UNIT - III

Greedy Method: General Method, Job Sequencing with deadlines, Knapsack Problem, Minimum cost spanning trees, Single Source Shortest Paths.

Dynamic Programming: General Method, All pairs shortest paths.0/1 Knapsack, String Editing, Travelling Salesperson problem, Optimal Binary Search Trees.

Greedy Method:

The greedy method is used to solve problems with 'n' inputs, where the goal is to select a subset that satisfies specific constraints. Any subset that meets these constraints is called a feasible solution. The objective is to find the optimum feasible solution—the one that optimizes a given objective function.

General Method:

The Greedy method proposes dividing the algorithm into stages, where each stage considers one input at a time. At each stage a decision is made regarding whether or particular input is in the optimum solution. If the inclusion of next input to the partially constructed solution will result in an infeasible solution, then the input will not be considered and will not be added to partially constructed set otherwise it is added.

Key Characteristics of Greedy Algorithms:

1. **Locally Optimal Choice**: At each step, the algorithm chooses the best available option i.e. it focuses on what seems to be the best solution at the current moment.
2. **Irrevocable Decisions**: Once a choice is made, it cannot be undone. The algorithm moves forward based on the current choice i.e. once a choice is made, it cannot be undone.
3. **Optimal Substructure**: In this local choices must lead to the optimal overall solution.
4. **Feasibility and Constraint Satisfaction**: The chosen steps must satisfy the problem's constraints, ensuring the solution is feasible.

Steps in a Greedy Algorithm:

1. Define a candidate set: It defines the list of choices that an algorithm can choose in each step.
2. Selection: In this step the best option based on a specific rule is selected.
3. Feasibility Check: In this step, it checks whether inclusion of choice leads to valid solution or not. .
4. Solution Construction: In this step the valid solution is added to the current solution.
5. Repeat the same process until the solution is complete.

Control Abstraction

Algorithm Greedy (a,n)

```
{
    Solution: = 0;
     for i: = 1 to n do
     {
                        x: = select (a);
                         if feasible (solution, x) then
                        solution: = Union (solution, x);
     }
                        return solution;
}
```

Select is a function which is used to select an input from the set. Feasible is a function which verifies the constraints and determines whether the resultant solution is feasible or not. Union is a function which is used to add elements to the partially constructed set.

Advantages:

- Simple to understand
- Easy to implement
- Efficient in terms of time complexity

Disadvantages:

- May not always yield optimal solution for all problems.

Applications:

- Job Scheduling with Deadlines
- Fractional Knapsack
- Minimum Cost Spanning Trees
- Single Source Shortest Path
- Optimal Merge Pattern, etc

Job Sequencing With Deadlines:

The Job Sequencing with Deadlines problem is an optimization problem where a set of jobs is given, each with:

- A profit associated with completing the job.
- A deadline before which the job must be completed.

The objective is to schedule jobs in such a way that the total profit is maximized, ensuring that each job is completed by its deadline.

Steps:

1. **Sort Jobs by Profit**: Sort all jobs in decreasing order of their profits.
2. **Find Slots for Each Job**: For each job, starting with the one with the highest profit:
    - Find a slot before its deadline where it can be scheduled.
    - If such a slot exists, assign the job to that slot otherwise skip the job.
3. **Maximize Profit**: By choosing jobs based on their profits and scheduling them within their deadlines, the algorithm ensures maximum profit.

Algorithm Greedy Job (d,j,n)

```
{
    j: = {1};
    for i: = 2 to n do
    {
            if (all jobs in j u {i} can be completed by their deadlines) then
            j: = ju{ i};
     }
}
```

Ex: - Obtain the optimal sequence for the following jobs.

		j_1 j_2 j_3 j_4
(P_1, P_2, P_3, P_4)	=	(100, 10, 15, 27)
(d_1, d_2, d_3, d_4)	=	(2, 1, 2, 1)

n =4

Steps:

- Arrange all the jobs in decreasing order of their profit.

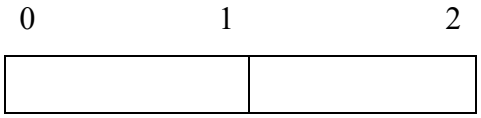
  (P_1, P_2, P_3, P_4) = (100, 10, 15, 27)

  (d_1, d_2, d_3, d_4) = (2, 1, 2, 1)

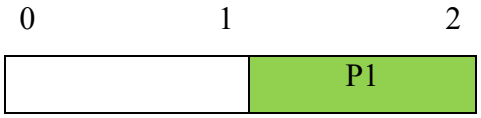
- Generate the slots by considering the maximum deadline.

  0 to 1 - One slot

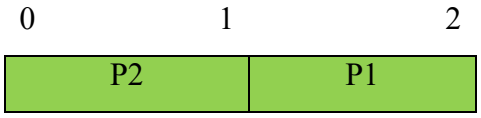
  1 to 2 - Second slot



- Schedule the jobs based on their deadlines.
  - The highest profit job is P1 and its deadline is 2, it is placed in 2nd slot.



  - Next highest profit job is P2, placed in 1st slot.



  - There is no more space for 3rd and 4th slots.
- Calculate the profit by adding profits of jobs, which are scheduled successfully.

  Profit P= 100+27 = 127

Time Complexity:

- Worst Case : $O(n^2)$
- Best and Average Case : O(n)

Exercise:

1. **What is the solution generated by the function JS when n = 7,**
   P [1 : 7] = (3,5,20,18,1,6,30) and W [1:7] = (1,3,4,3,2,1,2)
2. **Profit : 20 15 10 5 1 Deadlines: 4 1 3 2 1 n= 5.**
3. **Profit : 70 12 18 35 Deadlines: 2 1 2 1 n= 4.**
4. **Profit: 20 10 40 30 Deadlines: 2 1 1 1**
5. **Profit: 100 19 27 25 15 Deadlines: 2 1 2 1 3**
6. **Profit:1, 5, 20, 15, 10 Deadlines: 1, 2, 4, 1, 3**
7. **P[1:9]= [85,25,16,40,55,19,92,80,15] and D[1:7]= [5,4,3,3,4,5,2,3,7]**

Knapsack Problem (Fractional Knapsack):

A knapsack with capacity 'm' is given in to which we are required to place certain weights such that the sum of the weights will not exceed the knapsack capacity. Associated with each weight we have associated profit which will be earned by the inclusion of the object in to the knapsack.

If it is not possible to include an object entirely a fraction of the object can be included and accordingly a fraction of the profit is earned.

Given 'n' objects and a bag of capacity 'm' and each object 'i' has a profit p_i and weight w_i associated with it. If a fraction x_j $(0 \leq x, \leq 1)$ of the object i is placed in the bag. A profit pi x xi is made. The objective in the problem is to obtain the maximum profit by filling the bag with given objects.

The knapsack problem can be stated mathematically as follows.

