Design and Analysis of Algorithms

Unit - III

Dr. R. Bhuvaneswari
Assistant Professor
Department of Computer Science
Periyar Govt. Arts College, Cuddalore.

Periyar Govt. Arts College
Cuddalore

Dynamic Programming

Syllabus

UNIT-III

Dynamic Programming: General Method – Multistage Graphs – All-Pair shortest paths –Optimal binary search trees – 0/1 Knapsack – Travelling salesperson problem.

Text Book:

Ellis Horowitz, Sartaj Sahni and Sanguthevar Rajasekaran, Computer Algorithms C++, Second Edition, Universities Press, 2007. (For Units II to V)

Dynamic Programming

General Method:

- It is an algorithm design method that can be used when the solution to a problem can be viewed as a sequence of decisions.
- It obtains the solution using "**Principle of Optimality**".
- It states that " In an optimal sequence of decisions or choices, each subsequence must also be optimal", ie., whatever the initial state and decision are, the remaining decisions must constitute an optimal decision sequence.
- The 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.
- Sequences containing suboptimal subsequences cannot be optimal and so will not be generated.

Multistage Graphs

- A multistage graph G = (V, E) is a directed graph in which the vertices are partitioned into $k \geq 2$ disjoint sets V_i, $1 \leq i \leq k$.
- If $\langle u, v \rangle$ is an edge in E, then $u \in V_i$ and $v \in V_{i+1}$.
- The sets V_1 and V_k are such that $|V_1| = |V_k| = 1$.
- The vertex **s** is the source and the **t** the sink (destination).
- The multistage graph problem is to find a minimum cost path from s to t.
- The cost of s to t is the sum of the cost of the edges on the path.
- The multistage graph problem can be solved in 2 ways.
  - Forward method
  - Backward method

Multistage Graphs

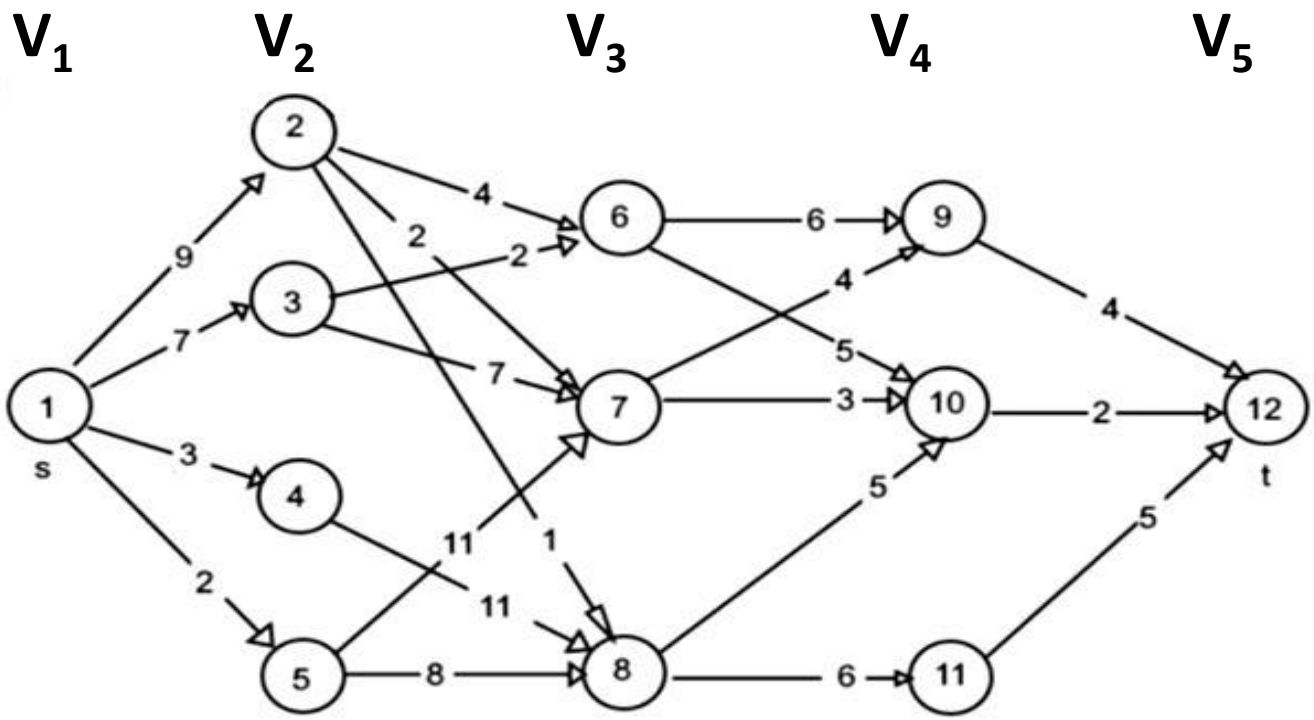
Forward Approach

- In the forward approach, the cost of each and every node is found starting from the k stage to the 1st stage.
- The minimum cost path from the source to destination is found ie., stage 1 to stage k.
- For forward approach,

