UNIT - 3

Dynamic programming

Introduction:

Dynamic programming is an algorithm design method that can be used when the solution to a problem can be viewed as the result of a sequence of decisions.

For some of the problems that may be viewed in this way, an optimal sequence of decisions can be found by making the decisions one at a time and never making an erroneous decision. This is true for all problems solvable by the greedy method. For many other problems, it is not possible to make stepwise decisions in such a manner that the sequence of decisions made is optimal.

One way to solve problems for which it is not possible to make a sequence of stepwise decisions leading to an optimal decision sequence is to try all possible decision sequences.

In dynamic programming an optimal sequence of decisions is obtained by making explicit appeal to the principle of optimality.

Principle of optimality:

The principle of optimality states that an optimal sequence of decisions has the property that whatever the initial state and decision are, the remaining decisions must constitute an optimal decision sequence with regard to the state resulting from the first decision.

Thus, the essential difference between the greedy method and dynamic programming is that in the greedy method only one decision sequence is ever generated. In dynamic programming, many decision sequences may be generated. However, sequences containing suboptimal subsequences cannot be optimal.

Multistage Graphs:-

A multistage Graph $G=(V,E)$ is a directed graph, where the vertices are partitioned into $k \geq 2$, disjoint sets V_i, where $1 \leq i \leq n$. Every edge connects two nodes from two partitions. Each edge is associated with an edge cost $C(i,j)$. The initial node is called a source and the last node is called sink. The multistage problem is a problem of finding the shortest path from the source to the destination. This problem is also called the 'stage coach problem'. The dynamic programming approach can be used to solve this problem.

Applying the dynamic programming approach, the problem is divided into subproblems, called stages.

Dynamic programming starts with a small problem. A smaller problem is a nearly completed journey with just one more stage to go from the current stage to the destination. Then the problem is enlarged by adding one more stage to the current problem. It is observed that there is no need for recomputation, as the already-stored stage cost can be reused. The shortest path can be obtained using both the forward and the backward computation procedure.

Forward Computation procedure:

In this, the decision x_i is made in terms of the optimal decisions $(x_{n-1}, x_n, \ldots, x_1)$,

~~which in the~~

Informal algorithm:-

<u>Step-1</u>:- Read directed Graph $G = \langle V, E \rangle$ with k stages.

<u>Step-2</u>:- Let n be the number of nodes and dist[1...k] be the distance array

<u>Step 3</u>:- Set initial cost to zero

<u>Step-4</u>:- loop index i from $(n-1)$ to 1.

i, Find a vertex v from the next stage such that the edge connecting the current stage and the next stage is minimum, that is

(j, v) + Cost(v) is minimum.

stage vertex.

ii, update cost and store v in another array dist[].

<u>Step-5</u>:- Return cost

<u>Step-6</u>:- End.

The shortest path can finally be obtained or reconstructed by tracking the distance array.

Algorithm MFG (G, K, n)

{

// Graph G, K is the number of stages, n vertices.

cost = 0

n = |V|

stage = n-1

while (j <= stage) do

{ choose a vertex v such that c[j,v] + cost v is minimum

cost[j] = c[j,v] + cost(v)

j = j-1

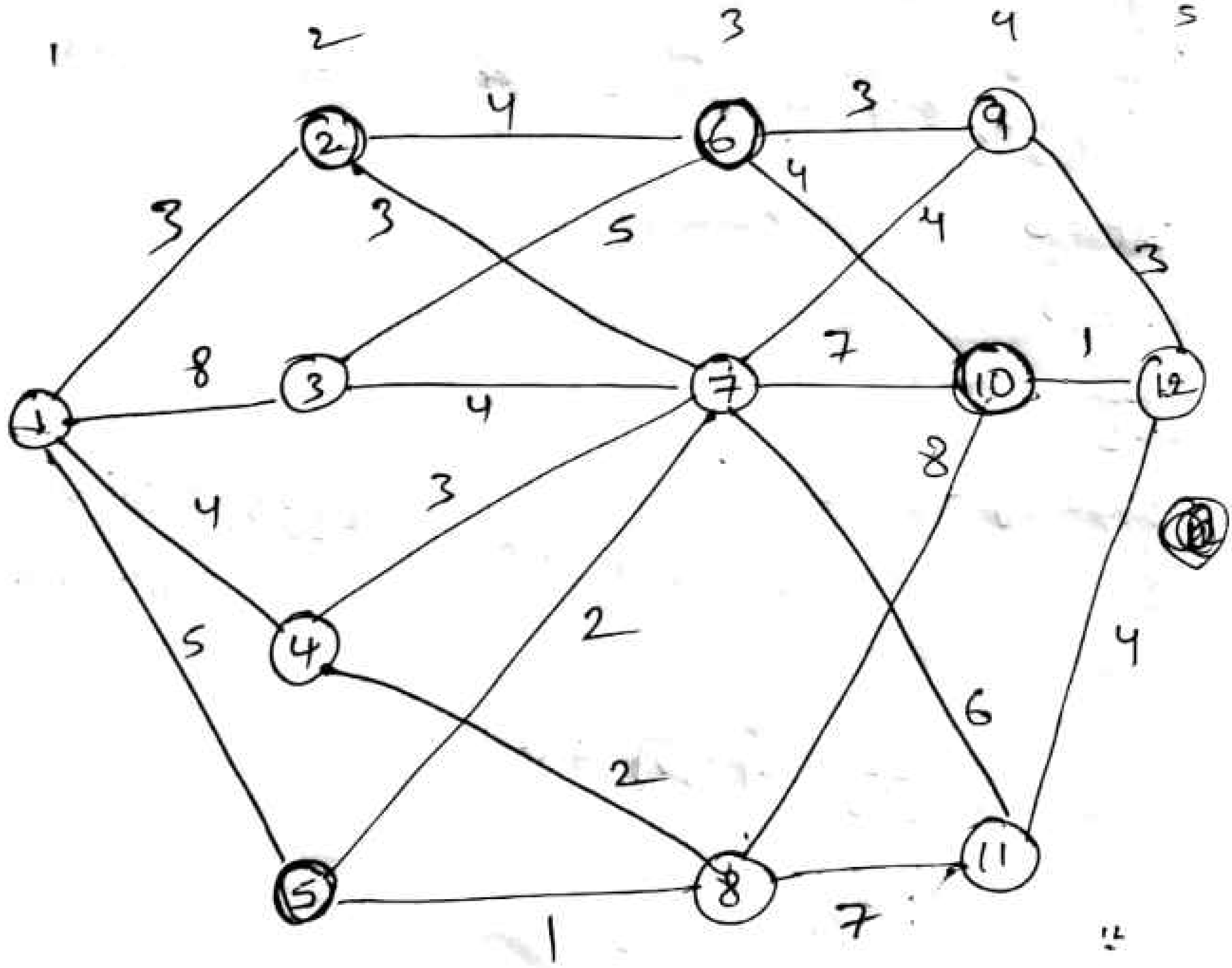
dist[j] = v

}

return cost[j]

}

Ex:- Find the shortest path between node 1 and 12, and also evaluate the cost of finding the shortest path.



Sol:
$Cost(5,12) = 0$ in stage 5

In stage 4

$Cost(4,9) = 3 + Cost(5,12) = 3+0 = 3$

$Cost(4,10) = 1 + Cost(5,12) = 1+0 = 1$

$Cost(4,11) = 4 + Cost(5,12) = 4+0 = 4$

$\therefore$ minimum Cost is 1

In stage 3

$$Cost(3,6) = \min \begin{cases} 3 + Cost(4,9) \\ 4 + Cost(4,10) \end{cases}$$

$$= \min \begin{cases} 3+3 \\ 4+1 \end{cases} = \min \begin{cases} 6 \\ 5 \checkmark \end{cases}$$

∴ The minimum cost is 5 from edge cost (3,6)

$$Cost(3,7) = \min \begin{cases} 4 + cost(4,9) \\ 7 + cost(4,10) \\ 6 + cost(4,11) \end{cases}$$

$$= \min \begin{cases} 4+3 = 7 \\ 7+1 = 8 \checkmark \\ 6+4 = 10 \end{cases}$$

∴ The minimum cost is 7 from edge cost (3,7)

$$Cost(3,8) = \min \begin{cases} 8 + Cost(4,10) \\ 7 + Cost(4,11) \end{cases} = \min \begin{cases} 8+1 = 9 \\ 7+4 = 11 \end{cases}$$

∴ The minimum cost is 9 from edge cost (3,8)

In Stage 2

$$Cost(2,2) = \min \begin{cases} 4 + Cost(3,6) = 4+5 = 9 \leftarrow \min \\ 3 + Cost(3,7) = 3+7 = 10 \end{cases}$$

$$Cost(2,3) = \min \begin{cases} 5 + Cost(3,6) = 5+5 = 10 \leftarrow \min \\ 4 + Cost(3,7) = 4+7 = 11 \end{cases}$$

$$Cost(2,4) = \min \begin{cases} 3 + Cost(3,7) = 3+7 = 10 \leftarrow \min \\ 2 + Cost(3,8) = 2+9 = 11 \end{cases}$$

$$\text{Cost}(2,5) = \min \begin{cases} 2 + \text{cost}(3,7) = 2+7 = 9 \leftarrow \min \\ 1 + \text{cost}(3,8) = 1+9 = 10 \end{cases}$$

In Stage 1

$$\text{Cost}(1,1) = \min \begin{cases} 3 + \text{Cost}(2,2) \\ 8 + \text{Cost}(2,3) \\ 4 + \text{Cost}(2,4) \\ 5 + \text{Cost}(2,5) \end{cases} = \min \begin{cases} 3+9 = 12 \leftarrow \min \\ 8+10 = 18 \\ 4+10 = 14 \\ 5+9 = 14 \end{cases}$$

The minimum cost is 12. The shortest path from node 1 to node 12 is having a cost of 12.

