UNIT-4

Dynamic Programming: Introduction, 0/1 Knapsack problem, All pairs shortest paths, Optimal Binary search trees, Travelling salesman problem.

Q) Briefly explain dynamic programming.

D*ynamic Programming* is a general algorithm design technique for solving problems defined by recurrences with overlapping subproblems i.e; subproblems are not independent they subproblems share subsubproblems

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.

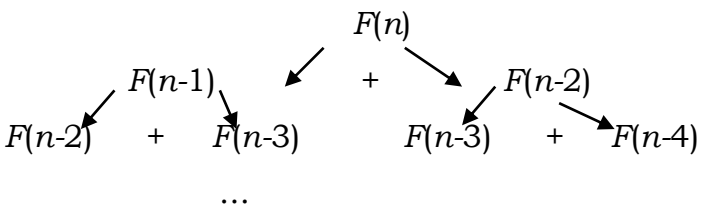
Main idea:

- set up a recurrence relating a solution to a larger instance to solutions of some smaller instances
- solve smaller instances once
- record solutions in a table
- extract solution to the initial instance from that table

In DP many decision sequences may be generated, however sequences containing suboptimal subsequences can not be optimal and hence they will not be generated.

Let us suppose that we need to make a sequence of decisions $x_1,x_2..x_n$. In forward approach the decisions on the x_i are made in the order $x_1,x_2..x_n$.
In backward approach, the decisions on the x_i are made in the order x_n, x_n, .. x_2, x_1.

- Eg. Computing the n^{th} Fibonacci number recursively (top-down):



Computing the n^{th} Fibonacci number using bottom-up iteration (dynamic programming) and recording values:

$F(0) = 0$

$F(1) = 1$

$F(2) = 1+0 = 1$

…

$F(n-2) =$

$F(n-1) =$

$F(n) = F(n-1) + F(n-2)$

0	1	1	..	F(n-2)	F(n-1)	F(n)

Q) What is principal of optimality?

Principle of optimality: Suppose that in solving a problem, we have to make a sequence of decisions D_1, D_2, …, D_n. If this sequence is optimal, then the last k decisions, 1 <k <n must be optimal.

e.g. the shortest path problem

If i, i_1, i_2, …, j is a shortest path from i to j, then i_1, i_2, …, j must be a shortest path from $i_1$to j.

Q) Explain about transitive closure with an example.

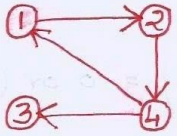
The ***transitive closure*** of a directed graph with n vertices can be defined as the $n \times n$ boolean matrix $T = \{tij\}$, in which the element in the ith row and the jth column is 1 if there exists a nontrivial path (i.e., directed path of a positive length) from the ith vertex to the jth vertex; otherwise, tij is 0.

ALGORITHM *Warshall*$(A[1..n, 1..n])$

```
//Implements Warshall's algorithm for computing the transitive closure
//Input: The adjacency matrix A of a digraph with n vertices
//Output: The transitive closure of the digraph
R^(0) ← A
for k ← 1 to n do
    for i ← 1 to n do
        for j ← 1 to n do
            R^(k)[i, j] ← R^(k−1)[i, j] or (R^(k−1)[i, k] and R^(k−1)[k, j])
return R^(n)
```

Time complexity: $O(n^3)$

Eg. Find the transitive closure for the following graph.



$R^{(0)}$ is the adjacency matrix for the given graph where '1's reflect the existence of paths with no intermediate vertices, otherwise '0's.

$$\therefore R^{(0)} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \end{bmatrix} \end{array}$$

From Warshal's alg we have the following formula/ reccurence relation:

$$r_{ij}^{(k)} = r_{ij}^{(k-1)} \text{ or } \left(r_{ik}^{(k-1)} \text{ and } r_{kj}^{(k-1)} \right)$$

When computing $R^{(1)}$, 1st row & 1st column of $R^{(0)}$ remains same in $R^{(1)}$,

K ↓

$r_{22}^{(1)} = r_{22}^{(0)} \text{ or } \left(r_{21}^{(0)} \text{ and } r_{12}^{(0)} \right)$

i ↓ ↘ j

$= 0 \text{ or } (0 \text{ and } 1) = 0 \text{ or } 0 = 0$

$r_{23}^{(1)} = r_{23}^{(0)} \text{ or } \left(r_{21}^{(0)} \text{ and } r_{13}^{(0)} \right) = 0 \text{ or } (0 \text{ and } 0) = 0$

$r_{24}^{(1)}$ not necessary to compute as it had already $1 \Rightarrow r_{24}^{(1)} = 1$.

$r_{32}^{(1)} = r_{32}^{(0)} \text{ or } \left(r_{31}^{(0)} \text{ and } r_{12}^{(0)} \right) = 0 \text{ or } (0 \text{ and } 1) = 0$

$r_{33}^{(1)} = r_{33}^{(0)} \text{ or } \left(r_{31}^{(0)} \text{ and } r_{13}^{(0)} \right) = 0 \text{ or } (0 \text{ and } 0) = 0$

$r_{34}^{(1)} = r_{34}^{(0)} \text{ or } \left(r_{31}^{(0)} \text{ and } r_{14}^{(0)} \right) = 0 \text{ or } (0 \text{ and } 0) = 0$

$r_{42}^{(1)} = r_{42}^{0}$ or $\left(r_{41}^{(0)} \text{ and } r_{12}^{(0)} \right) = 0$ or $(1 \text{ and } 1) = 1$

$r_{43}^{1} = 1 \quad (\because r_{43}^{0} = 1)$

$r_{44}^{1} = r_{44}^{0}$ or $\left(r_{41}^{(0)} \text{ and } r_{14}^{(0)} \right) = 0$ or $(1 \text{ and } 0) = 0$

$$\therefore R^{(1)} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 \end{bmatrix} \end{array}$$

$$\text{Using } R^{(2)} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix} \end{array}$$

$$\text{Using } R^{(3)} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix} \end{array}$$

$$\text{Using } R^{(4)} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix} \end{array}$$

$\therefore$ In $R^{(4)}$, 1's reflect the existence of paths with intermediate vertices

Q) Explain All pairs shortest path(Floyd's Warshall's Algorithm) with suitable example.