$$Cost(i, j) = \min_{\substack{l \in V_{i+1} \\ \langle j, l \rangle \in E}} \{c(j, l) + cost(i+1, l)\}$$

  where i is the level number.
- Time complexity: O(|V|+|E|)

Multistage Graphs



Periyar Govt. Arts College
Cuddalore

Minimum cost estimation – Forward Approach

$$Cost(i, j) = \min_{l \in V_{i+1}, \langle j, l \rangle \in E} \{c(j, l) + cost(i+1, l)\}$$

		Min. Cost
cost(5,12)	0	0
cost(4,9)	min{c(9,12)+cost(5,12)} = {4 + 0}	4
cost(4,10)	min{**c(10,12)+cost(5,12)**} = {2 + 0}	2
cost(4,11)	min{c(11,12)+cost(5,12)} = {5+ 0}	5
cost(3,6)	min{c(6,9)+cost(4,9), **c(6,10)+cost(4,10)**} = min{6+ 4, **5+2**}	7
cost(3,7)	min{c(7,9)+cost(4,9), **c(7,10)+cost(4,10)**} = min{4+4, **3+2**}	5
cost(3,8)	min{c(8,10)+cost(4,10), c(8,11)+cost(4,11)} = min{5+2, 6+5}	7

Minimum cost estimation – Forward Approach

		Min. Cost
cost(2,2)	min{c(2,6)+cost(3,6), **c(2,7)+cost(3,7)**, c(2,8)+cost(3,8)} = min{4+7, 2+5, 1+7}	7
cost(2,3)	min{**c(3,6)+cost(3,6)**, c(3,7)+cost(3,7)} = min{2+7, 7+5}	9
cost(2,4)	min{c(4,8)+cost(3,8)} = min{11+7}	18
cost(2,5)	min{c(5,7)+cost(3,7), c(5,8)+cost(3,8)} = min{11+5, 8+7}	15
cost(1,1)	min{**c(1,2)+cost(2,2)**, **c(1,3)+cost(2,3)**, c(1,4)+cost(2,4), c(1,5)+cost(2,5)} = min{9+7, 7+9, 3+18, 2+15}	16

1 ⇨ 2 ⇨ 7 ⇨ 10 ⇨ 12

1 ⇨ 3 ⇨ 6 ⇨ 10 ⇨ 12

Multistage Graphs

```
Algorithm FGraph(G, k, n, p)
//p[1:k] is a minimum cost path
{
   cost[n] = 0.0;
   for j = n-1 to 1 step -1 do
  {
     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;
  }
  p[1] = 1; p[k] = n;
  for j = 2 to k-1 do
              p[j] = d[p[j-1]];
}
```