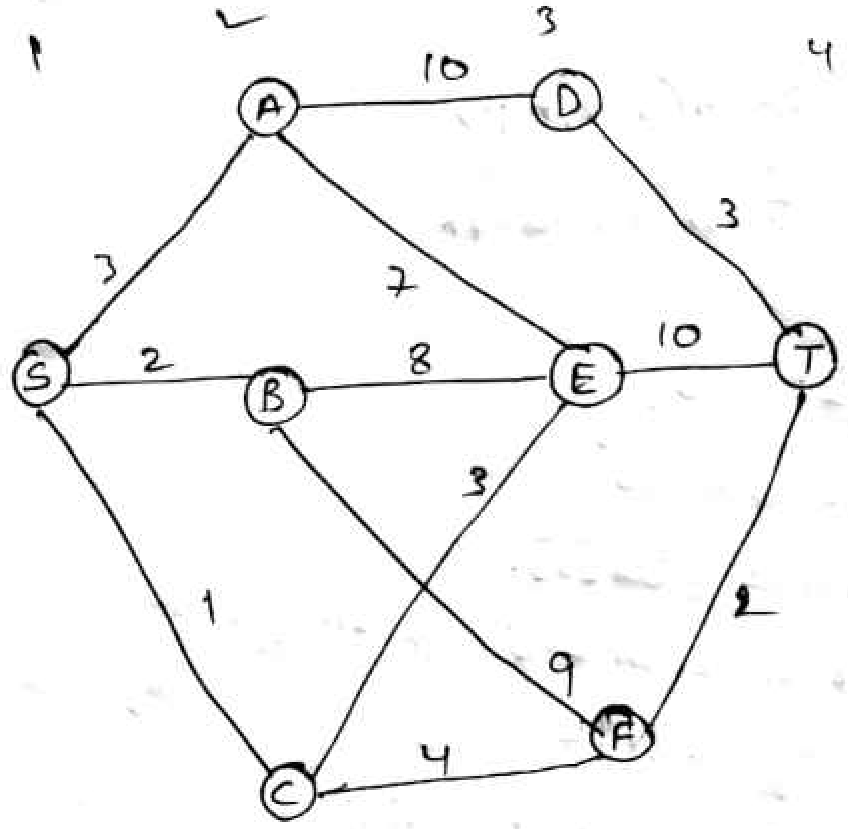
And the path is

$$1 \rightarrow 2 \rightarrow 6 \rightarrow 10 \rightarrow 12$$

Backward Computation procedure

The multistage graph problem can also be solved using the backward computation procedure. This method is same as the forward method but differs in only one aspect. That is, the loop tracks from 1 to n-1. Thus, the distance computation in the algorithm for the backward method is from the source.

Ex:-



Sol:-

In stage 1-2

$Cost(S, A) = 3$

$Cost(S, B) = 2$

$Cost(S, C) = 1$

In stage 1-3

$$Cost(S, D) = \min \begin{cases} Cost(S, A) + 10 = 3 + 10 = 13 \end{cases}$$

$$Cost(S, E) = \min \begin{cases} Cost(S, A) + 7 = 3 + 7 = 10 \\ Cost(S, B) + 8 = 2 + 8 = 10 \\ Cost(S, C) + 3 = 1 + 3 = 4 \leftarrow \text{min} \end{cases}$$

$$Cost(S, F) = \min \begin{cases} Cost(S, B) + 9 = 2 + 9 = 11 \\ Cost(S, C) + 4 = 1 + 4 = 5 \leftarrow \text{min} \end{cases}$$

In stage 1-4

$$Cost(S, T) = \min \begin{cases} Cost(S, D) + 3 = 13 + 3 = 16 \\ Cost(S, E) + 10 = 4 + 10 = 14 \\ Cost(S, F) + 2 = 5 + 2 = 7 \text{ min} \end{cases}$$

Path: $S \rightarrow C \rightarrow F \rightarrow T$.

12, 14, 21, 27, 30, 37, 39, 40, 55

```
Algorithm BGraph(G, k, n, p)
//Same function as FGraph
{
  bcost[1] := 0.0;
  for j=2 to n do
  { //compute bcost[j].
    Let r be such (r,j) is an edge of
    G and 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]];
  }
}
```

Algorithm FGraph(G, k, n, P)