Given a weighted connected graph (undirected or directed), the ***all-pairs shortest paths problem*** asks to find the distances—i.e., the lengths of the shortest paths— from each vertex to all other vertices.

•Let d k(i, j)be the length of a shortest path from i to j with intermediate vertices numbered not higher than k where 0 ≤ k ≤ n, then

d^0(i, j)=c(i, j)(no intermediate vertices at all)

•d^k(i, j)=min { d^{k-1}(i, j), d^{k-1}(i, k)+ d^{k-1}(k, j) }

–and d^n(i, j)is the length of a shortest path from i to j

we need to find d^n with d^0=cost matrix .

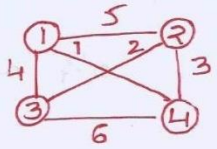
•General formula: d^k [i, j]= min {d^{k-1}[i, j], d^{k-1}[i, k]+ d^{k-1}[k, j] }

ALGORITHM *Floyd*$(W[1..n, 1..n])$

```
//Implements Floyd's algorithm for the all-pairs shortest-paths problem
//Input: The weight matrix W of a graph with no negative-length cycle
//Output: The distance matrix of the shortest paths' lengths
D ← W  //is not necessary if W can be overwritten
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
```

Time Complexity: $O(n^3)$

Eg. Find the shortest distance from each vertex to other vertices in the below graph.



The All Pairs shortest-path problem is to determine a matrix A such that $A(i,j)$ is the length of a shortest path i, j.

From floyd's alg. we have the recurrence relation :

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

$$\Rightarrow A^0(i,j) = cost(i,j), \quad 1\le i\le n, \quad 1\le j\le n$$

$$\boxed{A^k(i,j) = \min\{A^{k-1}(i,j), A^{k-1}(i,k) + A^{k-1}(k,j)\}, \quad k\ge 1}$$

The cost matrix for given graph is given by

$$A^0 = cost(i,j) = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array}\begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 5 & 4 & 1 \\ 5 & 0 & 2 & 3 \\ 4 & 2 & 0 & 6 \\ 1 & 3 & 6 & 0 \end{bmatrix} \end{array}$$

where $cost(i,j) = \begin{cases} 0 & \text{if } i=j \\ \text{edge cost}(i,j) & \text{if } i\neq j \ \&\ (i,j)\in E \\ \infty & \text{if } i\neq j \ \&\ (i,j)\notin E \end{cases}$

and E is set of edges in graph 'G'.

When computing $A^{(1)}$,

- 1st row & 1st column can be copied from $A^{(0)}$ to $A^{(1)}$ as they remain constant.
- all diagonal elements will be 0's always.

K = 1

$a^1_{23} = \min\{a^0_{23}, (a^0_{21} + a^0_{13})\}$

$= \min\{2, (5+4)\} = 2$

$a^1_{24} = \min\{a^0_{24}, (a^0_{21} + a^0_{14})\} = \min\{3, (5+1)\} = 3$

$a^1_{32} = \min\{a^0_{32}, (a^0_{31} + a^0_{12})\} = \min\{2, (4+5)\} = 2$

$a^1_{34} = \min\{a^0_{34}, (a^0_{31} + a^0_{14})\} = \min\{6, (4+1)\} = 5$

$a^1_{42} = \min\{a^0_{42}, (a^0_{41} + a^0_{12})\} = \min\{3, (1+5)\} = 3$

$a^1_{43} = \min\{a^0_{43}, (a^0_{41} + a^0_{13})\} = \min\{6, (1+4)\} = 5$

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

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

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

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

Q) Explain 0/1 knapsack problem using dynamic programming with suitable example.

Knapsack Problem

Given a set of items, each with a weight and a value, determine a subset of items to include in a collection so that the total weight is less than or equal to a given limit and the total value is as large as possible.

Fractional Knapsack

In this case, items can be broken into smaller pieces, hence we can select fractions of items.

According to the problem statement,

- There are **n** items in the store
- Weight of $\mathbf{i^{th}}$ item wi > 0
- Profit for $\mathbf{i^{th}}$ item pi>0 and
- Capacity of the Knapsack is **W**

In this version of Knapsack problem, items can be broken into smaller pieces. So, the thief may take only a fraction x_i of $\mathbf{i^{th}}$ item.

$$0 \le x_i \le 1$$

0/1 Knapsack:

In this item cannot be broken which means we should take the item as a whole or should leave it. That's why it is called **0/1 knapsack Problem**.

The classic dynamic programming approach, on the other hand, works

bottom up: it fills a table with solutions to all smaller subproblems, but each of them is solved only once. An unsatisfying aspect of this approach is that solutions to some of these smaller subproblems are often not necessary for getting a solution to the problem given. Since this drawback is not present in the top-down approach, it is natural to try to combine the strengths of the top-down and bottom-up approaches.

The goal is to get a method that solves only subproblems that are necessary and does so only once. Such a method exists; it is based on using memory functions.

This method solves a given problem in the top-down manner but, in addition, maintains a table of the kind that would have been used by a bottom-up dynamic programming algorithm. Initially, all the table's entries are initialized with a special "null" symbol to indicate that they have not yet been calculated. Thereafter, whenever a new value needs to be calculated, the method checks the corresponding entry in the table first: if this entry is not "null," it is simply retrieved from the table; otherwise, it is computed by the recursive call whose result is then recorded in the table.