Multistage Graphs

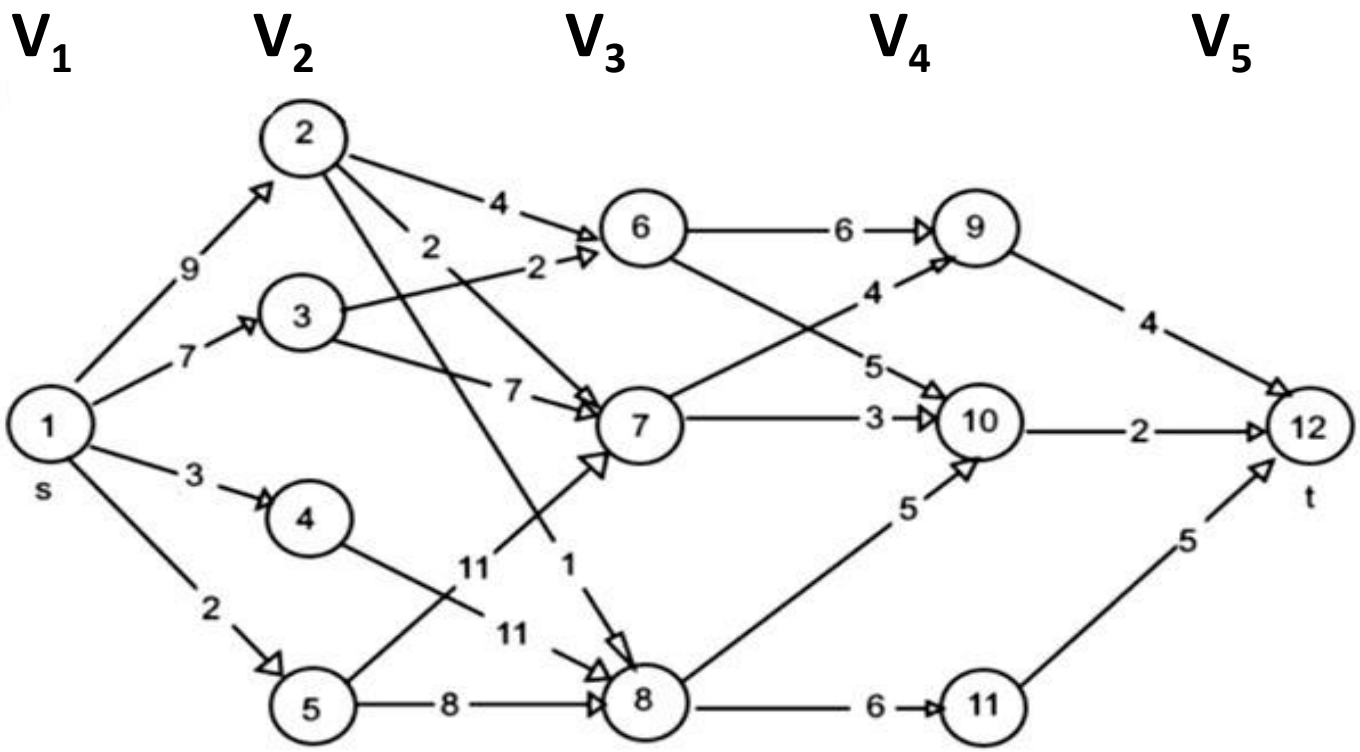
Backward Approach

- In the backward approach, the cost of each and every node is found starting from the 1^{st} stage to the k^{th} stage.
- The minimum cost path from the source to destination is found ie., stage k to stage 1.
- For backward approach,

$$bcost(i, j) = \min_{\substack{l \in V_{i-1} \\ \langle l, j \rangle \in E}} \{bcost(i-1, l) + c(l, j)\}$$

where i is the level number.

Multistage Graphs



Dr. R. Bhuvaneswari
Periyar Govt. Arts College
Cuddalore

Minimum cost estimation – Backward Approach

$$bcost(i, j) = \min_{\substack{l \in V_{i-1} \\ \langle l, j \rangle \in E}} \{bcost(i-1, l) + c(l, j)\}$$

		Min. Cost
bcost(1,1)	0	0
bcost(2,2)	min{**bcost(1,1)+c(1,2)**} =min{0+9}	9
bcost(2,3)	min{**bcost(1,1)+c(1,3)**} =min{0+7}	7
bcost(2,4)	min{bcost(1,1)+c(1,4)} =min{0+3}	3
bcost(2,5)	min{bcost(1,1)+c(1,5)} =min{0+2}	2
bcost(3,6)	min{bcost(2,2)+c(2,6),**bcost(2,3)+c(3,6)**} = min{9+4,7+2}	9
bcost(3,7)	min{**bcost(2,2)+c(2,7)**,bcost(2,3)+c(3,7), bcost(2,5)+c(5,7)} = min{9+2,7+7,2+11}	11

Minimum cost estimation – Backward Approach

		Min. Cost
bcost(3,8)	min{bcost(2,2)+c(2,8),bcost(2,4)+c(4,8), bcost(2,5)+c(5,8)} = min{9+1,3+11,2+8}	10
bcost(4,9)	min{bcost(3,6)+c(6,9),Bcost(3,7)+c(7,9)} = min{9+6,11+4}	15
bcost(4,10)	min{**bcost(3,6)+c(6,10)**,**bcost(3,7)+c(7,10)**, bcost(3,8)+c(8,10)} = min{9+5,11+3,10+5}	14
bcost(4,11)	min{bcost(3,8)+c(8,11)} = min{10+6}	16
Bcost(5,12)	min{bcost(4,9)+c(9,12),**bcost(4,10)+c(10,12)**, bcost(4,11)+c(11,12)} = min{15+4,14+2,16+5}	16