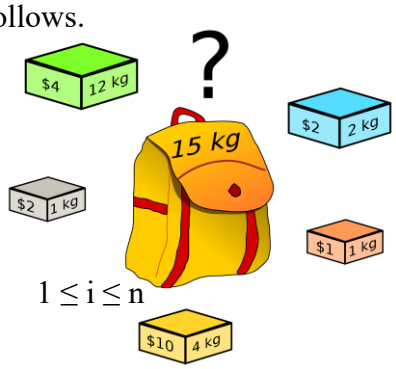
Maximize $\sum P_i x_i$

$1 \leq i \leq n$

Subject to

$\sum W_i x_i \leq m$

$1 \leq i \leq n$ and $0 \leq xi \leq 1$, $1 \leq i \leq n$



The profits and weights are positive numbers.

Algorithm Greedy knapsack (m, n)

```
{
    for i: = 1 to n do x(i) = 0.0i// initialize x
    u : = m;
    for i: = 1 to n do
    {
        if (w(i) > u) then break;
        x(i): = 1.0; u: = u-w(i);
    }
    if (i≤n) then x(i): = u/w(i);
}
```

Analysis: - If we do not consider the time for sorting the inputs then all of the three greedy strategies complexity will be O(n).

Ex: - Consider 3 objects whose profits and weights are defined as

$(P_1, P_2, P_3) = (25, 24, 15)$

$(W_1, W_2, W_3) = (18, 15, 10)$

$n=3 \qquad m=20$

Find the optimal solution for,

(i) Maximum profit

(ii) Minimum weight

(iii) Maximum profit per unit weight.

Solution:

Consider a knapsack of capacity 20. Determine the optimum strategy for placing the objects in to the knapsack. The problem can be solved by the greedy approach where in the inputs are arranged according to selection process (greedy strategy) and solve the problem in stages.

(i) Maximum Profit: In this case we will place an item in the bag whose profit is maximum.

X1= 1 complete item w1 is kept in the bag then 2 units place is left in the bag. (20-18).

X2 = out of space so weight or item to be placed = 2/15.

When w2 is places no space is left in the bag. So we cannot place the 3rd item.

$$\sum p_i x_i = p_1x_1+p_2x_2+p_3x_3$$
$$= 25 \times 1 + 24 \times 2/15 + 15 \times 0$$
$$= 25 + 3.2 + 0$$
$$= 28.2$$

(ii) **Minimum Weight:** In this case we will place an item in bag, whose weight is minimum.

X3 = 1

X2 = 10/ 15

X1= 0

$$\sum p_i x_i = p_1x_1+p_2x_2+p_3x_3$$
$$= 25 \times 0 + 24 \times 10/15 + 15 \times 1$$
$$= 0 + 16 + 15$$
$$= 31$$

(iii) **Maximum profit per unit weight:** In this case we will place an item in the bag, whose profit per unit ratio is maximum.

X1 = p1/w1 = 28/18 = 1.4

X2 = p2/w2 = 24/15 = 1.6

X3 = p3/w3 = 15/10 = 1.5

Maximum profit per unit is obtained by placing x2 in the bag. So,

X2 = 1

X3 = 5/10

X1 = 0

$$\sum p_i x_i = p_1x_1+p_2x_2+p_3x_3$$
$$= 25 \times 0 + 24 \times 1 + 15 \times 5/10$$
$$= 0 + 24 + 7.5$$
$$= 31.5$$

If objects are selected in increasing order of pi/wi, proves that the algorithm finds an optimal solution.

Exercise:

1. Find the optimal solution of the Knapsack instance n= 7, M=15, (p1,p2,….p7) = (10,5,15,7,6,18,3) and (w1,w2,…..w7)=(2,3,5,7,1,4,1)
2. Weight={10,30,20,50} Profit={40,30,80,70}, M=60
3. Weight={5,10,15,22,25} Profit={30,40,45,77,90}, M=60

Minimum Spanning Trees:

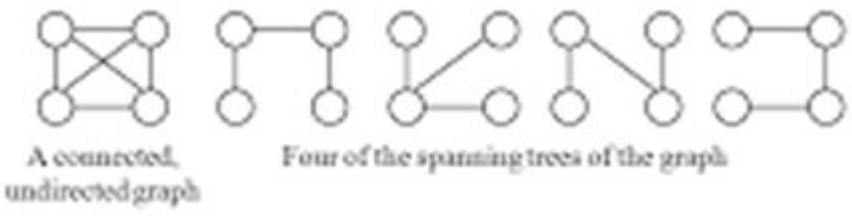
Spanning Tree:

Let G = (V, E) be an undirected connected graph. A sub-graph t = (V, E`) of G is a spanning tree of G if and only if t is a tree. A Subgraph 't' of a graph 'G' is called as a spanning tree if

(i) It includes all the vertices of 'G'

(ii) It is a tree

For a given graph 'G' there can be more than one spanning tree.



The graph with 'n' vertices, the number of spanning trees generated is 2nCn / n+1.

Minimum Cost Spanning Trees:

A tree is to be defined as an undirected, acyclic and connected graph. A graph has more than one spanning trees. A minimum spanning tree is a spanning tree, that has weights with the edges and the total weight of the tree is less. The minimum spanning tree is obtained by the use of two standard algorithms. They are;

1. Prim's Algorithm
2. Kruskal's Algorithm

Prim's Algorithm:

Prim's algorithm is a greedy algorithm that finds minimum cost spanning tree for a weighted undirected graph.

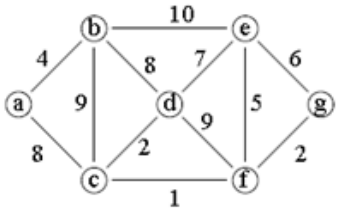
Steps:

1. Begin with any vertex which you think would be suitable and add to the tree.
2. Find an edge that connects to any vertex in the tree to any vertex that is not in the tree. Note that we don't have to form cycles and connect the vertex which has minimum weight.
3. Stop when n-1 edges have been added to the tree.

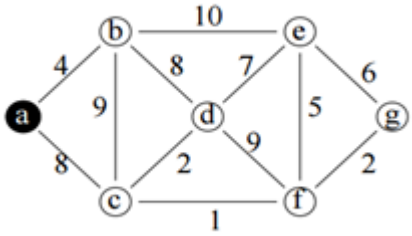
Analysis: -

The time required by the prince algorithm is directly proportional to the no. of vertices. If a graph 'G' has 'n' vertices then the time required by prim's algorithm is $0(n^2)$

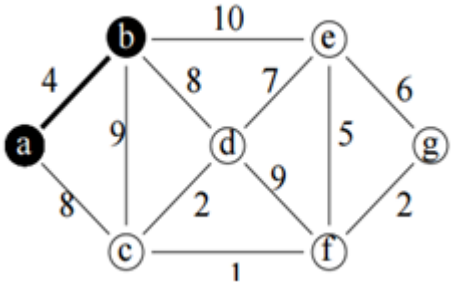
Ex: Consider the following graph and draw the minimum spanning tree.



Step1: Initially all vertices are unvisited i.e. S is empty.