ALGORITHM *MFKnapsack*(i, j)

```
//Implements the memory function method for the knapsack problem
//Input: A nonnegative integer i indicating the number of the first
//       items being considered and a nonnegative integer j indicating
//       the knapsack capacity
//Output: The value of an optimal feasible subset of the first i items
//Note: Uses as global variables input arrays Weights[1..n], Values[1..n],
//and table F[0..n, 0..W] whose entries are initialized with −1's except for
//row 0 and column 0 initialized with 0's
if F[i, j] < 0
    if j < Weights[i]
        value ← MFKnapsack(i − 1, j)
    else
        value ← max(MFKnapsack(i − 1, j),
                    Values[i] + MFKnapsack(i − 1, j − Weights[i]))
    F[i, j] ← value
return F[i, j]
```

Time Complexity:

The time efficiency and space efficiency of this algorithm are both in θ(nW).

The time needed to find the composition of an optimal solution is in O(n).

Eg. Find the optimal solution for the 0/1 knapsack problem making use of dynamic programming. consider $n=4$, $w=5kg$, $w[1:4]=(2,3,4,5)$ and $P[1:4]=(3,4,5,6)$

Given, Knapsack capacity $(w) = 5kg$

No. of items $(n) = 4$

Weight of each item, $(w_1, w_2, w_3, w_4) = (2, 3, 4, 5)$

and Value of each item, $(P_1, P_2, P_3, P_4) = (3, 4, 5, 6)$

The recurrence relation (formula) is

$$F(i,j) = \begin{cases} \max\{F(i-1,j),\ Value_i + F(i-1, j-weight_i)\} & \text{if } j - w_i \geq 0 \\ F(i-1,j) & j - w_i < 0 \end{cases}$$

Initialize the F table as below (fill 0^{th} row & 0^{th} col. with 0's) and compute remaining values using above formula

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0					
2	0					
3	0					
4	0					

i ↑ , j ↑

computing $F(1,1)$:

$i=1$, $j=1$, $Value_1 = 3$, $weight_1 = 2$.

$\therefore F(1,1) = \max\{F(1-1, 1),\ 3 + F(1-1, 1-2)\}$

$= \max\{F(0,1),\ 3 + F(0,-1)\}$ ↳ ignore as $F(0,-1)$ doesn't exist as $j < weight_i$ i.e; $1 < 2$.

$= T(0,1)$

$= 0$

$$F(1,2) = \max\{F(1-1,2),\ 3+F(1-1,2-2)\}$$
$$= \max\{F(0,2),\ 3+F(0,0)\}$$
$$= \max\{0,\ 3+0\} = 3$$

$$F(1,3) = \max\{F(0,3),\ 3+F(0,1)\} = \max\{0,\ 3+0\} = 3$$
$$F(1,4) = \max\{F(0,4),\ 3+F(0,2)\} = \max\{0,\ 3+0\} = 3$$
$$F(1,5) = \max\{F(0,5),\ 3+F(0,3)\} = \max\{0,\ 3+0\} = 3$$

$$F(2,1) = \max\{F(1,1),\ 4+F(1,-2)\} = F(1,1) = 0$$
$$F(2,2) = \max\{F(1,2),\ 4+F(1,-1)\} = F(1,2) = 3$$
$$F(2,3) = \max\{F(1,3),\ 4+F(1,0)\} = \max\{3,\ 4+0\} = 4$$
$$F(2,4) = \max\{F(1,4),\ 4+F(1,1)\} = \max\{3,\ 4+0\} = 4$$
$$F(2,5) = \max\{F(1,5),\ 4+F(1,2)\} = \max\{3,\ 4+3\} = 7$$

$$F(3,1) = \max\{F(2,1),\ 5+F(2,-3)\} = F(2,1) = 0$$
$$F(3,2) = \max\{F(2,2),\ 5+F(2,-2)\} = F(2,2) = 3$$
$$F(3,3) = \max\{F(2,3),\ 5+F(2,-1)\} = F(2,3) = 4$$
$$F(3,4) = \max\{F(2,4),\ 5+F(2,0)\} = \max\{F(2,4),\ 5+0\} = 5$$
$$F(3,5) = \max\{F(2,5),\ 5+F(2,1)\} = \max\{7,\ 5+0\} = 7$$

$$F(4,1) = \max\{F(3,1),\ 6+F(3,-4)\} = F(3,1) = 0$$
$$F(4,2) = \max\{F(3,2),\ 6+F(3,-3)\} = F(3,2) = 3$$
$$F(4,3) = \max\{F(3,3),\ 6+F(3,-2)\} = F(3,3) = 4$$
$$F(4,4) = \max\{F(3,4),\ 6+F(3,-1)\} = F(3,4) = 5$$
$$F(4,5) = \max\{F(3,5),\ 6+F(3,0)\} = \max\{7,\ 6+0\} = 7.$$

∴ The final F-table from above computations is

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	3	3	3	3
2	0	0	3	4	4	7
3	0	0	3	4	5	7
4	0	0	3	4	5	(7)

The last entry represents the max. possible value that can be kept into the knapsack

Here it is 7

∴ Max. value can be in knapsack = 7.

Finding items in knapsack:

The max. value is 7. So check when 7 is in F-table for the 1st time. It is 2nd row.

∴ I2 is to be included, whose wt. is 3 & value = 4

So remaining profit of knapsack = 7 - 4 = 3

Now check when 3 is kept in T-table for the 1st time. It is 1st row.

∴ I1 is to be included, whose wt is 2 & value = 3

So remaining profit = 3 - 3 = 0.

∴ Solution = $(x1, x2, x3, x4) = (1, 1, 0, 0)$.

Q) Explain travelling salesmen problem using dynamic programming with suitable example.

Given a set of cities and distance between every pair of cities, the problem is to find the shortest possible route that visits every city exactly once and returns to the starting point.