12 ⇨ 10 ⇨ 7 ⇨ 2 ⇨ 1

12 ⇨ 10 ⇨ 6 ⇨ 3 ⇨ 1

1⇨ 2 ⇨ 7 ⇨ 10 ⇨ 12

1⇨ 3 ⇨ 6 ⇨ 10 ⇨ 12

Multistage Graphs

```
Algorithm BGraph(G, k, n, p)
{
  bcost[1] = 0.0;
  for j = 2 to n do
  {
    Let r be such that ⟨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;
  }
  p[1] = 1; p[k] = n;
  for j = k-1 to 2 step -1 do
        p[j] = d[p[j+1]];
}
```

All pair shortest paths

- All pairs shortest path problem is the determination of the shortest graph distances between every pair of vertices in a given directed graph G.
- That is, for every pair of vertices (i, j), we are to find a shortest path from i to j as well as from j to i. These two paths are the same when G is undirected.
- Let G = (V, E) be a directed graph with n vertices.
- Let cost be a cost adjacency matrix for G such that cost(i, i) = 0, $1 \leq i \leq n$.
- Cost(i, j) is the length of edge $\langle i, j \rangle$ if $\langle i, j \rangle \in E(G)$ and cost(i, j) = ∞ if $i \neq j$ and $\langle i, j \rangle \notin E(G)$.
- All pair shortest path problem is to determine a matrix A such that A(i, j) is the length of a shortest path from i to j.
- Since each application of this procedure requires $O(n^2)$ time, the matrix A can be obtained in $O(n^3)$ time.

All pair shortest paths

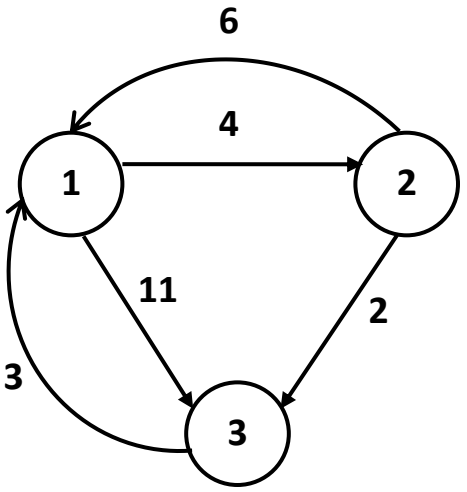
- The shortest i to j path in G, i ≠ j originates at vertex i and goes through some intermediate vertices and terminates at vertex j.
- If k is an intermediate vertex on this shortest path, then the subpaths from i to k and from k to j must be shortest paths from i to k and k to j, respectively.
- Otherwise, the i to j path is not of minimum length.
- So, the principle of optimality holds.
- Let $A^k(i, j)$ represent the length of a shortest path from i to j going through no vertex of index greater than k, we obtain:

$$A(i,j) = \min_{1 \le k \le n} \{A^{k-1}(i,k) + A^{k-1}(k,j), cost(i,j)\}$$

- Time complexity of this algorithm is $O(n^3)$

All pair shortest paths

```
Algorithm AllPaths(cost, A, n)
{
   for i =1 to n do
   {
    for j = 1 to n do
           A[i, j] = cost[i, j];
   }
   for k = 1 to n do
   {
    for j = 1 to n do
    {
      for j = 1 to n do
           A[i, j] = min{A[i, j], A[i, k]+A[k, j]};
    }
   }
}
```