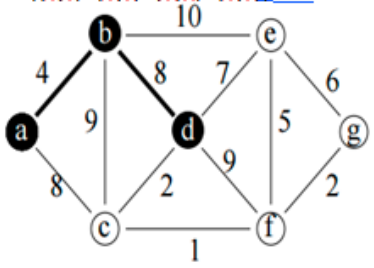
Step2: Now **a** contains two vertices b and c as neighbors. Among these edges distance from **a to b** is minimum. So include (a,b) into minimum spanning tree. Now S={a, b}. Repeat the same procedure until spanning tree contains (n-1) edges.



min((a,b) (a,c))

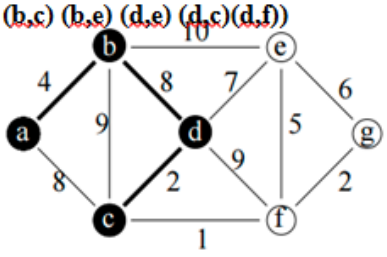
S={a, b}

3. min ((a,c) (b,c) (b,d) (b,e))



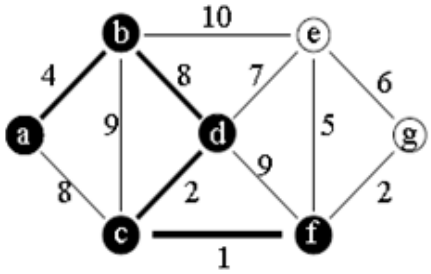
S= {{a, b} {b, d}}

4. min ((a,c), (b,c) (b,e) (d,e) (d,c)(d,f))



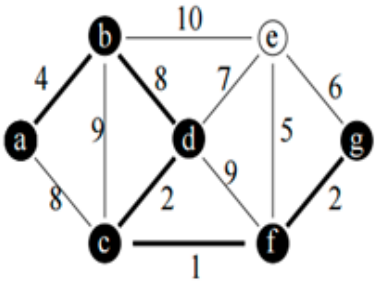
S= {{a, b} {b, d} {d, c}}

5. min ((b,e) (d,e) (d,f)(c,f))



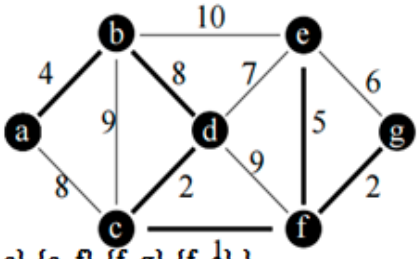
S= {{a, b} {b, d} {d, c} {c,f} }

6. min ((b,e) (d,e) (f,g), (f,e))



S= {{a, b} {b, d} {d, c} {c, f} {f, g}}

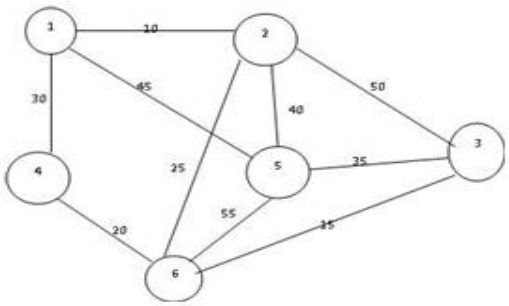
7. min ((b,e) (d,e) (f,e) (g,e))

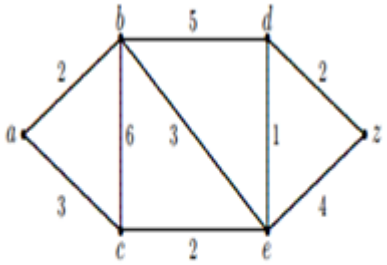


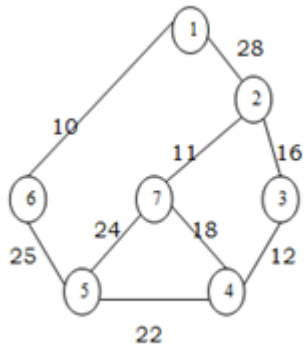
S= { {a, b} {b, d} {d, c} {c, f} {f, g} {f, e} }

Total Cost = 4 + 8 + 2+ 1 + 2 + 5 = 22

Exercise:







Kruskal's Algorithm:

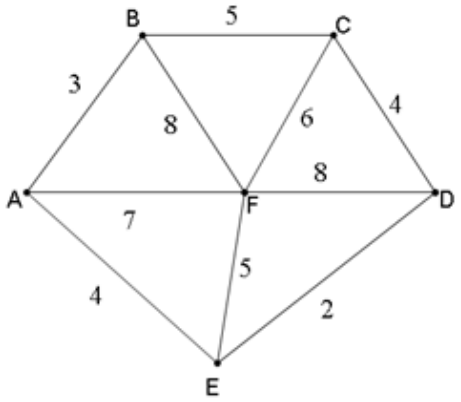
This algorithm constructs the minimum spanning tree of a graph by adding edges to the spanning tree one-by-one. This algorithm is conceptually quite simple. The edges are selected and added to the spanning tree in increasing order of their weights. An edge is added to the tree only if it does not create a cycle.

Steps:

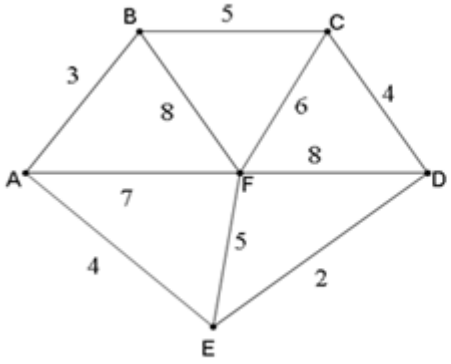
1. Arrange the edges by weight: least weight first and heaviest last.
2. Choose the lightest not examined edge from the diagram. Add this chosen edge to the tree, only if doing so will not make a cycle.
3. Stop the process whenever n-1 edges have been added to the tree.

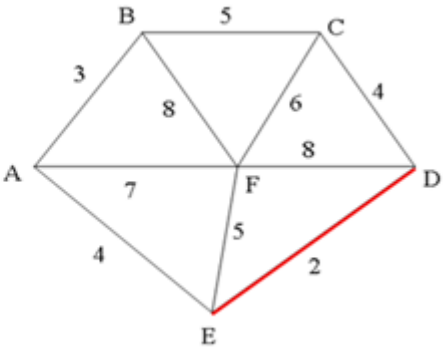
Analysis: - If the no: of edges in the graph is given by /E/ then the time for Kruskals algorithm is given by 0 (|E| log |E|).