Let G=(V,E) be a directed graph with edge cost C_{ij}. The variable C_{ij} is define such that $C_{ij} > 0$ every i,j and $C_{ij} = \infty$ if $(i,j) \in E$. Let $|V| = n$ and assume $n>1$.

A tour of G is a directed simple cycle that include every vertex in V. The cost of a tour is the sum of the cost of the edges on the tour. The **travelling salesperson problem** is to find a tour of minimum cost without loss of generality, assume a tour is a simple path that starts and ends at vertex 1.

Every tour consists of an edge (1,k) for some $k \in V-\{1\}$ and a path from vertex k to vertex 1. The path from vertex k to vertex 1 goes through each vertex in V-{1,k} exactly once. It is easy to see that if the tour is optimal, then the path from k to 1 must be a shortest k to1 path going through all vertices in V-{1,k}

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.

From the principal of optimality, it follows that

$$g(1,V-\{1\}) = \min_{2 \le k \le n} \{C1k + g(k, V-\{1,k\})\}$$

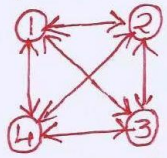
Generalizing above, we obtain for i E S

$$g(i,S) = \min_{j \in S} \{Cij+g(j,S-\{j\})\} \quad \text{-- (1)}$$

Time complexity: $O(n^2 2^n)$ as the computation of g(i,S) with $|S| = k$ requires k-1 comparisons when solving equation (1).

Space Complexity: $O(n.2^n)$

Eg. construct an optimal travelling sales person tour using dynamic programming with following data. Assume starting vertex as 1.



graph, G

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

The recurrence for optimal travelling sales person tour is

$$g(i,S) = \min_{j \in S} \{ c_{ij} + g(j, S-\{j\}) \} \quad \text{—(1)}$$

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

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

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

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

Now by using (1), we have

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

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

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

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

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

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

Now let us compute $g(i,S)$ with $|S| = 2$, $i \notin |S|$, $i \neq 1$, $1 \notin S$

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

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

$= \min\{ 29, 25 \}$

$= 25$.

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

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

$= 25.$

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

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

$= 23.$

finally from (1), we get

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

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

$= \min\{35,\ 40,\ 43\}$

$= 35.$

∴ The optimal tour cost is 35.

The tour of this length can be constructed if we retain with each $g(i,S)$ the value of j that minimizes the rt hand side of eq. (1).

$J(1,\{2,3,4\}) = 2$ ⇒ The tour starts from 1 & moves to 2

$J(2,\{3,4\}) = 4$ ⇒ next we move to 4.

$J(4,\{3\}) = 3$ ⇒ next we move to 3.

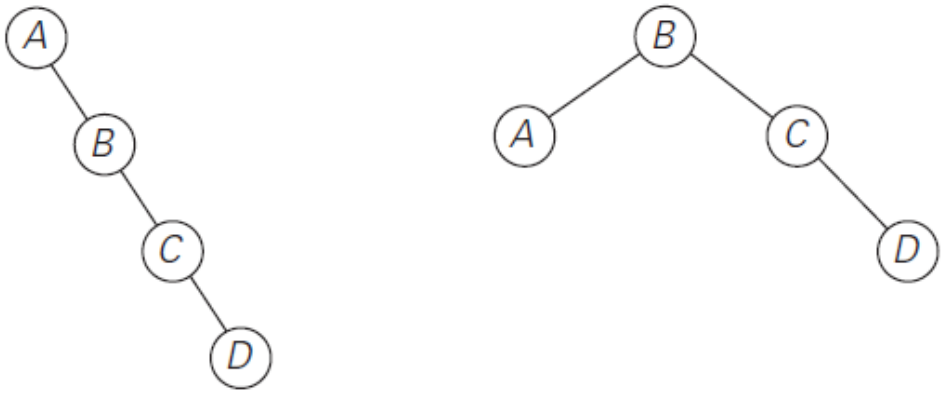
∴ The optimal tour is $1 \to 2 \to 4 \to 3$ with cost = 35.

Q) Explain about Optimal Binary Search Tree with Successful and Unsuccessful search probabilities with suitable example.

OBST is a binary search tree which provides the smallest possible search time (or expected search) for a given sequence of accesses (or access probabilities).

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.

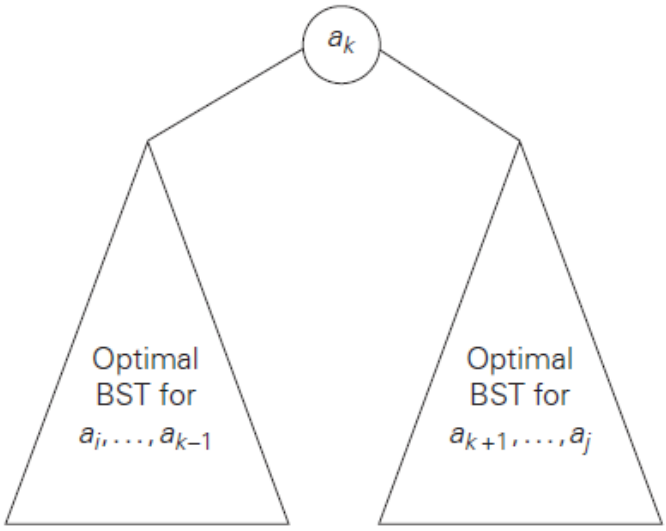
Eg.



' Two out of 14 possible binary search trees with keys A, B, C, and D.

For our tiny example, we could find the optimal tree by generating all 14 binary search trees with 4 keys. As a general algorithm, this exhaustive-search approach is unrealistic: the total number of binary search trees with n keys is equal to the nth **Catalan number**,

$$c(n) = \frac{1}{n+1}\binom{2n}{n} \quad \text{for } n > 0, \quad c(0) = 1,$$



Binary search tree (BST) with root a_k and two optimal binary search subtrees T_i^{k-1} and T_{k+1}^{j}.