// The input is a k-stage graph G = (V, E) with n vertices

// 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 a vertex such that (j, r) is an edge of
    G and 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]];
  }
}
```

Complexity

The graph $G=(V,E)$ is given in the adjacency list. Let $n=|V|$ and $m=|E|$. There is only one for-loop that gets executed $n-1$ times. The path tracking requires $O(k)$ times, where k is the number of stages. Therefore, the Complexity of the algorithm is $O(n+m)$.

Topic: All pairs shortest Paths Algorithm

The all pairs shortest path problem can be considered the mother of all routing problems. It aims to compute the shortest path from each vertex v to every other u, using standard single source algorithms.

All pairs shortest path algorithm is also called Floyd-warshall Algorithm.

problem Def:- Let $G=(V,E)$ be a directed graph consisting of n-vertices and each edge is associated with a weight. The problem of finding the shortest path b/w all pairs of vertices in a graph is called All pair shortest path.

It can be computed using the following.

$$\cancel{A^k(i,j) = w(i,j), \text{ if } k=0}$$

$$A^k(i,j) = \min\left\{A^{k-1}(i,j),\ A^{k-1}(i,k) + A^{k-1}(k,j)\right\}$$

where $A^k(i,j)$ represents the cost of ~~adjacency~~ shortest path ~~matrix of a graph with k^{th} vertices.~~ from vertex i to vertex j.

Informal Algorithm

step-1 Read weighted graph $G=(V,E)$

step-2: Initialize $A[i,j]$ as follows:

$$A[i,j] = \begin{cases} 0 & \text{if } i=j \\ \infty & \text{if edge}(i,j) \notin E(G) \\ w_{ij} & \text{if edge}(i,j) \in E(G) \end{cases}$$

step-3:- For intermediate nodes k, $1 \le k \le n$, recursively Compute $A^k[i,j] = \min\limits_{1 \le k \le n}\left\{A^{k-1}[i,j],\ A^{k-1}[i,k] + A^{k-1}[k,j]\right\}$

step-4: Return Matrix A^k

step-5: End

Algorithm

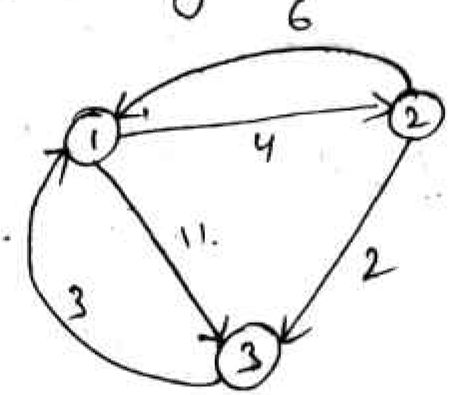
```
Algorithm shortest path(w,A,n)
// w is weighted Array matrix
// n is the no. of vertices
// A is the cost of shortest path from vertex i to j
{
 for i <-1 to n do
  {
   for j <-1 to n do
    {
     A[i,j] <-- w[i,j];   //if k=0
    }
  }
 for k <-1 to n do
  {
   for i <-1 to n do
    {
     for j <-1 to n do
      {
       A[i,j] <-- min(A[i,j], A[i,k] + A[k,j]);
      }
    }
  }
```

Analysis:-

Since, the loop iterate three times. The time complexity for this algorithm is $O(n^3)$.

Ex:-

Consider a digraph



Find the all pairs shortest path.

Sol:-

The Initial matrix

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

By Consider A^0 finding A^1

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

$A^1[2,3] = \min\{A^0[2,3], A^0[2,1] + A^0[1,3]\}$

$= \min\{2, 6+11\}$

$= 2$

$A^1[3,2] = \min\{A^0[3,2], A^0[3,1] + A^0[1,2]\}$

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

$= \min\{\infty, 7\}$

$= 7$

By Considering A^1 finding A^2

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

$A^2[1,3] = \min\{A^1[1,3], A^1[1,2] + A^1[2,3]\}$

$= \min\{11, 4+2\}$

$= \min\{11, 6\}$

$= 6$

$A^2[3,1] = \min\{A^1[3,1], A^1[3,2] + A^1[2,1]\}$

$= \min\{3, 7+6\}$

$= \min\{3, 13\}$

$= 3$

By Considering A^2 finding A^3.

$$A^3 = \begin{array}{c|ccc} & 1 & 2 & 3 \\ \hline 1 & 0 & \boxed{4} & 6 \\ 2 & \boxed{5} & 0 & 2 \\ 3 & 3 & 7 & 0 \end{array}$$

$A^3[1,2] = \min\{A^2[1,2], A^2[1,3] + A^2[3,2]\}$

$= \min\{4, 6+7\}$

$= \min\{4, 13\}$

$= 4$

$A^3[2,1] = \min\{A^2[2,1], A^2[2,3] + A^2[3,1]\}$

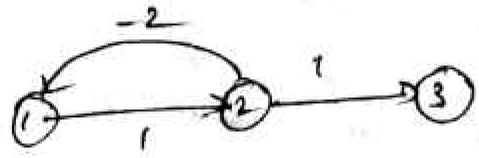
$= \min\{6, 2+3\}$

$= \min\{6, 5\}$

$= 5$

We stop at A^3, because, the no. of nodes or vertices is 3.

Ex:



Sol:

$A^0 =$

	1	2	3
1	0	1	∞
2	-2	0	1
3	∞	∞	0

$A^1 =$

	1	2	3
1	0	1	∞
2	-2	0	1
3	∞		0

$A^1[2,3] = \min\{A^0[2,3], A^0[2,1] + A^0[1,3]\}$

$= \min\{1, -2+\infty\}$

$= 1$

$A^1[3,2] = \min\{A^0[3,2], A^0[3,1] + A^0[1,2]\}$

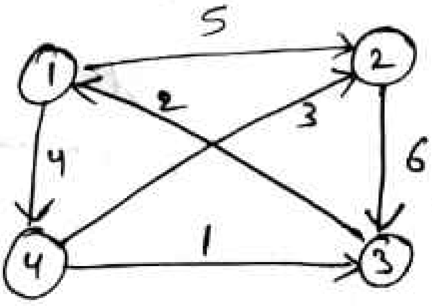
$= \min\{\infty, \infty + 1\}$

$= \infty$

~~$A^1[3,2] = \min\{A^1[3,2]\}$~~

∴ All pairs shortest path (Floyd-warshall algorithm) do not accept for negative lengths.

Ex:



$A^0 =$

	1	2	3	4
1	0	5	∞	4
2	∞	0	6	∞
3	2	∞	0	∞
4	∞	3	1	0

$A^4 =$

	1	2	3	4
1	0	5	5	4
2	8	0	6	12
3	2	7	0	6
4	3	3	1	0

Bellman-Ford Algorithm (Single source Shortest Path)

→ Single source shortest path problem in dynamic programming is used to solved by using Bellman & Ford algorithm.

→ many shortest path problems doesnot allow negative cost, But where as Bellman ford algorithm allows negative.

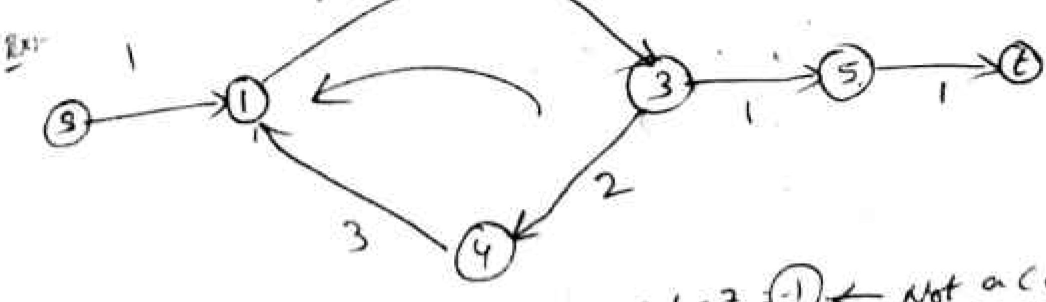
→ To solve single source shortest path problem in dynamic programming. Bellman-Ford introduced Relaxiation Rule.

if $d(u) + cost(u,v) < d(v)$ then.

change as $d(v) = d(u) + cost(u,v)$

→ Bellman-Ford algorithm can handle negative edges, but not a negative cycle (a set of edges such that the sum of their weights is negative).

Ex:-



$1-7+3+2 = 6-7 = -1$ ← Not accepted.

→ The initialization process of the algorithm is as follows:

$d(s) = 0$

for all other vertices v, $v \neq s$

$d[v] = \infty$

After initialization, every edge is considered for relaxation. Relaxation means to reduce the upper bound of the edge of the shortest path till the upper bound of the edges is reduced to the length of the actual shortest path.

Informal procedure:

step-1 choose the source vertex and label it as '0'.

step-2: Label all vertices except the source as '∞'

step-3: Repeat n-1 times, where n is the number of vertices.