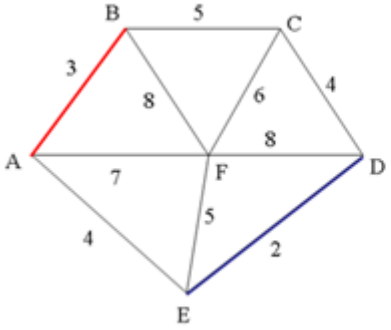
Ex:

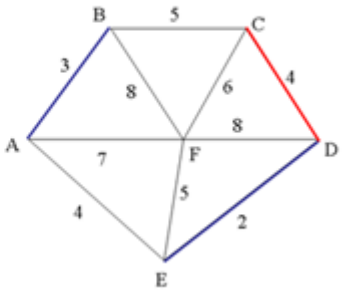


Edge	ED	AB	AE	CD	~~BC~~	EF	~~CF~~	~~AF~~	~~BF~~	~~FD~~
Weight	2	3	4	4	5	5	6	7	8	8

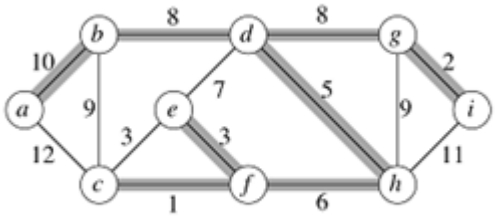


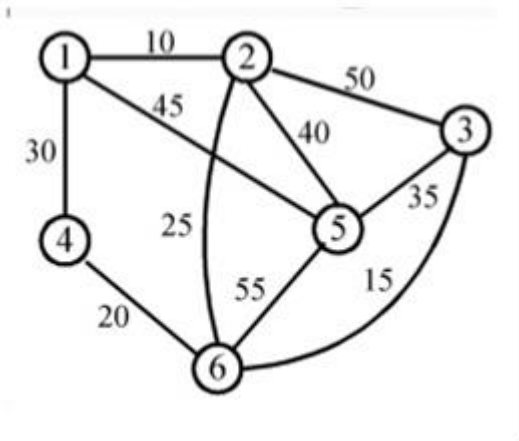


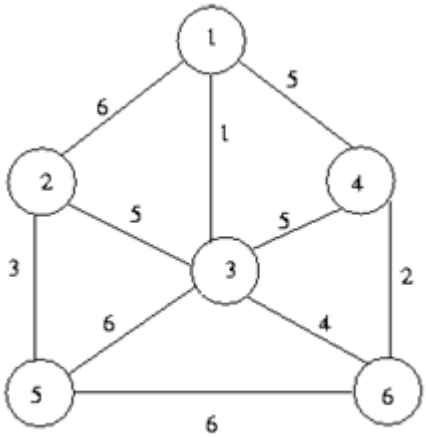




Exercise:







Differences between Prim's and Kruskal's algorithms:

S.No.	Prim's	Kruskal's
1	Select any vertex.	Select the shortest edge in a network.
2	Select the shortest edge connected to that vertex.	Select the next shortest edge which does not create a cycle.
3	Select the shortest edge connected to any vertex already connected.	Repeat step 2 until all vertices have been connected.
4	Repeat step 3 until all vertices have been connected.	

Single Source Shortest Path (Dijkstra's shortest path):

In single source shortest path problem, the shortest distance from a single vertex is obtained. Let G(V,E) be a graph, then in single source shortest path problem, the shortest paths from vertex s to all remaining vertices is determined. The vertex S is then called a source.

Dijkstra's algorithm finds the shortest paths from a given node s to all other nodes of a weighted graph with positive weights. Node s is called a starting node.

Algorithm:

This algorithm follows Greedy approach that solves the single source shortest path problem for a directed graph with non-negative edge weights. Each node is labeled with its distance from the source node along with the best-known path. Initially all nodes are labeled with infinity since no paths are known initially. The algorithm will go on in simple steps. As algorithm processed, the paths would find, the labels will change, and reflects the better paths. The following is the stepwise procedure of algorithm.

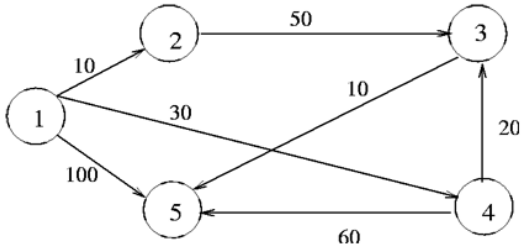
1. **Source node is initialized and can be indicated as a filled circle.**
2. **Initial path lost to neighboring nodes or link cost is computed and these nodes are re-labeled considering source node.**
3. **Examine all the adjacent nodes and find the smallest label and make it permanent.**
4. **The smallest label node is now working node, then step 2 & 3are repeated till the destination node reaches.**

Complexity of Dijkstra's Algorithm

With adjacency matrix representation, the running time is $O(n^2)$ By using an adjacency list representation and a partially ordered tree data structure for organizing the set V - S, the complexity can be shown to be O(elog n)

Where e is the number of edges and n is the number of vertices in the digraph.

Ex:



Initially:

$S = \{1\}$ $D[2] = 10$ $D[3] = \infty$ $D[4] = 30$ $D[5] = 100$

Iteration 1

Select w = 2, so that S = {1, 2}

$D[3] =$ $\min(\infty, D[2] + C[2, 3]) = 60$

$D[4] =$ $\min(30, D[2] + C[2, 4]) = 30$

$D[5] =$ $\min(100, D[2] + C[2, 5]) = 100$

Iteration 2

Select w = 4, so that S = {1, 2, 4}

$D[3] =$ $\min(60, D[4] + C[4, 3]) = 50$

$D[5] =$ $\min(100, D[4] + C[4, 5]) = 90$

Iteration 3

Select w = 3, so that S = {1, 2, 4, 3}

$D[5] =$ $\min(90, D[3] + C[3, 5]) = 60$

Iteration 4

Select w = 5, so that S = {1, 2, 4, 3, 5}

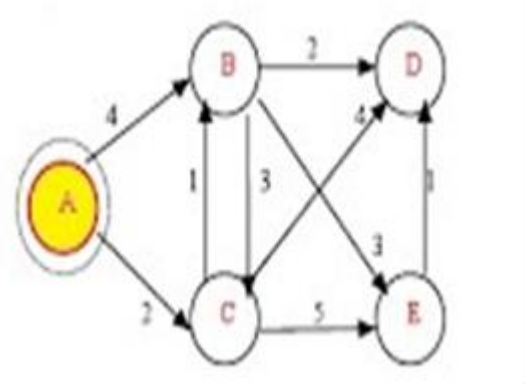
$D[2] = 10$

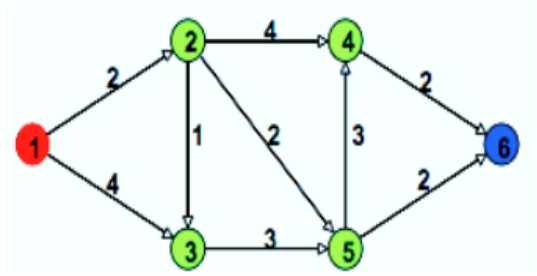
$D[3] = 50$

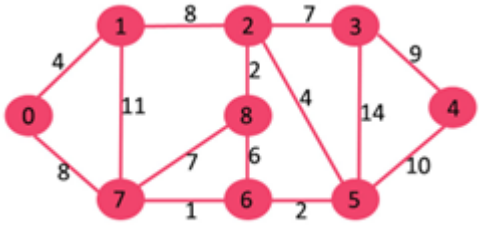
$D[4] = 30$

$D[5] = 60$

Exercise:


A
B
C
D
E
4
2
1
3
4
3
3
5


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


0
1
2
3
4
5
6
7
8
4
8
11
7
2
4
14
6
9
10
1
2

Dynamic Programming: General Method, All pairs shortest paths.0/1 Knapsack, String Editing, Travelling Salesperson problem, Optimal Binary Search Trees.

Dynamic Programming:

Dynamic programming is used to solve optimization problems by following principle of optimality. Principle of optimality says that a problem can be solved by taking a decision in each stage. The essential characteristic of dynamic programming is the multistage nature of the optimization procedure. It is applicable when the sub-problems are not independent.