Given a set of identifiers $\{a_1,a_2,..,a_n\}$. Suppose we need to construct a binary search tree and p(i) be the probability with which we search for a_i then:

If a binary search tree represents n identifiers, then there will be exactly n internal nodes and n+1 external nodes. Every node internal node represents a point where a successful search may terminate. Every external node represents a point where an unsuccessful search may terminate.

If a successful search terminates at an internal node at level l, then l comparison is needed. Hence the expected cost contribution from the internal node for ai is p(i)*level(ai).

The identifiers not in the binary search tree can be partitioned into n+1 equivalence classes E_i, $0 \le i \le n$. If the failure node for E_i is at level ι, then only ι -1 comparison are needed.

Let q(i) be the probability that the identifier x being searched for is in E_i, then clearly $\sum_{i=1}^{n} Pi + \sum_{i=0}^{n} qi = 1$, and the cost contribution for the failure node for E_i is q(i)*(level (E_i) - 1).

There fore, the cost of the optimal binary search tree is:

$$\sum_{i=1}^{n} Pi * level(ai) + \sum_{i=0}^{n} qi * (level(Ei) - 1)$$

```
Algorithm OBST(p, q, n)
// Given n distinct identifiers a_1 < a_2 < ··· < a_n and probabilities
// p[i], 1 ≤ i ≤ n, and q[i], 0 ≤ i ≤ n, this algorithm computes
// the cost c[i, j] of optimal binary search trees t_ij for identifiers
// a_{i+1}, ..., a_j. It also computes r[i, j], the root of t_ij.
// w[i, j] is the weight of t_ij.
{
    for i := 0 to n − 1 do
    {
        // Initialize.
        w[i, i] := q[i]; r[i, i] := 0; c[i, i] := 0.0;
        // Optimal trees with one node
        w[i, i + 1] := q[i] + q[i + 1] + p[i + 1];
        r[i, i + 1] := i + 1;
        c[i, i + 1] := q[i] + q[i + 1] + p[i + 1];
    }
    w[n, n] := q[n]; r[n, n] := 0; c[n, n] := 0.0;
    for m := 2 to n do  // Find optimal trees with m nodes.
        for i := 0 to n − m do
        {
            j := i + m;
            w[i, j] := w[i, j − 1] + p[j] + q[j];
            // Solve 5.12 using Knuth's result.
            k := Find(c, r, i, j);
                // A value of l in the range r[i, j − 1] ≤ l
                // ≤ r[i + 1, j] that minimizes c[i, l − 1] + c[l, j];
            c[i, j] := w[i, j] + c[i, k − 1] + c[k, j];
            r[i, j] := k;
        }
    write (c[0, n], w[0, n], r[0, n]);
}

Algorithm Find(c, r, i, j)
{
    min := ∞;
    for m := r[i, j − 1] to r[i + 1, j] do
        if (c[i, m − 1] + c[m, j]) < min then
        {
            min := c[i, m − 1] + c[m, j]; l := m;
        }
    return l;
}
```

Time complexity: The computing time for above algorithm is $O(n^2)$. To construct obst from r[i,j] is O(n). So total time to construct obst is $O(n^3)$.

Space complexity = $O(n^2)$

Eg. construct optimal binary search tree for the below data

(a_1, a_2, a_3, a_4) = (end, goto, print, stop)

$P(a_1) = \frac{1}{20}$ $P(a_2) = \frac{1}{5}$ $P(a_3) = \frac{1}{10}$ $P(4) = \frac{1}{20}$

$q(0) = \frac{1}{5}$ $q(1) = \frac{1}{10}$ $q(2) = \frac{1}{5}$ $q(3) = \frac{1}{20}$ $P(4) = \frac{1}{20}$

Given data set is (a_1, a_2, a_3, a_4) = (end, goto, print, stop)

For convenience, we multiply p's & q's with 20.

Then we have,

$(P_1, P_2, P_3, P_4) = (1, 4, 2, 1)$

$(q_0, q_1, q_2, q_3, q_4) = (4, 2, 4, 1, 1)$

We know,

$w_{i,j}$ – wt. of the tree (optimal binary search tree)

$$w_{ij} = P_j + q_j\, w_{i,j-1} \quad \text{if } i<j$$
$$= q_i \quad \text{if } i=j \qquad \text{—(1)}$$

$c(i,j)$ = cost of OBST, then

$$c(i,j) = \min_{i<k\le j} \{ c(i,k-1) + c(k,j) + w(i,j) \} \qquad \text{—(2)}$$
$$c(i,i) = 0 \quad 0 \le i \le n$$

Let $r(i,j)$ – be k value for which a_k is the root of OBST.

$r(i,j)$ = value of k that minimizes

$r(i,i) = 0 \quad 0 \le i \le n.$

For the given instance, $n = 4$

The values $w(i,j)$ $c(i,j)$ $r(i,j)$ are calculated as follows & the values were shown in the following table.