i. If label of 'v' is larger than that of u + cost of the edge(u,v) then relax the edge.

ii. update the label of v as the label of u + cost of the edge(u,v).

step-4: check the presence of any negative edge cycle by repeating the iteration and carry out the procedure if any edges still relax. If so, report the presence of a negative weight cycle.

Algorithm Bellman(G.w, s)

// weighted Graph G, w is the weight and s is the source

{

d(s) = 0

for all over vertices v, v ≠ s

d[v] = ∞

N = |v(G)|

Repeat n-1 times

{
for each edge (u,v) ∈ E(G) do

{

if (dist(u) + Cost(u,v) < dist(v))

dist(v) = dist(u) + Cost(u,v);

}

for each edge (u,v) ∈ E(G) do

{ if (d(v) > d[u] + Cost(u,v)

print 'negative edge cycle is present';

}

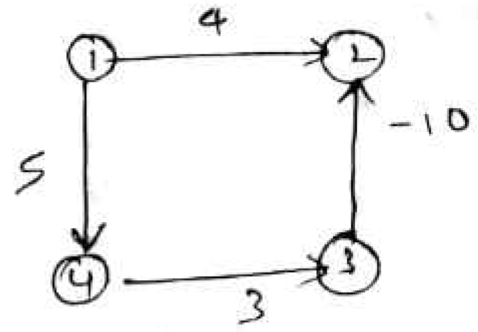
}

}

Complexity

The algorithm takes $O(nm)$ time, where n is the no. of vertices in graph G and m is the no. of vertices in graph G. Thus, the Complexity of the algorithm would be $O(n^2)$.

Ex: solve using Dynamic programming



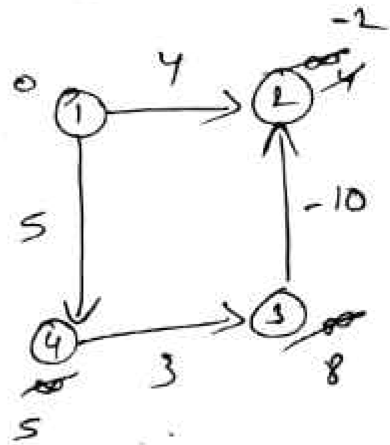
Sol: Initially the distance of source vertex is 0. and remaining vertices is ∞

Now from edges list (1,2)(1,4)(4,3)(3,2)

Now apply relaxation rule

if $d(u) + cost(u,v) < d(v)$

$d(v) = d(u) + cost(u,v)$



For 1st vertices

$d(1) + cost(1,2) < d(2)$

$0 + 4 < \infty$

$4 < \infty$ — T

so, $d(2) = 4$

$d(1) + cost(1,4) < d(4)$

$0 + 5 < \infty$ — T

so, $d(4) = 5$

For 2nd vertices No out goings

For 4th vertices

$d(4) + cost(4,3) < d(3)$

$5 + 3 < \infty$

$8 < \infty$

so, $d(4) = 8$

For 3rd vertices

$d(3) + cost(3,2) < d(2)$

$8 + -10 < 4$

$-2 < 4$ — T

so $d(2) = -2$

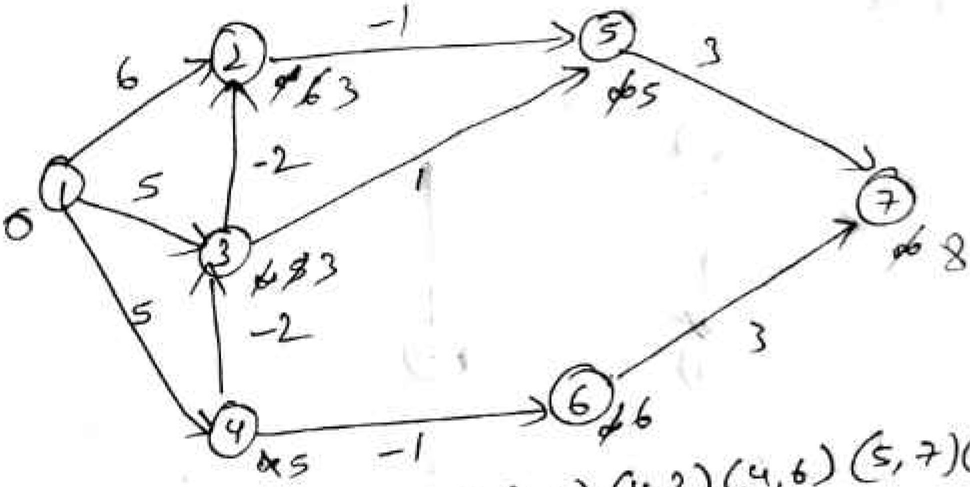
$0 + 4 < -2$

$4 < -2$ — False

$d(v) > d(u) + cost(u,v)$

negative edge cycle is present.

Ex:-



Sol: (1,2) (1,3) (1,4) (2,5) (3,5) (3,2) (4,3) (4,6) (5,7) (6,7)

(1) $d(1) + cost(1,2) < d(2)$

$0 + 6 < \infty$ — T

so $d(2) = 6$

$d(1) + cost(1,3) < d(3)$

$0 + 5 < \infty$ — T

so $d(3) = 5$

Ily $d(4) = 5$

$d(2) + cost(2,5) < d(5)$

$6 + (-1) < \infty$ — T

$5 < \infty$

so, $d(5) = 5$

$d(3) + cost(3,5) < d(5)$

$5 + 1 < 5$

$6 < 5$ — F

$d(3) + cost(3,2) < d(2)$

$5 - 2 < 6$

$3 < 6$ — T

so $d(2) = 3$

$d(4) + cost(4,3) < d(3)$

$5 + -2 < 5$

$3 < 5$ — T

so, $d(3) = 3$

$d(4) + cost(4,6) < d(6)$

$5 + (-1) < \infty$

$4 < \infty$ — T

so, $d(6) = 4$

$d(5) + cost(5,7) < d(7)$

$5 + 3 < \infty$ — T

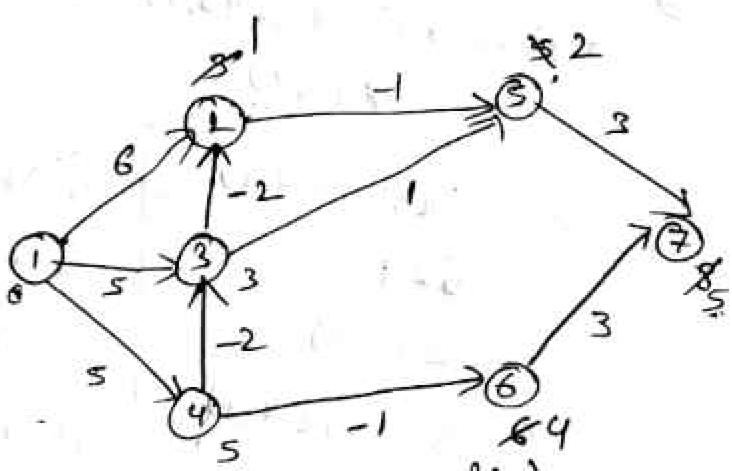
so, $d(7) = 8$

$d(6) + cost(6,7) < d(7)$

$6 + 3 < 8$

$9 < 8$ — F

$d(1) = 0$
$d(2) = 3$
$d(3) = 3$
$d(4) = 5$
$d(5) = 5$
$d(6) = 6$
$d(7) = 8$



② $d(1) + cost(1,2) < d(2)$

$0 + 6 < 3$

$6 < 3$ — F

$d(1) + cost(1,3) < d(3)$

$0 + 5 < 3$

$5 < 3$ — F

$d(1) + cost(1,4) < d(4)$

$0 + 5 < 5$ — F

$d(2) + cost(2,5) < d(5)$

$3 + (-1) < 5$

$2 < 5$

so, $d(5) = 2$

$d(3) + cost(3,5) < d(5)$

$3 + 1 < 3$

$4 < 3$ — F

$d(3) + cost(3,2) < d(2)$

$3 + (-2) < 3$

$1 < 3$ (T)

so, $d(2) = 1$

$d(4) + cost(4,3) < d(3)$

$5 + -2 < 3$

$3 < 3$ (F)

$d(4) + cost(4,6) < d(6)$

$5 + -1 < 6$

$4 < 6$ — T

$d(6) \leftarrow 4$

$d(5) + cost(5,7) < d(7)$

$2 + 3 < 8$

$5 < 8$ (T)

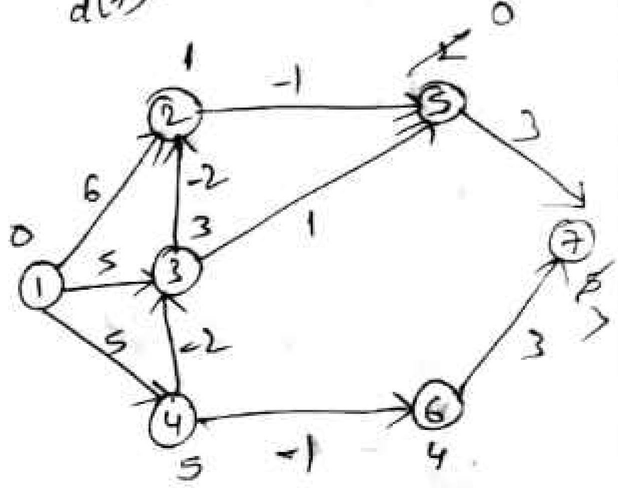
so $d(7) = 5$

$d(6) + cost(6,7) < d(7)$

$4 + 3 < 5$

$7 < 5$ (F)

$d(1) = 0$
$d(2) = 1$
$d(3) = 3$
$d(4) = 5$
$d(5) = 2$
$d(6) = 4$
$d(7) = 5$



(1,2)(1,3)(1,4)(2,5)(3,2)(3,5)
(4,3)(4,6)(5,7)(6,7)

$d(1) + cost(1,2) < d(2)$

$0 + 6 < 1$

$6 < 1$ (F)

$d(1) + cost(1,3) < d(3)$

$0 + 5 < 3$

$5 < 3$ (F)

$d(1) + cost(1,4) < d(4)$

$0 + 5 < 5$

$5 < 5$ (F)

$d(2) + cost(2,5) < d(5)$

$1 + -1 < 2$

$0 < 2$ (T)

$d(5) = 0$

$d(3) + cost(3,2) < d(2)$

$3 + -2 < 1$

$1 < 1$ (F)

$d(3) + cost(3,5) < d(5)$

$3 + 1 < 0$ (F)

$d(4) + cost(4,3) < d(3)$

$5 + -2 < 3$

$3 < 3$ (F)

$d(4) + cost(4,6) < d(6)$

$5 + -1 < 4$

$4 < 4$ (F)

$d(5) + cost(5,7) < d(7)$

$0 + 3 < 5$ (T)

$d(7) = 3$

$d(6) + cost(6,7) < d(7)$

$4 + 3 < 3$ (F)

$d(1) = 0$

$d(2) = 1$

$d(3) = 3$

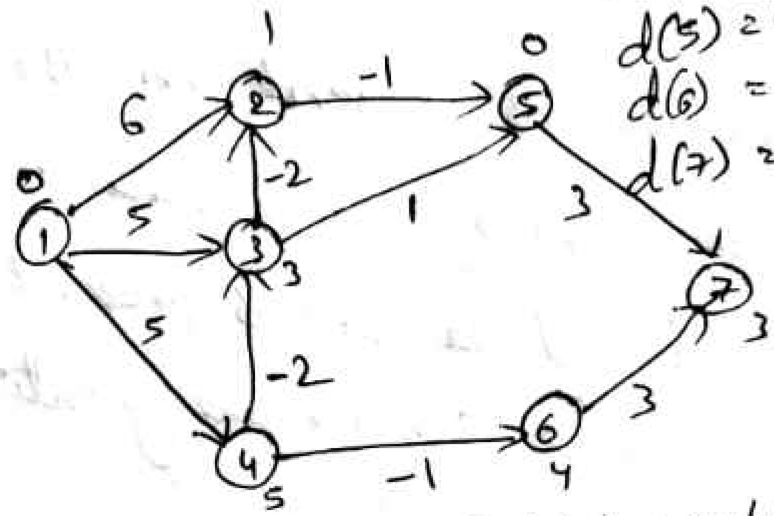
$d(4) = 5$

$d(5) = 0$

$d(6) = 4$

$d(7) = 3$

(4)



$d(1) + cost(1,2) < d(2)$

$0 + 6 < 1$ (F)

$d(1) + cost(1,3) < d(3)$

$0 + 5 < 3$ (F)

$d(1) + cost(1,4) < d(4)$

$0 + 5 < 5$ (F)

$d(2) + cost(2,5) < d(5)$

$1 + -1 < 0$ (F)

$d(3) + cost(3,2) < d(2)$

$3 + -2 < 1$ (F)

$d(3) + cost(3,5) < d(5)$

$3 + 1 < 0$ (F)

$d(4) + cost(4,3) < d(3)$

$5 + -2 < 3$ (F)

$d(4) + cost(4,6) < d(6)$

$5 + -1 < 4$ (F)

$d(5) + cost(5,7) < d(7)$

$0 + 3 < 3$ (F)

$d(6) + cost(6,7) < d(7)$

$4 + 3 < 3$ (F)

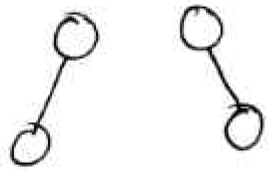
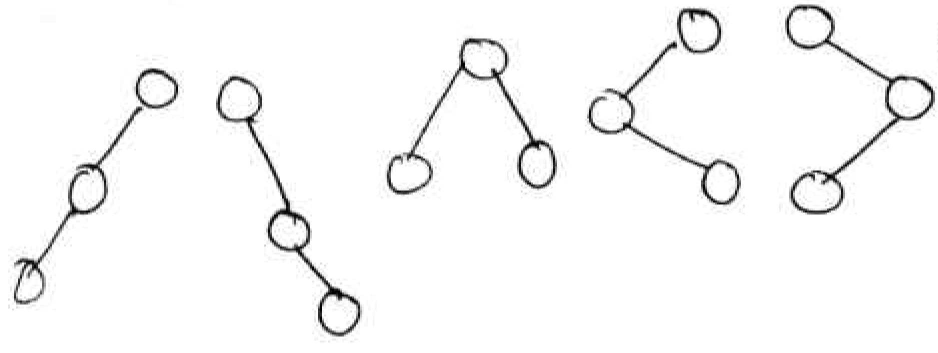
lly case (5) and (6) No change

Note:- we can find shortest path's from every vertices to other vertices in graph

Optimal Binary Search Tree (OBST)

Binary search tree: The values present in the left subtree are less than root value and the values present in the right subtree are greater than the root value.

→ For a given set of identifies, we can construct more than one binary search tree

No. of nodes (or) Identifiers	Possible tree structure	Possible no. of trees
0	Empty tree	1
1		1
2		2
3		5

→ It Can be observed that the no of Possible resulting trees is a Catalan number, The n^{th} Catalan number C_n is given as

$$C_n = \frac{1}{n+1} {}^{2n}C_n \quad \text{for } n \geq 0$$

Ex:- if n=3 (no. of nodes), then $C_3 = \frac{1}{3+1} {}^{2(3)}C_3$

$$= \frac{6C_3}{4}$$

$$\frac{n!}{(n-r)!\,r!}$$

$$= \frac{6!}{3!\,3! \times 4}$$