General Method:

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

Applications of the dynamic programming.

1. Matrix Chain Multiplication
2. 0/1 Knapsack
3. Multistage graphs
4. All-pairs shortest path
5. Single source shortest path
6. Optimal binary search trees
7. String editing
8. Reliability design
9. The travelling salesman problem
10. Flow shop scheduling

All Pairs Shortest Paths:

In all pairs shortest path problem, we have to find the shortest path between every pair of vertices of a graph. The graph may contain negative edges. This procedure requires $O(n^3)$ time.

Weighted Graph:

The weighted graph is a graph in which weights or distances are given along the edges.

W[i][j]= 0 if i=j

W[i][j]= ∞ if there is no edge between i and j.

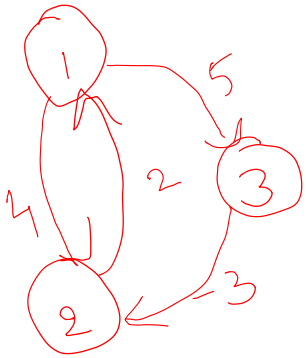
W[i][j]= weight of the edge

Floyd-Warshall algorithm is used for solving the all pairs shortest path problem. The problem is to find shortest distance between every pair of vertices in a given edge weighted directed graph. The shortest path can be computed using the following recursive method.

$A^K(i,j) = \min \{ A^{k-1}(i,j), [A^{k-1}(i,k) + A^{k-1}(k,j)] \}$

Where $i <= k <= n$

Ex:



$D^{(0)}$ =

	1	2	3
1	0	4	5
2	2	0	∞
3	∞	-3	0

For $D^{(1)}$, k=1:

$D^1(1, 2) = \min \{ D^0(i,j), (D^0(I,k)+D^0(k,j) \}$

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

$= \min \{4, (0+4) \} = 4$

$D^1(1, 3) = \min \{ D^0(1,3), (D^0(1,1)+D^0(1,3) \}$
$= \min \{5, (0+5) \} = 5$
$D^1(2, 1) = \min \{ D^0(2,1), (D^0(2,1)+D^0(1,1) \}$
$= \min \{2, (2+0) \} = 2$
$D^1(2, 3) = \min \{ D^0(2,3), (D^0(2,1)+D^0(1,3) \}$
$= \min \{\infty, (2+5) \} = 7$
$D^1(3, 1) = \min \{ D^0(3, 1), (D^0(3,1)+D^0(1,1) \}$
$= \min \{\infty, (\infty+0) \} = \infty$
$D^1(3, 2) = \min \{ D^0(3, 2), (D^0(3,1)+D^0(1,2) \}$
$= \min \{-3, (\infty+4) \} = -3$

The matrix after first insertion is;

$\mathbf{D^{(1)}} =$

	1	2	3
1	0	4	5
2	2	0	7
3	∞	-3	0

k=2:

$D^2(1, 2) = \min \{ D^1(i,j), (D^1(I,k)+D^1(k,j) \}$
$= \min \{ D^0(1,2), (D^1(1,2)+D^1(2,2) \}$
$= \min \{4, (0+4) \} = 4$
$D^2(1, 3) = \min \{ D^1(1,3), (D^1(1,2)+D^1(2,3) \}$
$= \min \{5, (4+7) \} = 5$
$D^2(2, 1) = \min \{ D^1(2,1), (D^1(2,2)+D^1(2,1) \}$
$= \min \{2, (0+2) \} = 2$
$D^2(2, 3) = \min \{ D^1(2,3), (D^1(2,2)+D^1(2,3) \}$
$= \min \{7, (0+7) \} = 7$
$D^2(3, 1) = \min \{ D^1(3, 1), (D^1(3,2)+D^1(2,1) \}$
$= \min \{\infty, (-3+2) \} = -1$
$D^2(3, 2) = \min \{ D^1(3, 2), (D^1(3,2)+D^1(2,2) \}$
$= \min \{-3, (-3+0) \} = -3$

$\mathbf{D^{(2)}}$ =

	1	2	3
1	0	4	5
2	2	0	7
3	-1	-3	0

k=3:

$D^3(1, 2) = \min \{ D^2(i,j), (D^2(I,k)+D^2(k,j) \}$

$= \min \{ D^2(1,2), (D^2(1,3)+D^2(3,2) \}$

$= \min \{4, (5+(-3)) \} = 2$

$D^3(1, 3) = \min \{ D^2(1,3), (D^2(1,3)+D^2(3,3) \}$

$= \min \{5, (5+0) \} = 5$

$D^3(2, 1) = \min \{ D^2(2,1), (D^2(2,3)+D^2(3,1) \}$

$= \min \{2, (7+(-1)) \} = 2$

$D^3(2, 3) = \min \{ D^2(2,3), (D^2(2,3)+D^2(3,3) \}$

$= \min \{7, (7+0) \} = 7$

$D^3(3, 1) = \min \{ D^2(3, 1), (D^2(3,3)+D^2(3,1) \}$

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

$D^3(3, 2) = \min \{ D^2(3, 2), (D^2(3,3)+D^2(3,2) \}$

$= \min \{-3, (0+(-3)) \} = -3$

$\mathbf{D^{(3)}}$ =

	1	2	3
1	0	2	5
2	2	0	7
3	-1	-3	0

Advantages:

The shortest path problem is defined on a directed, weighted graph, where the weights may be thought if as distances. The object is to find a path from a source node to destination node with a minimum cost.

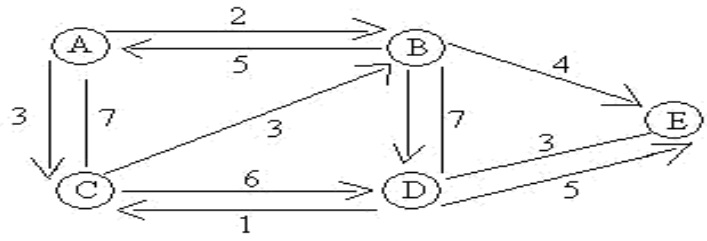
Application Areas:

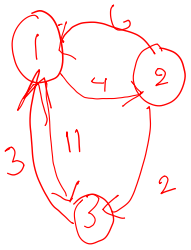
- ➔ Robot path planning
- ➔ Link state routing protocols
- ➔ Open Shortest Path First

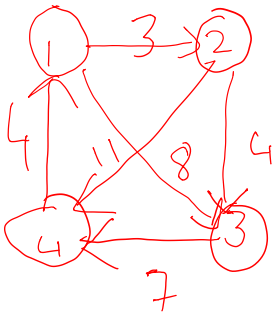
- ➔ Saving network resource
- ➔ Two metric routing problem
- ➔ Routing information protocol
- ➔ Optimal routing

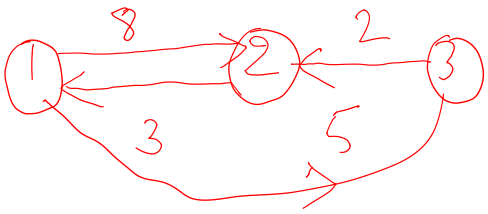
Exercise:

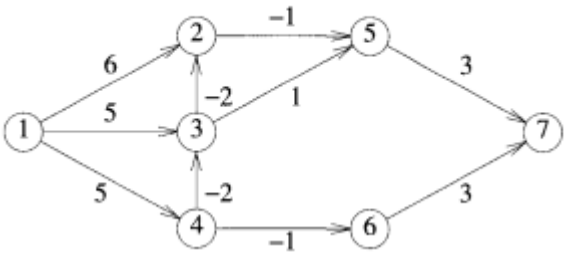
Find the shortest path b/w all pairs of nodes in the following graph and explain with the suitable algorithm











0/1 Knapsack:

A knapsack with capacity 'm' is given into which we are required to place certain weights such that the sum of the weights will not exceed the knapsack capacity. Each item is associated with a weight and a profit, will be earned by the inclusion of the object into the knapsack. There are two versions of the problem.