1->3 = 11
1->2->3 = 6
2->1 = 6
2->3->1 = 5

All pair shortest paths

Solve the problem for k = 1, 2, 3

Cost adjacency matrix

A^0	1	2	3
1	0	4	11
2	6	0	2
3	3	∞	0

Solving the equation for, k = 1

A^1	1	2	3
1	0	4	11
2	6	0	2
3	3	7	0

Solving the equation for, k = 2

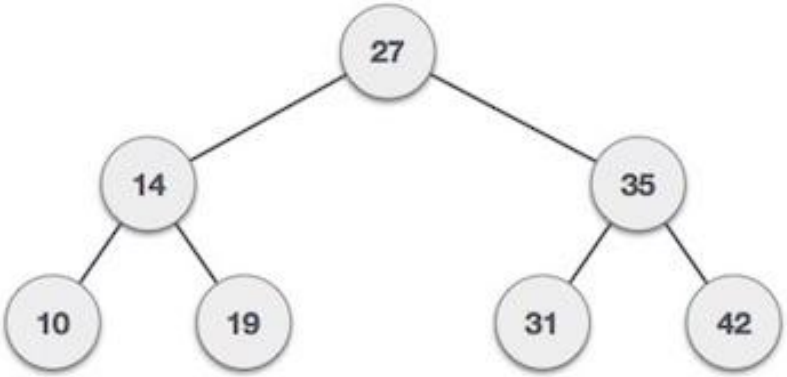
A^2	1	2	3
1	0	4	6
2	6	0	2
3	3	7	0

Solving the equation for, k = 3

A^3	1	2	3
1	0	4	6
2	5	0	2
3	3	7	0

Optimal Binary Search Trees

- A Binary Search Tree (BST) is a tree in which all the nodes follow the below mentioned properties :
  - ❖ The value of the key of the left sub-tree is less than the value of its parent (root) node's key.
  - ❖ The value of the key of the right sub-tree is greater than or equal to the value of its parent (root) node's key.



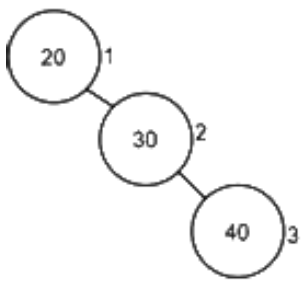
- An optimal binary search tree (OBST) is a binary search tree which provides the smallest possible search time.

Optimal Binary Search Trees

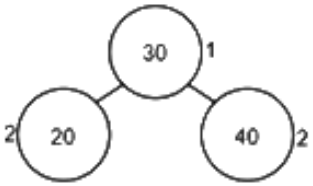
- The problem is to construct an optimal binary search tree (in terms of search time) for a set of integer keys, given the frequencies with which each key will be accessed.

Keys	20	30	40
Frequencies	2	1	6

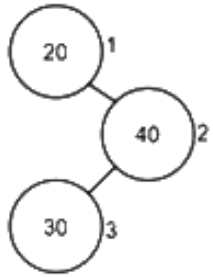
- As there are three different keys, we can get a total of 5 various BSTs by changing order of the keys. ie., $2nC_n/(n+1)$ number of tree can be generated.