$$= \frac{6 \times 5 \times \cancel{4!}}{3! \times \cancel{4!}}$$

$$= \frac{\cancel{6} \times 5}{\cancel{3} \times \cancel{2}}$$

$$= 5.$$

→ Each Binary search trees exhibit different Performance characteristics.

→ In general situation, we expect different identifiers to be searched for with different frequencies (probabilities)

→ There may be unsuccessful search cases also.

→ Let $\{a_1, a_2, \ldots, a_n\}$ with $a_1 < a_2 < \ldots < a_n$. Let $p(i)$ be the probability with which each a_i is searched.

→ Then $\Sigma p(i)$ where $1 \le i \le n$ is the probability of successful search.

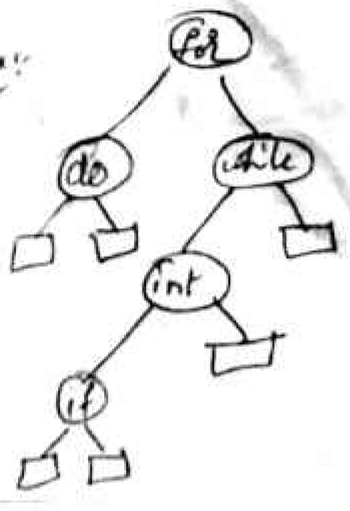
→ Let $q(i)$ be the probability that the identifier x is ~~search space~~ searched such that $a_i < x < a_{i+1}$.

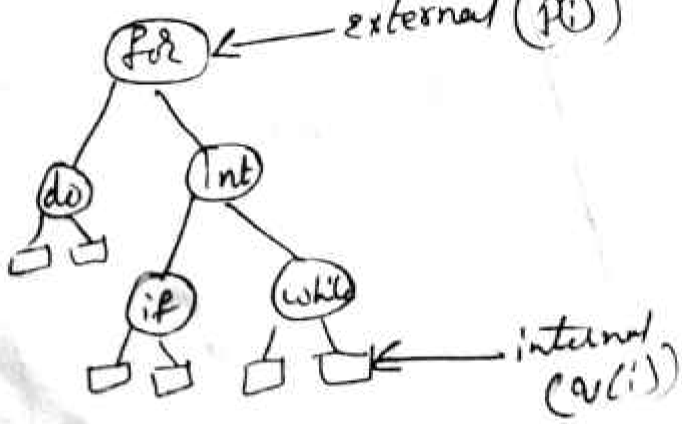
→ Then $\Sigma q(i)$ where $0 \le i \le n$ is the probability of unsuccessful search.

$$\sum_{1 \le i \le n} p(i) + \sum_{0 \le i \le n} q(i) = 1$$

→ To obtain a cost function for binary search tree, it is useful to add a fictitious node in place of empty subtree in the search tree.

Ex:





→ Successful search terminate at level I, unsuccessful search terminates at node level-1.

→ The formula for the expected cost of the binary search tree is

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

Equal probability

Ex: The possible binary search trees for the identifier set $\{a_1, a_2, a_3\} = (do, if, while)$. with equal probabilities $p(i) = q(i) = (1/7)$. Find the optimal

Solution

Sol: Here the no. of identifiers are 3 $(n=3)$

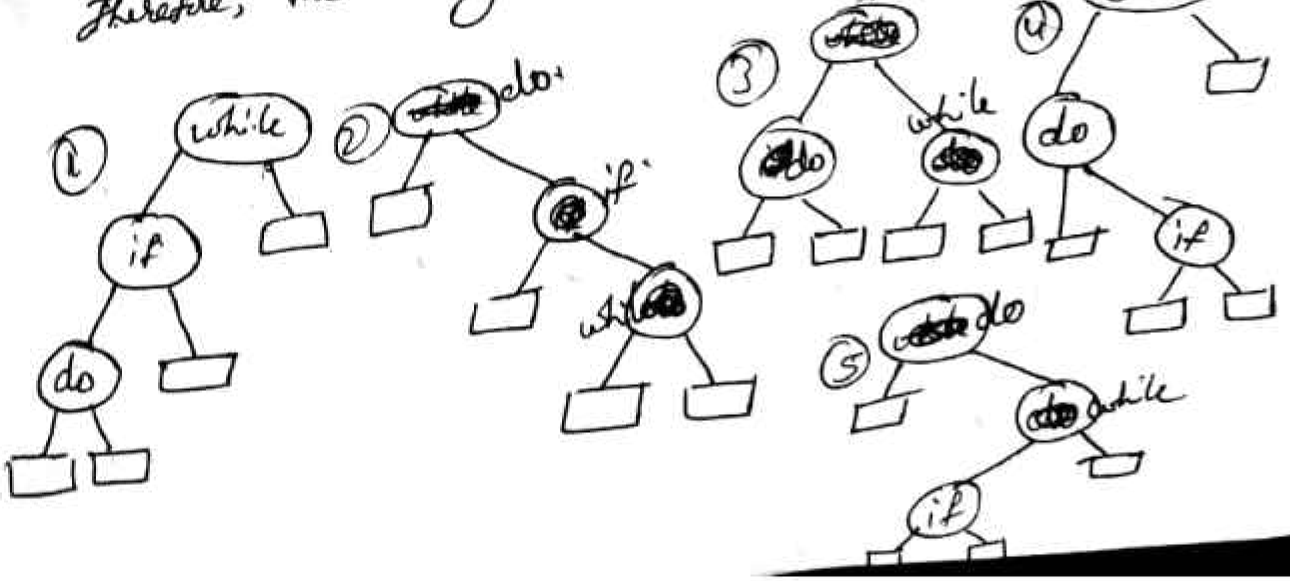
$$\therefore C_3 = \frac{1}{3+1}\, {}^{2(3)}C_3 = \frac{1}{4} \times {}^{6}C_3$$

$$= \frac{1}{4} \times \frac{6!}{(6-3)!\,3!}$$

$$= \frac{1}{4} \times \frac{6\times5\times4!}{3!\,3\times2\times1}$$

$$= 5$$

Therefore, the binary search trees formed are 5.



Applying cost of binary search Tree

Ans (1) Cost (tree 1)

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

$$= \frac{1}{7} \times 1 + \frac{1}{7} \times 2 + \frac{1}{7} \times 3 + \frac{1}{7} \times 1 + \frac{1}{7} \times 2 + \frac{1}{7} \times 3 + \frac{1}{7} \times 3$$

$$= 15/7$$

Cost (tree 2)

$$= \frac{1}{7} \times 1 + \frac{1}{7} \times 2 + \frac{1}{7} \times 3 + \frac{1}{7} \times 1 + \frac{1}{7} \times 2 + \frac{1}{7} \times 3 + \frac{1}{7} \times 3$$

$$= 15/7$$

Cost (tree 3)

$$= \frac{1}{7} \times 1 + \frac{1}{7} \times 2 + \frac{1}{7} \times 2 + \frac{1}{7} \times 2 + \frac{1}{7} \times 2 + \frac{1}{7} \times 2 + \frac{1}{7} \times 2$$

$$= 13/7$$

Cost (tree 4)

$$= \frac{1}{7} \times 1 + \frac{1}{7} \times 2 + \frac{1}{7} \times 3 + \frac{1}{7} \times 1 + \frac{1}{7} \times 2 + \frac{1}{7} \times 3 + \frac{1}{7} \times 3$$

$$= 15/7$$

Cost (tree 5)

$$= \frac{1}{7} \times 1 + \frac{1}{7} \times 2 + \frac{1}{7} \times 3 + \frac{1}{7} \times 1 + \frac{1}{7} \times 2 + \frac{1}{7} \times 3 + \frac{1}{7} \times 3$$

$$= 15/7$$

Since, tree3 is having the less value, it is the optimal solution.

unequal probability

→ The binary search tree for the identifier set $\{a_1, a_2, a_3\}$ = (do, if, while). with unequal probabilities

$a_1 > a_2 > a_3$

$p(1) = 0.5$ ✓ $\quad q(0) = 0.15$ ✓

$p(2) = 0.1$ ✓ $\quad q(1) = 0.1$ ✓

$p(3) = 0.05$ ✓ $\quad q(2) = 0.05$ ✓

$\quad q(3) = 0.05$ ✓

soln

← Same as above problem, By Cost of binary search tree are.

Cost(tree 1)

$= 0.05 \times 1 + 0.1 \times 2 + 0.5 \times 3 + 0.15 \times 3 + 0.1 \times 2 + 0.05 \times 3 + 0.05 \times 3$

$= 2.4$

Cost (tree 2)

$= 0.5 \times 1 + 0.1 \times 2 + 0.05 \times 3 + 0.15 \times 1 + 0.1 \times 2 + 0.05 \times 3 + 0.05 \times 3$

$= 1.5$

Cost(tree 3)

$= 0.1\times1 + 0.5\times2 + 0.05\times2 + 0.15\times2 + 0.1\times2 + 0.05\times3 + 0.05\times2$

$= 1.9$

Cost(tree 4)

$= 0.05\times1 + 0.5\times2 + 0.1\times3 + 0.15\times2 + 0.1\times2 + 0.05\times3 + 0.05\times3$

$= 2$

Cost(tree 5)

$= 0.5\times1 + 0.05\times2 + 0.1\times3 + 0.15\times2 + 0.1\times2 + 0.05\times3 + 0.05\times3$

$= 1.6$

Since, tree 2 is having the less Value, it is the optimal solution.

Note: In BST ~~can't~~ finding the optimal solution with root a_k or where $n>3$ is Complex. Because if $n>3$, for instance $n=4$ then the possible trees are

$$= \frac{1}{4+1} \, {}^{2(4)}C_4$$

$$= \frac{1}{5} \times {}^{8}C_4$$

$$= \frac{1}{5} \times \frac{8!}{4!\,4!}$$

$$= \frac{8\times7\times6}{4\times3\times2\times1} = 14.$$

Therefore, we go for OBST using Dynamic programming

OBST formula — An optimal binary search tree is a BST for which the nodes are arranged on levels such that the tree cost is minimum.

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

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

$$r(i,j) = k \text{ that minimizes } c(i,j)$$

Initially,

$w(i,i) = q(i)$

$c(i,i) = 0$

$r(i,i) = 0$

Sub the cost pair in $T_{ij} = T_{i,k-1}, T_{k,j}$ where k = rank of cost pair

Ex: Let $n=4$ and $(a_1, a_2, a_3, a_4) = (do, if, int, while)$.

$p(1:4) = (3,3,1,1)$ and $q(0:4) = (2,3,1,1,1)$

Sol: Since $n=4$

The no of possible trees are $C_4 = \frac{1}{4+1} \, {}^{2(4)}C_4$

$= \frac{1}{5} \times \frac{8!}{4!\,4!}$

$= \frac{8 \times \cancel{7} \times \cancel{6}^{2}}{\cancel{4} \times \cancel{3} \times \cancel{2}}$

$= 14$

Initially $w(i,i) = q(i)$ // weight

$c(i,i) = 0$ // cost

$r(i,i) = 0$ // rank

weight

$w(0,0) = q(0) = 2$

$w(1,1) = q(1) = 3$

$w(2,2) = q(2) = 1$

$w(3,3) = q(3) = 1$