$w_{00} = 4$	$w_{11} = 2$	$w_{22} = 4$	$w_{33} = 1$	$w_{44} = 1$
$c_{00} = 0$	$c_{11} = 0$	$c_{22} = 0$	$c_{33} = 0$	$c_{44} = 0$
$r_{00} = 0$	$r_{11} = 0$	$r_{22} = 0$	$r_{33} = 0$	$r_{44} = 0$
$w_{01} = 7$	$w_{12} = 10$	$w_{23} = 7$	$w_{34} = 3$	
$c_{01} = 7$	$c_{12} = 10$	$c_{23} = 7$	$c_{34} = 3$	
$r_{01} = 1$	$r_{12} = 2$	$r_{23} = 3$	$r_{34} = 4$	
$w_{02} = 15$	$w_{13} = 13$	$w_{24} = 9$		
$c_{02} = 22$	$c_{13} = 20$	$c_{24} = 12$		
$r_{02} = 2$	$r_{13} = 2$	$r_{24} = 3$		
$w_{03} = 18$	$w_{14} = 15$			
$c_{03} = 32$	$c_{14} = 27$			
$r_{03} = 2$	$r_{14} = 2$			
$w_{04} = 20$				
$c_{04} = 39$				
$r_{04} = 2$				

Computations:

From (1) we have $w_{ii} = q(i)$ for $i = 0$ to 4

$\therefore w_{00} = q_0 = 4$, $w_{11} = q_1 = 2$, $w_{22} = q_2 = 4$ $w_{33} = q_3 = 1$

$w_{44} = q_4 = 1$.

From (2) we have $c_{ii} = 0$ & $r_{ii} = 0$ for $i = 0$ to 4.

- $w(0,1) = p_1 + q_1 + w(0,0) = 1+2+4 = 7$

$c(0,1) \Rightarrow i=0, j=1.$

$k=1 \Rightarrow c_{01} = \min\{c_{00}+c_{11}\} + w_{01}$

$= (0+0) + 7 = 7.$

$r_{01} = 1.$

- $w_{12} = p_2 + q_2 + w_{11} = 4+4+2 = 10.$

$c_{12} \overset{k=2}{=} \min\{c_{11}+c_{22}\} + w_{12} = (0+0)+10 = 10.$

$r_{12} = 2$

- $w_{23} = p_3 + q_3 + w_{22} = 2+1+4 = 7$

$c_{24} : k=3.$

$c_{24} = \min\{c_{22}+c_{33}\} + w_{23} = (0+0)+7 = 7$

$r_{23} = 3$

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

$c_{34} : k=4.$

$c_{34} = \min\{c_{33}+c_{44}\} + w_{34} = (0+0)+3 = 3$

$r_{34} = 4.$

- $w_{02} = p_2 + q_2 + w_{01} = 4+4+7 = 15$

$c_{02} : k = 1, 2$

$c_{02} = \min\{\underset{k=1}{c_{00}+c_{12}}, \underset{k=2}{c_{01}+c_{22}}\} + w_{02}$

$= \min\{0+10, 7+0\} + w_{02}$

$= 7+15 = 22$

$r_{02} = 2$

• $w_{13} = p_3 + q_3 + w_{12} = 2 + 1 + 10 = 13$

c_{13} : $k = 2, 3.$

$$c_{13} = \min\{\underbrace{c_{11} + c_{23}}_{k=2}, \underbrace{c_{12} + c_{33}}_{k=3}\} + w_{13}$$
$$= \min\{0 + 7, 10 + 0\} + 13$$
$$= 7 + 13 = 20.$$

$r_{13} = 2.$

• $w_{24} = p_4 + q_4 + w_{23} = 1 + 1 + 7 = 9$

c_{24} : $k = 3, 4$

$$c_{24} = \min\{\underbrace{c_{22} + c_{34}}_{k=3}, \underbrace{c_{23} + c_{44}}_{k=4}\} + w_{24}$$
$$= \min\{0 + 3, 7 + 0\} + 9 = 3 + 9 = 12$$

$r_{24} = 3.$

• $w_{03} = p_3 + q_3 + w_{02} = 2 + 1 + 15 = 18$

c_{03} : $k = 1, 2, 3.$

$$c_{03} = \min\{\underbrace{c_{00} + c_{13}}_{k=1}, \underbrace{c_{01} + c_{23}}_{k=2}, \underbrace{c_{02} + c_{33}}_{k=3}\} + w_{03}$$
$$= \min\{0 + 20, 7 + 7, 22 + 0\} + 18 = 14 + 18 = 32$$

$r_{03} = 2$

• $w_{14} = p_4 + q_4 + w_{13} = 1 + 1 + 13 = 15$

c_{14} : $k = 2, 3, 4$

$$c_{14} = \min\{\underbrace{c_{11} + c_{24}}_{k=2}, \underbrace{c_{12} + c_{34}}_{k=3}, \underbrace{c_{13} + c_{44}}_{k=4}\} + w_{14}$$
$$= \min\{0 + 12, 10 + 3, 20 + 0\} + 15 = 27$$

$r_{14} = 2$

$w_{04} = p_4 + q_4 + w_{03} = 1 + 1 + 18 = 20$

c_{04} : $k = 1, 2, 3, 4.$

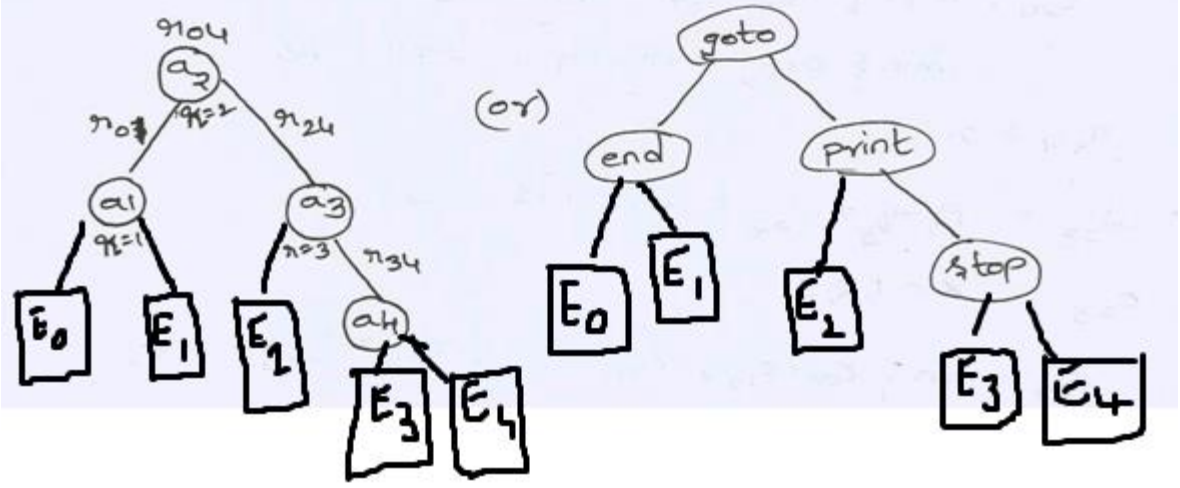
$$c_{04} = \min\{\underset{k=1}{c_{00}+c_{14}}, \underset{k=2}{c_{01}+c_{24}}, \underset{k=3}{c_{02}+c_{34}}, \underset{k=4}{c_{03}+c_{44}}\} + w_{04}$$