1. Fractional knapsack, where the items are divisible. This will be solved by Greedy Approach.
2. 0/1 Knapsack, where the items are indivisible. In this you can either take an item or not. This will be solved by Dynamic Programming.

0/1 Knapsack problem:

This problem contains either 0 or 1. 0 means we cannot place the item in the bag, 1 means we can place the item in the bag. Here, fractions of item can't be placed in the bag. Consider weights W1, W2,…, Wn and fraction of weights to be put in the bag should be X1, X2 …, Xn= 0 or 1. The dynamic programming solution for 0/1 Knapsack is;

$$Fn(M) = Max \{ fn-1(M), fn-1(M-Wn) +pn\}$$

Where, M is size of the bag.

When Xn = 1 then the size of the bag is reduced by Wn. As we placing the nth item in the bag, we should add the profit.

When an item is removed from the knapsack, then the corresponding weight is added to the capacity of the knapsack and profit of the item is reduced from the earned profit.

Ex:

N=4 M=7

(W1, W2, W3, W4) = (1,3,4,5)

(P1, P2, P3,P4) = (1, 4, 5,7)

Profit-Weight	0	1	2	3	4	5	6	7
1 – 1	0	1	1	1	1	1	1	1
4 – 3	0	1	1	4	5	5	5	5
5 – 4	0	1	1	4	5	6	6	9
7 – 5	0	1	1	4	5	7	8	9

1st Row: Weight of the first item is 1 and profit is 1 so we place the profit of the first item in the entire row.

2nd Row:

The weight of the 2nd item is 3 so we cannot place it in 1 kg or 2 kg. So the previous profit is placed in these columns.

3rd Column: It holds 3KGs. The third item will have a profit of 4. It is better than the previous profit. So place the profit 4.

$$V(i, w) = \max\{V[i-1,w], V[i-1, w-wi]+pi\}$$

V(2,3) = max{V[2-1, 3], V[2-1, 3-3]+4}
= max{V[1,3], V[1,0]+4}
= max{1, 0+4}
= 4

V(2,4) = max{V[1,4], V[1,1]+4}
= max{1, 1+4}
= 5

V(2,5) = max{V[1,5], V[1,2]+4}
= max{1, 1+4}
= 5

V(2,6) = max{V[1,6], V[1,3]+4}
= max{1, 1+4}
= 5

V(2,7) = max{V[1,7], V[1,4]+4}
= max{1, 1+4}
= 5

3rd Row: Weight of the third item is : 4 and profit is 5. So up to 3 columns the profit remains same as previous columns.

V(3,4) = max{V[2,4], V[2,0]+5}
= max{5, 0+5}
= 5

V(3,5) = max{V[2,5], V[2,1]+5}
= max{5, 1+5}
= 6

V(3,6) = max{V[2,6], V[2,2]+5}
= max{5, 1+5}
= 6

V(3,7) = max{V[2,7], V[2,3]+5}
= max{5, 4+5}
= 9

4th Row: Weight of the 4th item is: 5 and profit is 7. So up to 4 columns the profit remains same as previous.

V(4,5) = max{V[3,5], V[3,0]+7}
= max{6, 0+7}
= 7

V(4,6) = max{V[3,6], V[3,1]+7}
= max{6, 1+7}
= 8

V(4,7) = max{V[3,7], V[3,2]+7}
= max{9, 1+7}
= 9

- So the maximum profit is 9. Check it towards up until end. It will close at 3rd row. The value is changed at 3rd, which will clearly indicates that 3rd item is included. So the **item is 3 and profit is 5.**

  X3 =1 and X4 = 0 and Earned Profit = 5 and Knapsack Capacity = 7-4 = 3
- The remaining profit is 4 (9-5), check the profit 4 until it ends. It ends at 2nd row, which will clearly indicates that item 2 is included. So the **item is 2 and profit is 4**.

  X2=1, X3 =1 and X4 = 0

  Earned Profit = 5 + 4 = 9 and Knapsack Capacity = 3-3 = 0

Since the capacity of the knapsack is 0, item1 is not included in the knapsack.

X1=0, X2=1, X3 =1 and X4 = 0

Maximum Profit Earned = 9

Exercise:

1. N=3 (W_1, W_2, W_3)=(18, 15, 10,) . (P1, P2, P3)= (25, 24, 15) and m=20.
2. Array of profits is P= (11, 21, 31, 33) and array of weights are W=(2,11,22,15),Knapsack capacity is M=40 and number of items is n=4.
3. Solve the following 0/1 knapsack problem using dynamic programming M=6,n=3 (w1,w2,w3)=(2,3,3), (p1,p2,p3)=(1,2,4)
4. (w1,w2,w3)=(2,3,4), (p1,p2,p3)=(1,2,5), M=6, N=3

String Editing (Edit Distance):

String editing is used to find minimum number of edits required to convert one string to another string. Insert, delete and substitute are having equal cost. The time complexity is O(m x n).

Set of operations:

- Same character Cost is 0
- Delete character Cost is 1
- Insert character Cost is 1
- Substitute character Cost is 1

For example, given the strings *A = "cat"* and *B = "cars"*, *then the edit distance is 2* because the minimum number of transformations that we need to make is replace the "r" in B by "t" and then remove the "s" from *B*. After that, both strings are equal to "cat".

Ex:

String 1 : adceg String 2: abcfg convert string 1 as string 2.

