleast 2 children, then ‘i’ is an articulation point.
    b. Leaf nodes are not articulation points
    c. ‘i’ is an articulation point, if ‘i’ is not the root and has a child which does not connect to the existing vertices even with back edges.

Short Answer Questions:

1. Define principle of Optimality
2. What is depth first search?
3. What is multistage graph?
4. State travelling Salesperson problem.
5. Difference between Divide-and-Conquer and Dynamic Programming.
6. Explain 0/1 Knapsack.
7. Define articulation point and Bi-connected component.
8. Mention any two applications of DFS.
9. What is spanning tree of a graph?
10. Differentiate Greedy and Dynamic programming approaches.
11. What is a Hamiltonian Graph?

Long Answer Questions:

1. Explain the general method of Dynamic Programming.

2. What do you understand by all pairs shortest path problem and compare different algorithms pertinent to this problem.

3. Describe dynamic programming method. How dynamic programming can be used to solve optimal binary search trees?

4. Explain the detail about travelling salesperson problem with suitable example.

5. What is a Bi-connected component? Write a scheme to construct a Bi-connected graph.

6. What is multi-stage graph? Write dynamic programming expressions for Multistage graph corresponding to forward and backward approaches and explain with an example.

7. Write an algorithm to find the shortest path in a multi stage graph using dynamic programming.

8. Write the algorithm for Optimal Binary Search Tree. Show that the computing time of function OBST is Θ(n^2).

9. What are the differences between dynamic programming and divide-and-conquer techniques?

10. Consider the three stages of a system with (r1,r2,r3) = (0.2,0.5,0.3) and (c1,c2,c3) = (20,30,20) where the total cost of system 70 and (u1,u2,u3) = (3,2, 2) find the reliability design.

11. Using the optimal binary search tree algorithm compute w(i,j),R(i,j) and C(I,j), 0<=i<=j<=4 for the identifier set (a1,a2,a3,a4)=(end,goto,print,stop) with P(1)=1/20, P(2)=1/5, P(3)=1/10, P(4)=1/20,Q(0)=1/5,Q(1)=1/10,Q(2)=1/5,Q(3)=1/20,Q(4)=1/20. Using R(i.j)'s constru t OBST.

12. Compute w(i,j), c(i,j), r(i,j) for the identifier set n = 4, (*a*1,*a*2,*a*3,*a*4) = {cout, float, int, while} with P(1:4) = {2,1,1,2} and q = {2,3,1,1,1}. Using the r (i,j)s construct the optimal binary search tree.

13. Briefly argue how the principle of a optimality holds for 0/1 knapsack problem, generate the sets. S^i, 0<=i<=4. Where (w1,w2,w3,w4)=(10,15,6,9) and (P1,P2,P3,P4)=(2,5,8,9)—State the purging rules used. If knapsack capacity is m=25, what is optimal solution.