$$= \min\{0+27, 7+12, 22+3, 32+0\} + 20$$

$$= 19 + 20 = 39.$$

$r_{04} = 2$

∴ From above table, the cost of optimal binary search tree is $c_{04} = 39$.



Successful Search cost of the tree = 1(2) + 4(1) + 2(2) + 1(3) = 13.

Unsuccessful search cost of the tree = 4(3-1) + 2(3-1) + 4(3-1) + 1(4-1) + 1(4-1) = 26

So, total cost of the tree = 13+26 =39.

Q) Explain about Optimal Binary Search Tree with Successful search probabilities with suitable example.

- OBST is a binary search tree which provides the smallest possible search time (or expected search) for a given sequence of accesses (or access probabilities).

- 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.

- Given a set of identifiers $\{a_1,a_2,..,a_n\}$. Suppose we need to construct a binary search tree and p(i) be the probability with which we search for a_i then:

  If a successful search terminates at an internal node at level l, then l comparison is needed. Hence the expected cost contribution from the internal node for ai is p(i)*level(ai).

- There fore, the cost of the optimal binary search tree is:

  $\sum_{i=1}^{n} \text{Pi} * \text{level(ai)}$

ALGORITHM *OptimalBST*$(P[1..n])$

```
//Finds an optimal binary search tree by dynamic programming
//Input: An array P[1..n] of search probabilities for a sorted list of n keys
//Output: Average number of comparisons in successful searches in the
//        optimal BST and table R of subtrees' roots in the optimal BST
for i ← 1 to n do
    C[i, i − 1] ← 0
    C[i, i] ← P[i]
    R[i, i] ← i
C[n + 1, n] ← 0
for d ← 1 to n − 1 do    //diagonal count
    for i ← 1 to n − d do
        j ← i + d
        minval ← ∞
        for k ← i to j do
            if C[i, k − 1] + C[k + 1, j] < minval
                minval ← C[i, k − 1] + C[k + 1, j];   kmin ← k
        R[i, j] ← kmin
        sum ← P[i];   for s ← i + 1 to j do sum ← sum + P[s]
        C[i, j] ← minval + sum
return C[1, n], R
```

Time Complexity: The computing time for above algorithm is $O(n^2)$. To construct obst from r[i,j] is O(n). So total time to construct obst is $O(n^3)$.

Space complexity: $O(n^2)$

Eg. construct an optimal binary search tree for the following keys with the probabilities as

Keys	A	B	C	D	E
Prob.	0.25	0.2	0.05	0.2	0.3

For constructing OBST we have the following recurrence:

$$c(i,j) = \min_{i \le k \le j} \{c(i,k-1) + c(k+1,j)\} + \sum_{s=i}^{j} P_s \quad , 1 \le i \le j \le n \qquad \text{---(1)}$$

we assume in above formula that,

$$c(i, i-1) = 0 \quad , \quad 1 \le i \le n+1$$

that are no. of comparisions in empty tree

and from above (1),

$$c(i, i) = P_i \quad , \quad 1 \le i \le n$$

for one node binary search tree a_i.

The initial tables look like:

main table

	0	1	2	3	4	5
1	0	0.25				
2		0	0.2			
3			0	0.05		
4				0	0.2	
5					0	0.3
6						0

root table

	0	1	2	3	4	5
1		1				
2			2			
3				3		
4					4	
5						5

$$c(1,2) = \min \begin{cases} K=1 \Rightarrow c(1,0)+c(2,2)+\sum_{q=1}^{2} P_q = 0+0.2+0.25+0.2 = 0.65 \\ K=2 \Rightarrow c(1,1)+c(3,2)+\sum_{q=1}^{2} P_q = 0.25+0+0.25+0.2 = 0.7 \end{cases}$$

$\therefore c(1,2) = 0.65$ & $r_{12} = 1$

$$c(2,3) = \min \begin{cases} K=2 \Rightarrow c(2,1)+c(3,3)+\sum_{q=1}^{j} P_q = 0+0.05+0.2+0.05 = 0.05+0.25 = 0.3 \\ K=3 \Rightarrow c(2,2)+c(4,3)+\sum_{q=1}^{j} P_q = 0.2+0+0.25 = 0.45 \end{cases}$$

$\therefore c(2,3) = 0.3$, $r_{23} = 2$.

$$c(3,4) = \min \begin{cases} K=3 \Rightarrow c(3,3)+c(4,4)+\sum_{q=1}^{j} P_q = 0.05+0.2+0.05+0.2 = 0.25+0.25 = 0.5 \\ K=4 \Rightarrow c(3,3)+c(5,4)+\sum_{q=1}^{j} P_q = 0.05+0+0.25 = 0.3 \end{cases}$$