	NULL	A	B	C	F	G
NULL	0	1	2	3	4	5
A	1	0	1	2	3	4
D	2	1	1	2	3	4
C	3	2	2	1	2	3
E	4	3	3	2	2	3
G	5	4	4	3	3	2

FIRST ROW:

Null is changed to

- NULL – No more operations are required
- A - Requires 1 insertion so the cost is 1
- AB - Requires 2 insertions so the cost is 2
- ABC - Requires 3 insertions so the cost is 3
- ABCF - Requires 4 insertions so the cost is 4
- ABCFG - Requires 5 insertions so the cost is 5

SECOND ROW:

A is changed to

- NULL – Requires 1 deletion so the cost is 1
- A - No more operations are required
- AB - Requires 1 insertion so the cost is 1
- ABC - Requires 2 insertions so the cost is 2
- ABCF - Requires 3 insertions so the cost is 3
- ABCFG - Requires 4 insertions so the cost is 4

THIRD ROW:

AD is changed to

- NULL – Requires 2 deletions so the cost is 2
- A - Requires 1 deletion so the cost is 1
- AB - Requires 1 substitution so the cost is 1
- ABC - Requires 1 substitution and 1 insertions so the cost is 2
- ABCF - Requires 1 substitution and 2 insertions so the cost is 3
- ABCFG - Requires 1 substitution and 3 insertions so the cost is 4

FOURTH ROW:

ADC is changed to

- NULL – Requires 3 deletions so the cost is 3
- A - Requires 2 deletions so the cost is 2
- AB - Requires 1 substitution and 1 deletion so the cost is 2
- ABC - Requires 1 substitution so the cost is 1
- ABCF - Requires 1 substitution and 1 insertions so the cost is 2
- ABCFG - Requires 1 substitution and 2 insertions so the cost is 3

FIFTH ROW:

ADCE is changed to

- NULL – Requires 4 deletions so the cost is 4
- A - Requires 3 deletions so the cost is 3
- AB - Requires 1 substitution and 2 deletions so the cost is 3
- ABC - Requires 1 substitution and 1 deletion so the cost is 2

- ABCF - Requires 2 substitutions so the cost is 2
- ABCFG - Requires 2 substitutions and 1 insertion so the cost is 3

SIXTH ROW:

ADCEG is changed to

- NULL – Requires 5 deletions so the cost is 5
- A - Requires 4 deletions so the cost is 4
- AB - Requires 1 substitution and 3 deletions so the cost is 4
- ABC - Requires 1 substitution and 2 deletions so the cost is 3
- ABCF - Requires 2 substitutions and 1 deletion so the cost is 3
- ABCFG - Requires 2 substitutions so the cost is 2

The final answer is 2 operations are required to change the string ADCEG to ABCFG. The two operations are obtained by backtracking from the answer node as follows;

1. Start with answer node and move towards up by taking diagonal values from the table. GG are the same so the no of operations required is zero.

   Total operations = 0

2. The EG is transformed to FG, which requires one substitute operation. So the required operation is 1.

   Total operations = 0 + 1 = 1

3. The CEG is transformed to CFG, which requires one substitute operation which was performed in step 2. So the required operation is 0.

   Total operations = 1 + 0 = 1

4. The DCEG is transformed to BCFG, which requires two substitute operations and one was performed in step 2. So the required operation is 1.

   Total operations = 1 + 1 = 2

5. The ADCEG is transformed to ABCFG, which requires two substitute operations and which was performed in step 2 and step 4 respectively. So the required operation is 0.

   Total operations = 2 + 0 = 2

Travelling Salesperson Problem:

The idea behind in TSP is find minimum weight route which starts from a vertex and then travels through every other vertex exactly once and reaches the starting vertex. It is similar to Hamiltonian cycle problem, but the difference is in Travelling Salesman Problem, we need to find a minimum weight Hamiltonian Cycle.

By Brute force method, TSP would be generate all possible permutations and finding which permutation gives the minimum cost will generates the duplicates. The time complexity for Brute force algorithm is n!.

Ex:

	1	2	3	4
1	0	2	9	10
2	1	0	6	4
3	15	7	0	8
4	6	3	12	0

Initial vertex is 1. If we don't have path from one vertex to another we can put ∞. So start the tour from 1, we have to go through 2, 3, 4 and again reach to 1.

The idea is pick one subset at a time and goes through every vertex in the graph and has to find the minimum cost starting from 1 to that particular vertex going through all the vertices.

$G(i, \Phi) = C_i$, 31<= i <=n

K=0

$g(2,\emptyset) = c21 = 1$

$g(3,\emptyset) = c31 = 15$

$g(4,\emptyset) = c41 = 6$

K=1, i.e. the set contains one element.

Set {2}= g(3, {2}) i.e. Reaching 2 via 3.

= c32 + g(2, ∅) =c32 + c21 = 7 + 1 = 8

Set {2}= g(4, {2}) i.e. Reaching 2 via 4.

= c42 + g(2, ∅) =c42 + c21 = 3 + 1 = 4

Set {3}= g(2, {3}) i.e. Reaching 3 via 2.

= c23 + g(3, ∅) =c23 + c31 = 6 + 15 = 21

Set {3}= g(4, {3}) i.e. Reaching 3 via 4.

= c42 + g(3, ∅) =c43 + c31 = 12 + 15 = 27

Set {4}= g(2, {4}) i.e. Reaching 4 via 2.

= c24 + g(4, ∅) =c24 + c41 = 4 + 6 = 10

Set {4}= g(3, {4}) i.e. Reaching 4 via 3.

= c34 + g(4, ∅) =c34 + c41 = 8 + 6 = 14

K=2 i.e. set contains two elements.

Set {2,3} = g(4,{2,3}) = min(c42+g(2,{3}), c43+g(3,{2})

= min(3+21, 12+8) = min(24, 20) = 25

Set {2,4} = g(3, {2,4}) = min(c32+ g(2, {4}), c34+g(4, {2})

=min(7+10, 8+4) = min(17,12) = 12

Set {3,4} = g(2, {3,4}) = min(c23+g(3,{4}) , c24+g(4, {3})

= min(6+14, 4+27) = min(20, 31) = 20

Length of an optimal tour is:

F(1, {2,3,4}) = min(c12+g(2, {3,4}), c13+g(3, {2,4}), c14+g(4,{2,3})

= min(2+20, 9+12, 10+20) = min (22,21, 30) = 21

Successor of node 1 : P(1, {2,3,4}) = 3 1→ 3

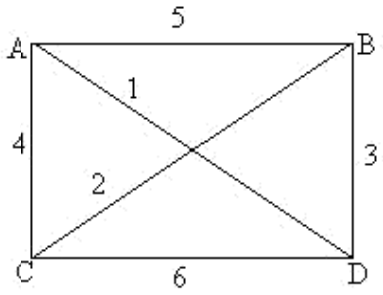
Successor of node 3 : P(3, {2,4}) = 4 1→3→4

Successor of node 4: P(4, {2}) = 2 1→3→4→2

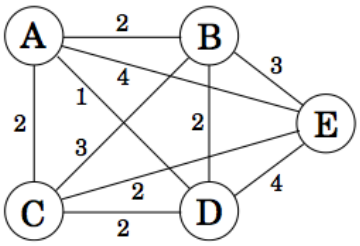
Successor of node 2: P(2, {∅ }) = 1 1→3→4→2→1

Exercise:

1. **Find the shortest tour of TSP for the following graph using dynamic programming**



2.



3.

	A	B	C
A	0	1	2
B	6	0	3
C	5	4	0

4.

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

Optimal Binary Search Trees (OBST):

OBST is a binary search tree which provides the minimum possible search time for a given sequence of access probabilities. The search time can be improved in OBST, placing the most frequesntly used data in the root and closer to the root element, while placing the least frequently used data near leaves and in leaves.

If 'n' keys are there then the number of possible binary search trees are; 2ncn / n+1.