$w(4,4) = q(4) = 1$

Cost

$c(0,0) = 0$

$c(1,1) = 0$

$c(2,2) = 0$

$c(3,3) = 0$

$c(4,4) = 0$

rank

$r(0,0) = 0$

$r(1,1) = 0$

$r(2,2) = 0$

$r(3,3) = 0$

$r(4,4) = 0$

Since the ~~external~~ internal nodes are 4, 4 cases are possible.

i.e., $j - i = 1$

$j - i = 2$

$j - i = 3$

$j - 1 = 4$

Case 1: $j - i = 1$ $(0,1)\ (1,2)\ (2,3)\ (3,4)$

$1 - 0$
$2 - 1$
$3 - 2$
$4 - 3$

weight

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

$w(0,1) = p(1) + q(1) + w(0,0)$
$= 3 + 3 + 2 = 8$

$w(1,2) = p(2) + q(2) + w(1,1)$
$= 3 + 1 + 3 = 7$

$w(2,3) = p(3) + q(3) + w(2,2)$
$= 1 + 1 + 1 = 3$

$w(3,4) = p(4) + q(4) + w(3,3)$

$= 1 + 1 + 1$

$= 3$

Cost:

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

$$c(0,1) = \min_{0<k\le 1} \{c(0,1-1) + c(1,1)\} + w(0,1) \quad k=1$$

$= \min\{c(0,0) + c(1,1)\} + w(0,1)$

$= \min\{0+0\} + 8$

$= 8$

Root

$R(0,1) = 1$ since $k=1$

$$c(1,2) = \min_{1<k\le 2} \{c(1,2-1) + c(2,2)\} + w(1,2) \quad k=2$$

$= \min\{c(1,1) + c(2,2)\} + w(1,2)$

$= 0 + 0 + 7$

$= 7$

$R(1,2) = 2$ since $k=2$

$$C(2,3) = \min_{2<k\leq 3} \{c(2,3-1) + c(3,3)\} + \omega(2,3)$$
$k=3$

$$= \min\{c(2,2) + c(3,3)\} + \omega(2,3)$$

$$= \min\{0+0\} + 3$$

$$= 3$$

$R(2,3) = 3$ since $k = 3$

$$C(3,4) = \min_{3<k\leq 4} \{c(3,4-1) + c(4,4)\} + \omega(3,4)$$
$k=4$

$$= \min\{c(3,3) + c(4,4)\} + \omega(3,4)$$

$$= \min\{0+0\} + 3$$

$$= 3$$

$R(2,3) = 4$ since $k = 4$

<u>Case 2</u> :- $j - i = 2$

$2 - 0$ $(0,2)$ $(1,3)$ $(2,4)$

$3 - 1$

$4 - 2$

$$\omega(0,2) = p(2) + q(2) + \omega(0, 2-1)$$

$$= 3 + 1 + \omega(0,1)$$

$$= 3 + 1 + 8$$

$$= 12$$

$$\omega(1,3) = p(3) + q(3) + \omega(1, 3-1)$$

$$= 1 + 1 + \omega(1,2)$$

$$= 2 + 7$$

$$= 9$$

$$\omega(2,4) = p(4) + p(4) + \omega(2, 4-1)$$

$$= 1 + 1 + \omega(2,3)$$

$$= 2 + 3 = 5$$

Cost

$$c(0,2) = \min_{0 < k \leq 2} \left\{ \underset{k=1}{c(0,1-1) + c(1,2)}, \underset{k=2}{c(0,2-1) + c(2,2)} \right\} + w(0,2)$$

$$= \min \left\{ c(0,0) + c(1,2), c(0,1) + c(2,2) \right\} + w(0,2)$$

$$= \min \left\{ \underset{k=1}{0+7}, 8+0 \right\} + 12$$

$$= 7+12 = 17$$

$r(0,2) = 1$ since $k=1$

$$\text{Cost}(1,3) = \min_{1 < k \leq 3} \left\{ \underset{k=2}{c(1,2-1) + c(2,3)}, \underset{k=3}{c(1,3-1) + c(3,3)} \right\} + w(1,3)$$

$$= \min \left\{ c(1,1) + c(2,3), c(1,2) + c(3,3) \right\} + w(1,3)$$

$$= \min \left\{ 0+3, 7+0 \right\} + 9$$

$$= 3+9 = 12$$

$r(1,3) = 2$ since $k=2$

$$\text{Cost}(2,4) = \min_{2 < k \leq 4} \left\{ \underset{k=3}{c(2,3-1) + c(3,4)}, \underset{k=4}{c(2,4-1) + c(4,4)} \right\} + w(2,4)$$

$$= \min \left\{ c(2,2) + c(3,4), c(2,3) + c(4,4) \right\} + 5$$

$$= \min \left\{ 0+3, 3+0 \right\} + 5$$

$$= 8$$

$R(2,4) = 3$ since $k=3$ are minimum

Gx 3: $j - i = 3$ $(0,3)\ (1,4)$

$3 - 0$

$4 - 1$

$$w(0,3) = p(3) + q(3) + w(0, 3-1)$$

$$= 1 + 1 + w(0,2)$$

$$= 1 + 1 + 12$$

$$= 14$$

$$w(1,4) = p(4) + q(4) + w(1, 4-1)$$

$$= 1 + 1 + w(1,3)$$

$$= 1 + 1 + 9$$

$$= 11$$

Cost

$$c(0,3) = \min_{0 < k \le 3} \left\{ \begin{array}{l} \underset{k=1}{c(0,1-1) + c(1,3)}, \\ \underset{k=2}{c(0,2-1) + (2,3)} \\ \underset{k=3}{c(0,3-1) + c(3,3)} \end{array} \right\} + w(0,3)$$

$$= \min\{c(0,0) + c(1,3),\ c(0,1) + c(2,3),\ c(0,2) + c(3,3)\} + w(0,3)$$

$$= \min\{0 + 12,\ 8 + 3,\ 19 + 0\} + 14$$

$$= \min\{12, 11, 19\} + 14$$

$$= 11 + 14$$

$$= 25$$

$r(0,3) = 2$ since $k = 2$

Scanned with CamScanner

$$c(1,4) = \min_{1<k\le 4} \{ \underset{k=2}{c(1,2-1) + c(2,4)}, \underset{k=3}{c(1,3-1) + c(3,4)}, \underset{k=4}{c(1,4-1) + c(4,4)} \} + w(1,4)$$

$$= \min\{ c(1,1) + c(2,4), c(1,2) + c(3,4), c(1,3) + c(4,4) \} + w(1,4)$$

$$= \min\{ 0+8, 7+3, 12+0 \} + 11$$

$$= \min\{8, 10, 12\} + 11$$

$$= 8 + 11$$

$$= 19$$

$r(1,4) = 2$ since $k = 2$

Case 4 :- $j - i = 4$ $(0,4)$
$4 - 0$

$$w(0,4) = P(4) + q(4) + w(0,4-1)$$

$$= 1 + 1 + w(0,3)$$

$$= 1 + 1 + 14$$

$$= 16$$

$$c(0,4) = \min_{0<k\le 4} \{ \underset{k=1}{c(0,1-1) + c(1,4)}, \underset{k=2}{c(0,2-1) + c(2,4)}, \underset{k=3}{c(0,3-1) + c(3,4)}, \underset{k=4}{c(0,4-1) + c(4,4)} \} + w(0,4)$$

$$= \min\{ c(0,0) + c(1,4), c(0,1) + c(2,4), c(0,2) + c(3,4), c(0,3) + c(4,4) \} + w(0,4)$$

$$= \min\{ 0+19, 8+8, 19+3, 25+0 \} + 16$$

$$= \min\{19, 16, 22, 25\} + 16 = 16 + 16 = 32$$

Rank $r(0,4) = 2$. Since $k = 2$

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

Now, Consider the last pair from the table

$w(0,4) = 16$

$c(0,4) = 32$

$r(0,4) = 2$

Sub the values in $T_{i,j} = T_{i,k-1}, T_{k,j}$

$k = 2$
Since $r(0,4) = 2$ $\therefore$ $T_{0,4} = T_{0,2-1}, T_{2,4}$

$= T_{0,1}, T_{2,4}$

Consider $T_{0,1} = T_{0,1-1}, T_{1,1}$ $k=1$ since $r_{01}=1$

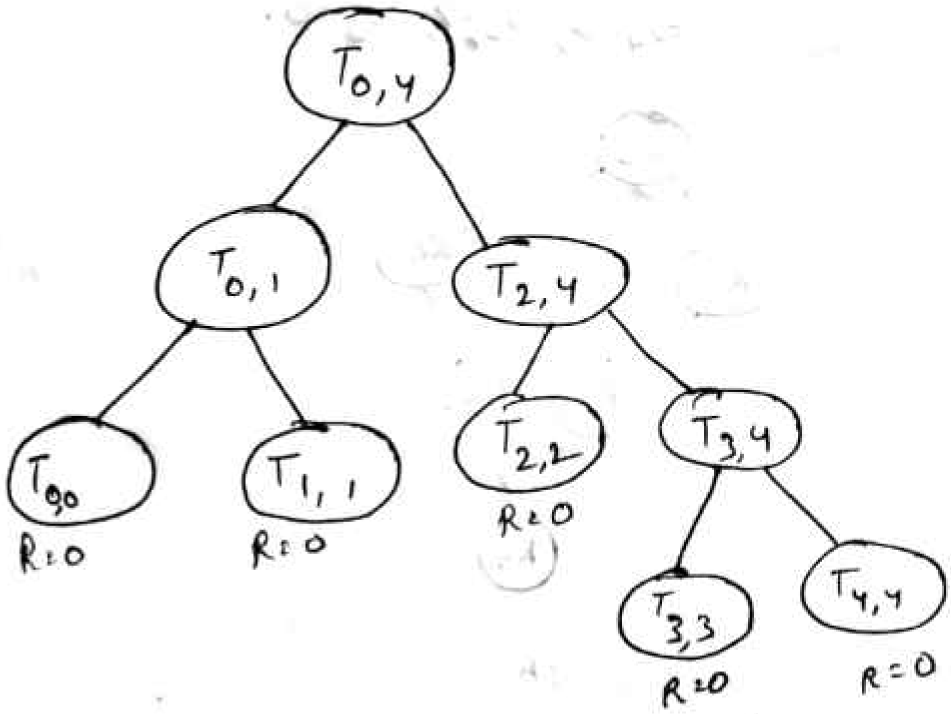
$= T_{0,0}, T_{1,1}$

$T_{2,4} = T_{2,3-1}, T_{3,4}$

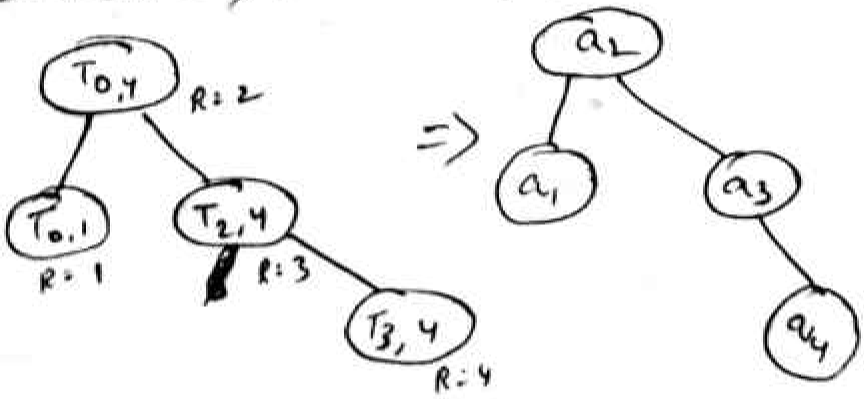
$= T_{2,2}, T_{3,4}$

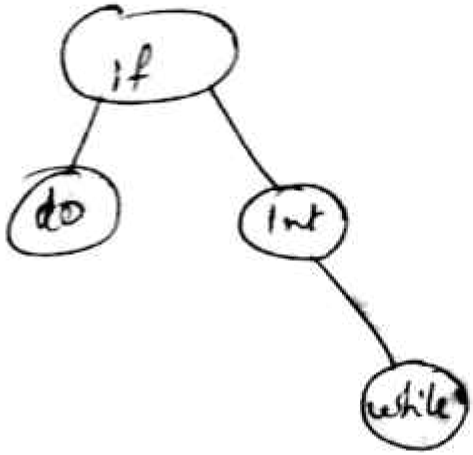
$T_{3,4} = T_{3,4-1}, T_{4,4}$

$= T_{3,3}, T_{4,4}$



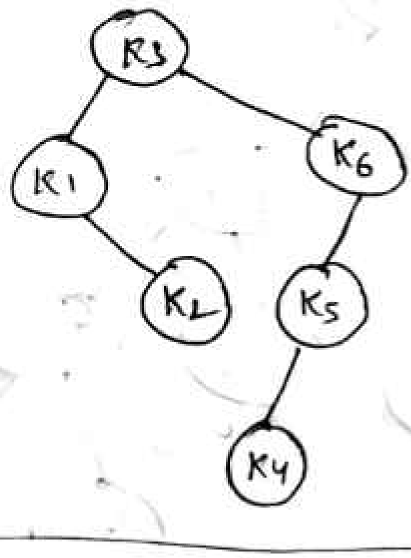
Since rank is 0, removed Internal nodes





The above is optimal BST.

Ex-2 :- Find the optimal binary search tree for $n=6$, having keys $k_1, \dots, k_6$ and weights $p_1=10$, $p_2=3$, $p_3=9$, $p_4=2$, $p_5=0$, $p_6=10$; $q_0=5$, $q_1=6$, $q_2=4$, $q_3=4$, $q_4=3$, $q_5=8$, $q_6=0$.



using Dynamic approach

$$C(i,j) = \begin{cases} \min\limits_{i<k\leq j} \left\{ c(i,k-1) + c(k,j) + \sum\limits_{i=1}^{j} p_i \right. & \text{if } i<j \\ p_i & \text{if } i=j \\ 0 & \text{if } i>j \end{cases}$$

Algorithm to Compute cost of OBST:-