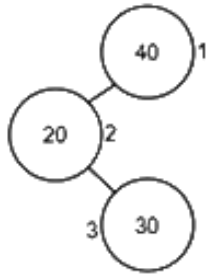
(1 * 2) +
(2 * 1) +
(3 * 6)
=22



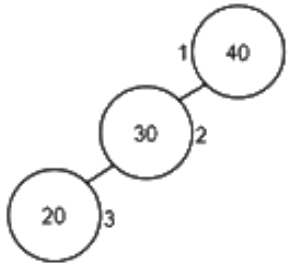
(1 * 1) +
(2 * 2) +
(2 * 6)
=17



(1 * 2) +
(2 * 6) +
(3 * 1)
=17



(1 * 6) +
(2 * 2) +
(3 * 1)
=13



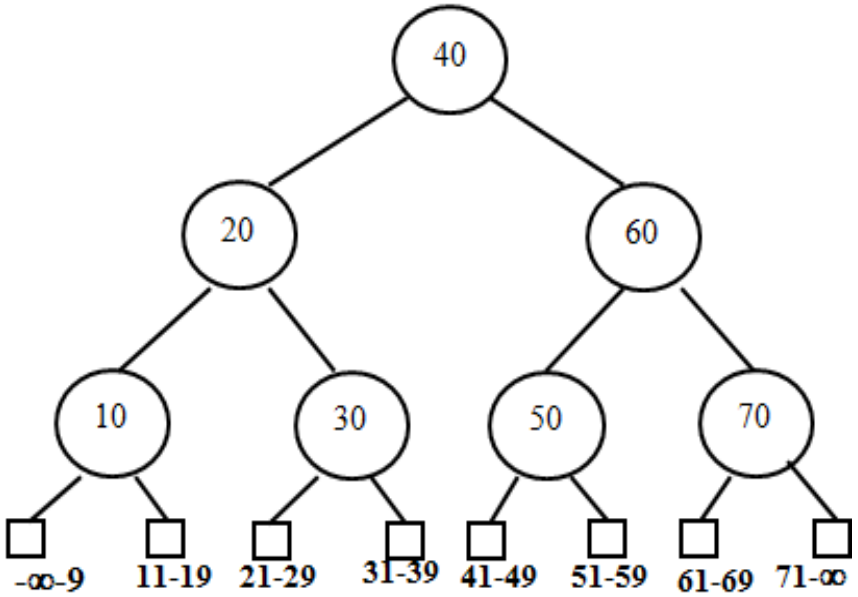
(1 * 6) +
(2 * 1) +
(3 * 2)
=14

Optimal Binary Search Trees

- The cost is computed by multiplying the each node's frequency with the level of tree(Here we are assuming that the tree starts from level 1) and then add them to compute the overall cost of BST.
- The above example, the 4th BST has the least cost among all, so it is the Optimal Binary Search Tree for the given data.
- If the number of nodes are less we can find optimal BST by checking all possible arrangements, but if the nodes are greater than 3 like 4,5,6….. then respectively 14,42,132….. , different BST's are possible so by checking all arrangements to find Optimal Cost may lead to extra overhead.
- So Dynamic Programming approach can be used to find the Optimal Binary Search Tree.
- The search time can be improved in Optimal Cost Binary Search Tree, placing the most frequently used data in the root and closer to the root element, while placing the least frequently used data near leaves and in leaves.

Optimal Binary Search Trees

- It has n keys (representation $k_1,k_2,\ldots,k_n$) in sorted order (so that $k_1<k_2<\ldots<k_n$), and we build a binary search tree from these keys.
- For each k_i ,we have a probability p_i that a search will be for k_i.
- In contrast, some searches may be for values not in k_i, and so we also have n+1 "dummy keys" $d_0,d_1,\ldots,d_n$ representating not in k_i.
- In particular, d_0 represents all values less than k_1, and d_n represents all values greater than k_n, and for i=1,2,…,n-1, the dummy key d_i represents all values between k_i and k_{i+1}.



Optimal Binary Search Trees

- The dummy keys are leaves (external nodes), and the data keys mean internal nodes.
- For n internal nodes n+1 external nodes will be present.
- The terminal node that is the left successor of k_1 can be interpreted as representing all key values that are not stored and are less than k_1. Similarly, the terminal node that is the right successor of k_n, represents all key values not stored in the tree that are greater than k_n.

Using Dynamic Approach

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

$$\mathbf{w[i,j] = w[i,j-1] + p_j + q_j}$$

Let p_i be the probability of successful search and q_j be the probability of unsuccessful search.

Optimal Binary Search Trees

Example:

Keys = {10, 20, 30, 0}

p(1:4) = {3, 3, 1, 1}

q(0:4) = {2, 3,1, 1, 1}

Initially, w(i, i) = q(i); c(i, i) = 0;