$\therefore c(3,4) = 0.3$, $r_{34} = 4$

$$c(4,5) = \min \begin{cases} K=4 \Rightarrow c(4,3)+c(5,5)+\sum_{q=i}^{j} P_q = 0+0.3+0.2+0.3 = 0.3+0.5 = 0.8 \\ K=5 \Rightarrow c(4,4)+c(6,5)+\sum_{q=i}^{j} P_q = 0.2+0+0.5 = 0.7 \end{cases}$$

$\therefore c(4,5) = 0.7$, $r_{45} = 5$.

$$c(1,3) = \min \begin{cases} K=1 \Rightarrow c(1,0)+c(2,3)+\sum_{q=i}^{j} P_q = 0+0.3+0.25+0.2+0.05 = 0.3+0.5 = 0.8 \\ K=2 \Rightarrow c(1,1)+c(3,3)+\sum_{q=i}^{j} P_q = 0.25+0.05+0.5 = 0.8 \\ K=3 \Rightarrow c(1,2)+c(4,3)+\sum_{q=i}^{j} P_q = 0.65+0+0.5 = 1.15 \end{cases}$$

$\therefore c(1,3) = 0.8$, $r_{13} = 1$

$$c(2,4) = \min \begin{cases} K=2 \Rightarrow c(2,1)+c(3,4)+\sum_{q=i}^{j} P_q = 0+0.3+0.2+0.05+0.2 = 0.3+0.45 = 0.75 \\ K=3 \Rightarrow c(2,2)+c(4,4)+\sum_{q=i}^{j} P_q = 0.2+0.2+0.45 = 0.85 \\ K=4 \Rightarrow c(2,3)+c(5,4)+\sum_{q=i}^{j} P_q = 0.3+0+0.45 = 0.75 \end{cases}$$

$\therefore c(2,4) = 0.75$, $r_{24} = 2$.

$$c(3,5) = \min \begin{cases} k=3 \Rightarrow c(3,2) + c(4,5) + \sum_{s=i}^{j} P_s = 0 + 0.7 + 0.05 + 0.2 + 0.3 = 0.7 + 0.55 = 1.25 \\ k=4 \Rightarrow c(3,3) + c(5,5) + \sum_{s=i}^{j} P_s = 0.05 + 0.3 + 0.55 = 0.9 \\ k=5 \Rightarrow c(3,4) + c(6,5) + \sum_{s=i}^{j} P_s = 0.3 + 0 + 0.55 = 0.85 \end{cases}$$

$\therefore c(3,5) = 0.85, \quad r_{35} = 5.$

$$c(1,4) = \min \begin{cases} k=1 \Rightarrow c(1,0) + c(2,4) + \sum_{s=i}^{j} P_s = 0 + 0.75 + 0.25 + 0.2 + 0.05 + 0.2 = 0.75 + 0.7 = 1.45 \\ k=2 \Rightarrow c(1,1) + c(3,4) + \sum_{s=i}^{j} P_s = 0.25 + 0.3 + 0.7 = 1.25 \\ k=3 \Rightarrow c(1,2) + c(4,4) + \sum_{s=i}^{j} P_s = 0.65 + 0.2 + 0.7 = 1.55 \\ k=4 \Rightarrow c(1,3) + c(5,4) + \sum P_s = 0.8 + 0 + 0.7 = 1.5 \end{cases}$$

$c(1,4) = 1.25, \quad r_{14} = 2.$

$$c(2,5) = \min \begin{cases} k=2 \Rightarrow c(2,4) + c(3,5) + \sum_{s=i}^{j} P_s = 0 + 0.85 + 0.2 + 0.05 + 0.2 + 0.3 = 0.85 + 0.75 = 1.6 \\ k=3 \Rightarrow c(2,2) + c(4,5) + \sum_{s=i}^{j} P_s = 0.2 + 0.7 + 0.75 = 1.65 \\ k=4 \Rightarrow c(2,3) + c(5,5) + \sum P_s = 0.3 + 0.3 + 0.75 = 1.35 \end{cases}$$

$\therefore c(2,5) = 1.35, \quad r_{25} = 4$

$$c(1,5) = \min \begin{cases} k=1 \Rightarrow c(1,0) + c(2,5) + \sum_{s=i}^{j} P_s = 0 + 1.35 + 0.25 + 0.2 + 0.05 + 0.2 + 0.3 = 1.35 + 1 = 2.35 \\ k=2 \Rightarrow c(1,1) + c(3,5) + \sum_{s=i}^{j} P_s = 0.25 + 0.85 + 1 = 2.1 \\ k=3 \Rightarrow c(1,2) + c(4,5) + \sum P_s = 0.65 + 0.7 + 1 = 2.35 \\ k=4 \Rightarrow c(1,3) + c(5,5) + \sum P_s = 0.8 + 0.3 + 1 = 2.1 \\ k=5 \Rightarrow c(1,4) + c(6,5) + \sum P_s = 1.25 + 0 + 1 = 2.25 \end{cases}$$

$\therefore c(1,5) = 2.1, \quad r_{15} = 2$

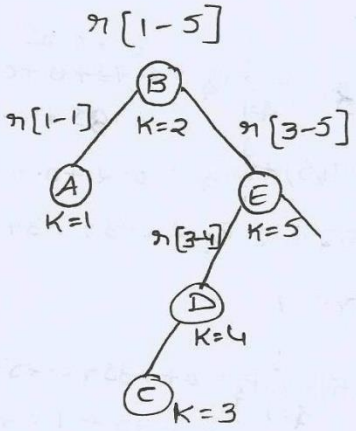
∴ The final tables are :

main table

	0	1	2	3	4	5
1	0	0.25	0.65	0.8	1.25	2.1
2		0	0.2	0.3	0.75	1.35
3			0	0.05	0.3	0.85
4				0	0.2	0.7
5					0	0.3
6						0

root table

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



∴ Search cost of the tree is

$2(0.25) + 1(0.2) + 4(0.05) + 3(0.2) + 2(0.3) = 2.1$