Algorithm C(i, j)

{

if C(i,j) already computed then

return C(i,j)

if i > j then

return 0

else if i = j then

return p_i

else

return $\min_{i < k \le j} \{C(i, k-1) + C(k, j)\} + \sum_{i=1}^{j} p_i$

}

Complexity

The time complexity is $O(n^3)$.

0/1 Knapsack problem

method-1

Let $c[i,\omega]$ to be the solution for items $1, 2, \ldots i$ and maximum weight ω. Then

$$c[i,\omega] = \begin{cases} 0 & \text{if } i=0 \text{ \& } \omega=0 \\ c[i-1,\omega] & \text{if } \omega_i \geq 0 \\ \max\{V_i + c[i-1, \omega-\omega_i], c[i-1,\omega]\} & \text{if } i>0 \text{ and } \omega \geq \omega_i \end{cases}$$

where V_i : value or profits of i^{th} item

ω : ~~Total maximum~~ weight ~~of the~~ consider to put into Knapsack

ω_i : weight of i^{th} item.

Ex: Consider the problem having weights and profits

weights = {3, 4, 6, 5}

profits = {2, 3, 1, 4}

The weight of the knapsack is 8kg

Sol: Since, the ~~weights and~~ items are 4. The possible no. of combinations are $2^n = 2^4 = 16$.

In a way, that 1 denotes that the item is picked and 0 denotes that item is not picked.

The optimal solution is that which combination is providing the maximum profit.

We create a matrix, ~~same~~ with columns 9 starting from 0 to 8, since the weight is 8. And rows ~~with~~ 5 starting from 0 to 4, since the items are 4.

Profits \ weights	0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0	0
1	0	0	0	2	2	2	2	2	2
2	0	0	0	2	3	3	3	5	5
3	0	0	0	2	3	3	3	5	5
4	0	0	0	2	3	~~0~~4	4	5	6

The first row and first column would be 0 as there is no item for $w=0$

when $i=1, w=1$

$c[1,4] = \max\{c[1-1,4],\ c[1-1,4-3]+2\}$

$= \max\{c[0,4],\ c[0,1]+2\}$

$= \max\{0,\ 0+2\}$

$= 2$

$c[1,5] = \max\{c[1-1,5],\ c[1-1,5-3]+2\}$

$= \max\{c[0,5],\ c[0,2]+2\}$

$= \max\{0,\ 0+2\}$

$= 2$

$c[1,6] = \max\{c[0,6],\ c[0,3]+2\}$

$= 2$

$c[1,7] = \max\{c[0,7],\ c[0,4]+2\}$

$= 2$

$c[1,8] = \max\{c[0,8],\ c[0,5]+2\}$

$= 2$

when $i=2, w=1$

$c[2,1] = \max\{c[2-1,1],\ c[2-1,1-4]+3\}$

$= \max\{c[1,1],\ c[1,-3]+3\} = 0+0 = 0$

not possible

$c[2,1] = 0$

$c[2,2] = \max\{c[2-1,2],\ \underline{c[2-1,2-4]+3}\} = 0+0 = 0$

not possible

$= 0$

$c[2,3] = \max\{c[2-1,3],\ c[2-1,3-4]+3\}$

$= \max\{2,0\} = 2$

$c[2,4] = \max\{c[2-1,4],\ c[2-1,4-4]+3\} = 0$

$= \max\{c[1,4],\ c[1,0]+3\}$

$= \max\{2,3\} = 3$

$c[2,5] = \max\{c[2-1,5], c[2-1,5-4]+3\}$

$= \max\{c[1,5], c[1,1]+3\}$

$= \max\{2, 0+3\}$

$= 3$

$c\{2,6] = \max\{c[2-1,6], c[2-1,6-4]+3\}$

$= \max\{2, 0+3\}$

$= 3$

$c[2,7] = \max\{c\{2-1,7], c[2-1,7-4]+3\}$

$= \max\{c[1,7], c[1,3]+3\}$

$= \max\{2, 2+3\}$

$= 5$

$c[2,8] = \max\{c[2-1,8], c[2-1,8-4]+3\}$

$= \max\{2, 2+3\} = 5$

when $i=3, w=1$

$c[3,1] = \max\{c[3-1,1], c[3-1,1-6]+1\}$

$= \max\{c[2,1], c[2,-5]+1\}$

$= \max\{0,0\} = 0$

$c[3,2] = \max\{c[3-1,2], c[3-1,2-6]+1\}$

$= \max\{0,0\} = 0$

$c[3,3] = \max\{c[3-1,3], c[3-1,3-6]+1\}$

$= \max\{2,0\}$

$= 2$

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

$= \max\{c[2,4], c[2,-2]+1\}$

$= \max\{3,0\} = 3$

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

$$= \max\{c[2,5],\ c[2,-1]+1\}$$

$$= \max\{3, 0\} = 3$$

$$c[3,6] = \max\{c[3-1,6],\ c[3-1,6-6]+1\}$$

$$= \max\{3, 1\}$$

$$= 3$$

$$c[3,7] = \max\{c[3-1,7],\ c[3-1,7-6]+1\}$$

$$= \max\{5, 0+1\} = 5$$

$$c[3,8] = \max\{c[3-1,8],\ c[3-1,8-6]+1\}$$

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

$$= 5$$

when $i=4, w=1$

$$c[4,1] = \max\{c[4-1,1],\ c[4-1,1-5]+4\}$$

$$= \max\{0, 0\} = 0$$

$$c[4,2] = \max\{c[4-1,2],\ c[4-1,2-5]+4\}$$

$$= \max\{0, 0\}$$

$$= 0$$

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

$$= \max\{2, 0\}$$

$$= 2$$

$$c[4,4] = \max\{c[4-1,4],\ c[4-1,4-5]+4\}$$

$$= \max\{3, 0\}$$

$$= 3$$

$c[4,5] = \max\{c[4-1,5], c[4-1,5-5]+4\}$

$= \max\{3, 4\}$

$= 4$

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

$= \max\{3, 4\}$

$= 4$

$c[4,7] = \max\{c[4-1,7], c[4-1,7-5]+4\}$

$= \max\{5, 4\}$

$= 5$

$c[4,8] = \max\{c[4-1,8], c[4-1,8-5]+4\}$

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

$= 6$

Add all the values obtained in the table.

item	x_1	x_2	x_3	x_4
	1	0	0	1

Since we have 4 items

profit of item

$6 - 4 = 2$

$2 - 2 = 0$

$\therefore$ The optimal solution is 1001

ie., w_1, w_4 are taken

Check

$\sum w_i x_i \leq m$

$\Rightarrow 3\times1 + 4\times0 + 6\times0 + 5\times1 \leq 8$

$\Rightarrow 3 + 5 \leq 8$ (T)

$\sum p_i x_i = 2\times1 + 3\times0 + 1\times0 + 4\times1$

$= 2+0+0+4 = \underline{6}$

Algorithm 0/1 knapsack(V, w, n, W)

{

for $w = 0$ to W

do $C[0, w] = 0$

for $i = 1$ to n

do $C[i, 0] = 0$

for $w = 1$ to W

do if $w_i \le w$

~~then if $v_i + C[i-1, w-w_i]$~~

then if $C[i, w] = v_i + C[i-1, w-w_i]$

else $C[i, w] = C[i-1, w]$

~~else~~

Analysis:-

Here 2^n comparsions are made so, $O(2^n)$ is Complexity.

Ex:- $P = \{1, 2, 5, 6\}$ $m = 8$ (capacity)

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

x_1	x_2	x_3	x_4
0	1	0	1

Solution

weights

Profits	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

Travelling Salesperson problem : (TSP)

· The TSP is a popular problem. Dynamic programming is a popular approach for solving the TSP. TSP is given as graph $G = (V, E)$ with weighted adjacency matrix.

Let us assume that vertex 1 is the starting node of the TSP tour. The problem involves computation of the minimum cost path that starting from vertex 1 visits all other vertices exactly once.