r(i, i) = 0

W[0,1] = w[0,0]+p1+q1=2+6=8

W[1,2] = w[1,1]+p2+q2=3+3+1=7

W[2,3] = w[2,2]+p3+q3=1+1+1=3

W[3,4] = w[3,3]+p4+q4 =1+1+1=3

W[0,2] = w[0,1]+p2+q2

W[1,3] = w[1,2]+p3+q3

W[2,4] = w[2,3]+p4+q4

W[0,3] = w[0,2]+p3+q3

W[1,4] = w[1,3]+p4+q4

W[0,4] = w[0,3]+p4+q4

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

Optimal Binary Search Trees

c[0,1] = min{**c[0,0]+c[1,1]**}+w[0,1] = 0+0+8=8
0<k≤1

c[1,2] = min{c[1,1]+c[2,2]}+w[1,2]=0+0+7=7
1<k≤2

c[2,3] =min{c[2,2]+c[3,3]}+w[2,3]=0+0+3=3
2<k≤3

c[3,4]=min{**c[3,3]+c[4,4]**}+w[3,4]=0+0+3=3
3<k≤4

c[0,2]=min{c[0,0]+c[1,2], c[0,1]+c[2,2]}+w[0,2]
0<k≤2

=min{0+7,8+0}+12=19

c[1,3]=min{c[1,1]+c[2,3],c[1,2]+c[3,3]}+w[1,3]
1<k≤3

=min{0+3,7+0}+9 = 12

c[2,4]=min{**c[2,2]+c[3,4]**,c[2,3]+c[4,4]}+w[2,4]
2<k≤4

=min{0+3,3+0}+5=8

Optimal Binary Search Trees

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

$0<k\leq 3$

$=\min\{0+12,8+3,19+0\}+14 = 25$

$c[1,4] = \min\{c[1,1]+c[2,4],\ c[1,2]+c[3,4],\ c[1,3]+c[4,4]\}+w[1,4]$

$1<k\leq 4$

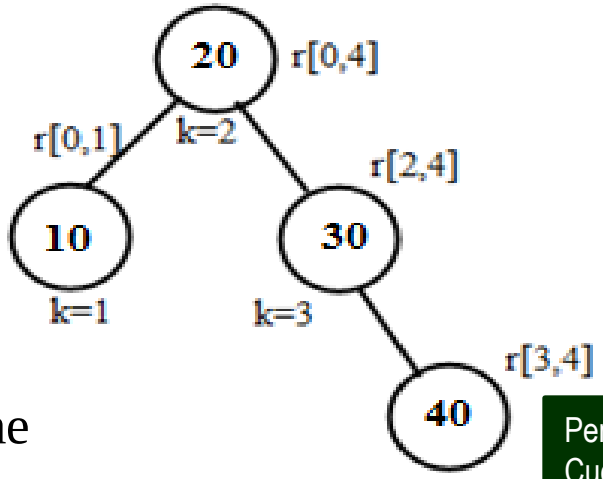
$= \min\{0+8,7+3,12+0\}+11 = 19$

$c[0,4] = \min\{c[0,0]+c[1,4],\ \mathbf{c[0,1]+c[2,4]},\ c[0,2]+c[3,4],\ c[0,3]+c[4,4]\}+w[0,4]$

$0<k\leq 4$

$=\min\{0+19,\mathbf{8+8},19+3,25+0\}+16 = 32$

The Optimal Binary Search Tree:



The algorithm requires $\mathbf{O(n^3)}$ time

0/1 Knapsack Problem

- Given n objects and a knapsack or bag.
- $w_i \rightarrow$ weight of object i.
- $m \rightarrow$ knapsack capacity.
- As the name suggests, objects are indivisible in this method. No fractional objects can be taken. An object can either be taken completely or left completely.
- Objective is to fill the knapsack that maximizes the total profit earned.
- Problem can be stated as

$$\text{maximize} \sum_{1 \le i \le n} p_i x_i$$

$$\text{subject to} \sum_{1 \le i \le n} w_i x_i \le m$$

$$x_i = 0 \text{ or } 1, 1 \le i \le n$$

0/1 Knapsack Problem

0/1 knapsack problem is solved using dynamic programming in the following steps-

- Draw a table say 'V' with (n+1) number of rows and (w+1) number of columns.
- Fill all the boxes of 0^{th} row and 0^{th} column with zeroes.
- Start filling the table row wise top to bottom from left to right.
- Use the following formula:

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

- value of the last box represents the maximum possible value that can be put into the knapsack.

0/1 Knapsack Problem

- To identify the items that must be put into the knapsack to obtain the maximum profit,
  - Consider the last column of the table.
  - Start scanning the entries from bottom to top.
  - On encountering an entry whose value is not same as the value stored in the entry immediately above it, mark the row label of that entry.
  - After all the entries are scanned, the marked labels represent the items that must be put into the knapsack.
- O(nw) time is taken to solve 0/1 knapsack problem using dynamic programming.

0/1 Knapsack Problem

$P_i = \{1, 2, 5, 6\}$

$w_i = (2, 3, 4, 5\}$

$m = 8, n = 4$

	W→	0	1	2	3	4	5	6	7	8
P_i	w_i	0	0	0	0	0	0	0	0	0
1	2	1	0	0	1	1	1	1	1	1
2	3	2	0	0	1	2	2	3	3	3
5	4	3	0	0	1	2	5	5	6	7
6	5	4	0	0	1	2	5	5	6	7

V[i ,W] = max{V[i-1 ,W] , V[i-1, W – w[i]] + p[i]}

V[1,1] = max{V[0,1], V[0,1-2]+1} = max{0, -} = 0

V[1,2] = max{V[0,2], V[0,2-2]+1} = max{0, 0+1} = 1

V[1,3] = max{V[0,3], V[0,3-2]+1} = max{0,0+1} = 1

V[1,4] = max{V[0,4], V[0,4-2]+1} = max{0, 0+1} = 1

V[1,5] = max{V[0,5], V[0,5-2]+1} = max{0, 0+1} = 1

V[1,6] = max{V[0,6], V[0,6-2]+1} = max{0, 0+1} = 1

V[1,7] = max{V[0,7], V[0,7-2]+1} = max{0, 0+1} = 1

V[1,8] = max{V[0,8], V[0,8-2]+1} = max{0, 0+1} = 1

0/1 Knapsack Problem

V[2,1] = max{V[1,1],V[1,1-3]+2} = max{0, -} = 0
V[2,2] = max{V[1,2],V[1,2-3]+2} = max{1, -} = 1
V[2,3] = max{V[1,3],V[1,3-3]+2} = max{1, 0+2} = 2
V[2,4] = max{V[1,4],V[1,4-3]+2} = max{1, 0+2} = 2
V[2,5] = max{V[1,5],V[1,5-3]+2} = max{1, 1+2} = 3
V[2,6] = max{V[1,6],V[1,6-3]+2} = max{1, 1+2} = 3
V[2,7] = max{V[1,7],V[1,7-3]+2} = max{1, 1+2} = 3
V[2,8] = max{V[1,8],V[1,8-3]+2} = max{1, 1+2} = 3