The distance matrix $d[i,j]$ is computed from the weighted adjacency matrix as follows:

$$d[i,j] = \begin{cases} \infty & \text{if } edge(i,j) \notin E(G) \\ 0 & \text{if } i = j \\ w_{ij} & \text{if } edge(i,j) \in E(G) \end{cases}$$

The recursion involves computation of a function g~~cost~~(i, S) that indicates the shortest path starting from vertex 1, visiting all the vertices of set S, and ending at vertex i. The function g~~cost~~(S, i) is given as follows:

$$g\,\cancel{cost}(i, S) = \min_{k \in S} \{ c_{ik} + g(k, S - \{k\}) \}$$

The final cost of the tour can be computed using dynamic programming as follows:

$$g(1, V-\{1\}) = \min_{2 \le k \le n} \{ C_{1k} + g(k, V-\{1,k\}) \}$$

Informal algorithm

step-1: Read weighted graph $G = \langle V, E \rangle$

step-2: Initialize $d[i,j]$ as follows:

$$d[i,j] \text{ or } C_{ij} = \begin{cases} \infty & \text{if edge}(i,j) \notin E(G) \\ 0 & \text{if } i = j \\ \omega_{ij} & \text{if edge}(i,j) \in E(G) \end{cases}$$

step3:- Compute a function $g(i, s)$, a function that gives the length of the shortest path starting from a vertex i, ~~travelling~~ traversing through all the vertices in set S, and terminating at vertex i

~~as follows~~

Step-4:- Compute the minimum cost of the travelling salesperson tour as follows:

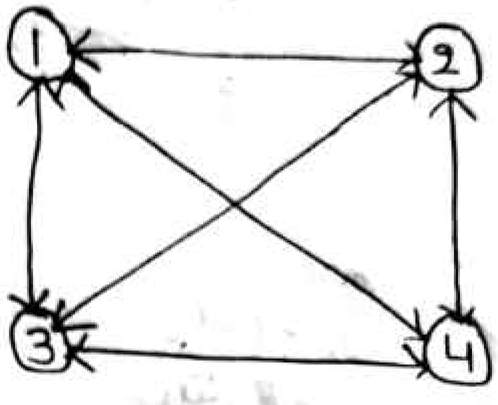
Compute $g(1, V-\{1\}) = \min_{2 \le k \le n} \{ c_{ik} + g(K, V-\{1,K\})$

using $g(i, S)$ computed in step-3

step-5: Return the value

Sol:-

Q. Solve the Travelling sales person problem for the graph given using the dynamic Programming.



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

Sol: Initially, we can start at vertex 1, (or) vertex 2 (or) vertex 3 (or) vertex 4.

- Suppose, we start at vertex 1, then we can visit the vertices 2, (or 3, (or) 4.
- Suppose after node 1 and length of $S=0$, the cases formed are:

→ Case: 0 when $|S|=0$

- $g(2, \emptyset)$
- $g(3, \emptyset)$
- $g(4, \emptyset)$

→ if $|S|=1$, we have to compute :

⇒ • Case : 1

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

→ if $|S|=2$, we have to compute :

⇒ • Case : 2

- $g(2, \{3,4\})$
- $g(3, \{2,4\})$
- $g(4, \{2,3\})$

→ if $|S|=3$, we have to compute :

⇒ • Case : 3

- $g(1, \{2,3,4\})$

* Now, substitute the given values in the formula : $g(i,S) = \min\limits_{k \in S} \{ c_{ik} + g(k, S-\{k\}) \}$

∴ Case : 0 ($|S|=0$) [For initial case : use the formula - $g(i,\emptyset) = c_{i1}$

- $g(2,\emptyset)$

$\cancel{g(2,\emptyset)} = \cancel{\min\limits_{k \in S} \{ c_{ik} + g(k, S-\{k\}) \}}$ (~~not possible~~)

$\therefore g(2,\emptyset) = c_{i1}$

$= c_{21} = 5$

- $g(3,\emptyset) = c_{i1}$

$= c_{31} = 6$

- $g(4, \phi) = C_{41}$
  $= 8$

$\boxed{g(i, \phi) = C_{i1}}$

* Case : 2

→ For case: 2 , we use the Formula :

$$g(i,S) = \min_{K \in S} \{ C_{iK} + g(K, S - \{K\}) \}$$

- $g(2, \{3\}) = \min_{K \in S} \{ C_{23} + g(3, \{3\} - \{3\}) \}$
  $= \min_{K \in S} \{ C_{23} + g(3, \phi) \}$
  $= \min_{K \in S} \{ 9 + 6 \}$
  $= 15$
- $g(2, \{4\}) = \min_{K \in S} \{ C_{24} + g(4, \{4\} - \{4\}) \}$
  $= \min_{K \in S} \{ C_{24} + g(4, \phi) \}$
  $= \min_{K \in S} \{ 10 + 8 \}$
  $= 18$
- $g(3, \{2\}) = \min_{K \in S} \{ C_{32} + g(2, \{2\} - \{2\}) \}$
  $= \min_{K \in S} \{ C_{32} + g(2, \phi) \}$
  $= \min_{K \in S} \{ 13 + 5 \}$
  $= 18$
- $g(3, \{4\}) = \min_{K \in S} \{ C_{34} + g(4, \{4\} - \{4\}) \}$
  $= \min_{K \in S} \{ C_{34} + g(4, \phi) \}$
  $= \min_{K \in S} \{ 12 + 8 \}$

$= 20.$

- $g(4,\{2\}) = \min\limits_{k \in S} \{ c_{42} + g(2, \{2\} - \{2\}) \}$

$= \min\limits_{k \in S} \{ c_{42} + g(2, \phi) \}$

$= \min\limits_{k \in S} \{ 8 + 5 \}$

$= 13$

- $g(4,\{3\}) = \min\limits_{k \in S} \{ c_{43} + g(3, \{3\} - \{3\}) \}$

$= \min\limits_{k \in S} \{ c_{43} + g(3, \phi) \}$

$= \min\limits_{k \in S} \{ 9 + 6 \}$

$= 15$

* Case: 3

→ For case: 3, we use the Formula:

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

- $g(2,\{3,4\}) = \min\limits_{k \in S} \{ c_{23} + g(3, \{3,4\} - \{3\}),$

$c_{24} + g(4, \{3,4\} - \{4\}) \}$

$= \min\limits_{k \in S} \{ c_{23} + g(3, \{4\}),$

$c_{24} + g(4, \{3\}) \}$

$= \min\limits_{k \in S} \{ 9 + 20, 10 + 15 \}$

$= \min\limits_{k \in S} \{ 29, 25 \}$

$= 25$

- $g(3,\{2,4\}) = \min\limits_{k \in S} \{ c_{32} + g(2, \{2,4\} - \{2\}),$

$c_{34} + g(4, \{2,4\} - \{4\}) \}$

$$= \min_{k \in S} \{ C_{32} + g(2, \{4\}),$$
$$C_{34} + g(4, \{2\}) \}$$

$$= \min_{k \in S} \{ 13 + 18, 12 + 13 \}$$

$$= \min_{k \in S} \{ 31, 25 \}$$

$$= 25$$

- $g(4, \{2,3\}) = \min_{k \in S} \{ C_{42} + g(2, \{2,3\} - \{2\}),$
$$C_{43} + g(3, \{2,3\} - \{3\}) \}$$

$$= \min_{k \in S} \{ C_{42} + g(2, \{3\}),$$
$$C_{43} + g(3, \{2\}) \}$$

$$= \min_{k \in S} \{ 8 + 15, 9 + 18 \}$$

$$= \min_{k \in S} \{ 23, 27 \}$$

$$= 23$$

* Case : 4

→ For Case : 4, we use the formula :

$$g(i, S) = \min_{k \in S} \{ C_{ik} + g(k, S - \{k\}) \}$$

- $g(1, \{2,3,4\}) = \min_{k \in S} \{ C_{12} + g(2, \{2,3,4\} - \{2\}),$
$$C_{13} + g(3, \{2,3,4\} - \{3\}),$$
$$C_{14} + g(4, \{2,3,4\} - \{4\}) \}$$

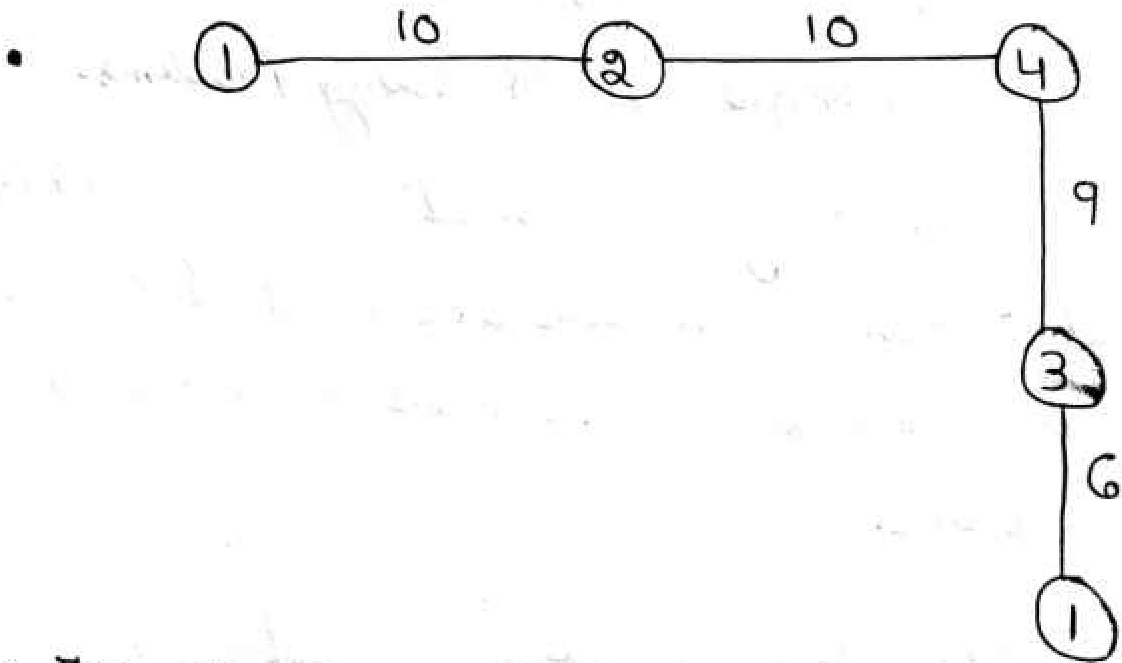
$$= \min_{k \in S} \{ C_{12} + g(2, \{3,4\}),$$
$$C_{13} + g(3, \{2,4\}),$$
$$C_{14} + g(4, \{2,3\}) \}$$

$$= \min_{k \in S} \{ 10 + 25, \; 15 + 25, \; 20 + 23 \}$$

$$= \min_{k \in S} \{ 35, 40, 43 \}$$

$$= \underline{\underline{35}}$$

* Optimal Solution:



∴ The above solution is the shortest path.

∴ The Time complexity is : $O(n \cdot 2^n)$.