V[3,1] = max{V[2,1],V[2,1-4]+5} = max{0, -} = 0
V[3,2] = max{V[2,2],V[2,2-4]+5} = max{1, -} = 1
V[3,3] = max{V[2,3],V[2,3-4]+5} = max{2, -} = 2
V[3,4] = max{V[2,4],V[2,4-4]+5} = max{2, 0+5} = 5
V[3,5] = max{V[2,5],V[2,5-4]+5} = max{2, 0+5} = 5
V[3,6] = max{V[2,6],V[2,6-4]+5} = max{2, 1+5} = 6

0/1 Knapsack Problem

V[3,7] = max{V[2,7],V[2,7-4]+5} = max{2, 2+5} = 7
V[3,8] = max{V[2,8],V[2,8-4]+5} = max{2, 2+5} = 7

V[4,1] = max{V[3,1],V[3,1-5]+6} = max{0, -} = 0
V[4,2] = max{V[3,2],V[3,2-5]+6} = max{1, -} = 1
V[4,3] = max{V[3,3],V[3,3-5]+6} = max{2, -} = 2
V[4,4] = max{V[3,4],V[3,4-5]+6} = max{5, -} = 5
V[4,5] = max{V[3,5],V[3,5-5]+6} = max{5, 0+6} = 6
V[4,6] = max{V[3,6],V[3,6-5]+6} = max{6, 0+6} = 6
V[4,7] = max{V[3,7],V[3,7-5]+6} = max{7, 1+6} = 7
V[4,8] = max{V[3,8],V[3,8-5]+6} = max{7, 2+6} = **8**

$x_1 = 0, x_2 = 1, x_3 = 0, x_4 = 1$

0/1 Knapsack Problem

```
for (i = 0; i ≤ n; i++)
{
   for(w = 0; w ≤ m; w++)
   {
      if(i==0 || w==0)
            k[i][w] = 0;
      else if(wt[i] ≤ w)
            k[i][w] = max(p[i]+k[i-1][w-wt[i], k[i-1][w]);
      else
            k[i][w] = k[i-1][w];
   }
}
```

Traveling Salesperson Problem

- The traveling salesperson problem is to find a tour of minimum cost.
- Let G = (V, E) be a directed graph with edge cost $C_{ij} = \infty$ if $\langle i, j \rangle \notin E$
- Let |V| = n and assume |n| > 1
- A tour G is a directed simple cycle that includes every vertex in V.
- The cost of a tour is the sum of the cost of the edges on the tour.
- Let 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.
- The function g(1, V-{1}) is the length of an optimal salesperson tour.

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

$$g(i, S) = \min_{j \in S} \{c_{ij} + g(j, S - \{j\})\} - - - - - 2$$

Traveling Salesperson Problem

$g(i,\phi) = C_{i1}, 1 \le i \le n.$

$|S| = \phi$

$g(2,\phi) = c_{21} = 5$

$g(3,\phi) = c_{31} = 6$

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

Using equation 2, we obtain

$|S| = 1$

$g(2,\{3\}) = c_{23} + g(3,\phi) = 9+6 = 15$

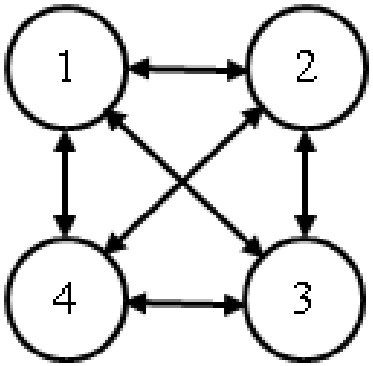
$g(2,\{4\}) = c_{24} + g(4,\phi) = 10+8 = 18$

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

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

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

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



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

Traveling Salesperson Problem

|S| = 2

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

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

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

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

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

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

Using equation 1, we obtain

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

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

The optimal tour is

1->2->4->3->1

$O(2^n n^2)$ time is taken to solve the traveling salesperson problem