Ex: Keys : 10 11 12

No. of keys = 3

No. of possible BSTs are = 2ncn / n+1

= 2*3 c 3 / 4

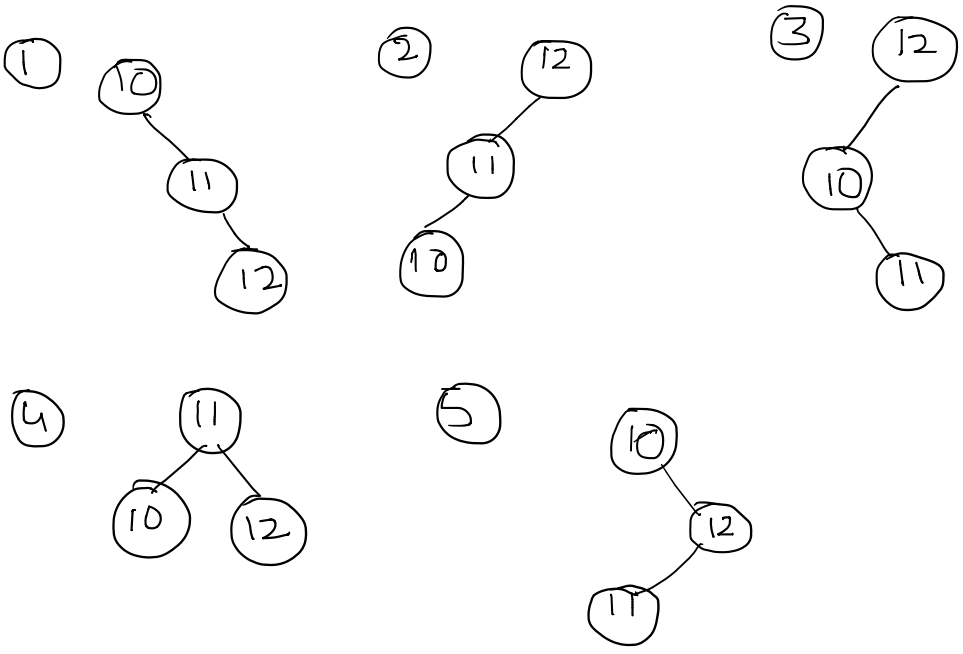
= 6C3 / 4

= (6!/3! * 3!) / 4

= 20/ 4

= 5

The possible trees are;



No. of comparisons for Tree 1 = 3 (Height of tree)

No. of comparisons for Tree 2 = 3 (Height of tree)

No. of comparisons for Tree 3 = 3 (Height of tree)

No. of comparisons for Tree 4 = 2 (Height of tree)

No. of comparisons for Tree 5 = 3 (Height of tree)

Tree 4 having less number of comparisons, hence it is optimal.

The keys are having probability for search and Search may have successful (internal nodes) or unsuccessful search (external node). The cost of a BST is calculated as;

$$c(T) = \sum_{1 \le i \le n} P(i) * level(ai) + \sum_{0 \le i \le n} q(i) * (level(Ei) - 1)$$

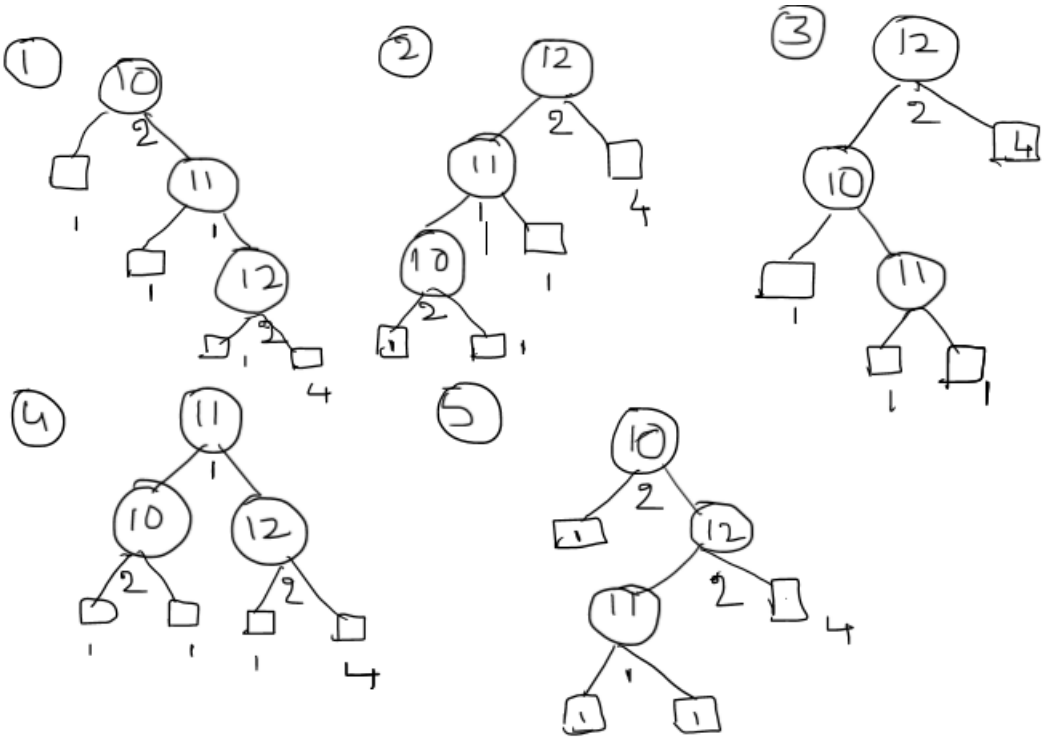
Where,

P(i) – Probability of searching an internal node (successful search)

q(i) – Probability of searching an external node (unsuccessful search)

Ex:

Keys are 10, 11, 12 and their probabilities for successful search are 2, 1, 2 and unsuccessful search are 1,1,1, 4. From the keys the following BSTs are obtained.



Cost of T(1) = 1*2+2*1+3*2 + 1*1+2*1+3*1+3*4 = 28

Cost of T(2) = 1*2+2*1+3*2 + 1*4+2*1+3*1+3*1 = 22

Cost of T(3) = 1*2+2*2+3*1 + 1*4+2*1+3*1+3*1 = 21

Cost of T(4) = 1*1+2*2+2*2 + 1*1+2*1+2*1+2*4 = 22

Cost of T(5) = 1*2+2*2+3*1 + 1*1+2*4+3*1+3*1 = 24

The minimum cost tree is T3. Hence the OBST is T3.

Ex:

N = 6 with K1 to K4 (do, if, int, while)

P1=3, P2=3, P3=1, P4=1

Q0=2, Q1=3, Q2=1, Q3=1, Q4=1

Solution: For finding OBST, calculate the values of W, C and R.

W(i, j) = p(j) + q(j) + W(i, j-1)

C(i,j) = min {(C(I,k-1)+C(k,j)} + W(i,j)

i<k<=j

R(i,j) = k (where cost is minimum)

i→	**0**	**1**	**2**	**3**	**4**
j-i = 0	W(0,0)= 2 C(0,0)=0 R(0,0) =0	W(1,1)= 3 C(1,1)=0 R(1,1) =0	W(2,2)= 1 C(2,2)=0 R(2,2) =0	W(3,3)= 1 C(3,3)=0 R(3,3) =0	W(4,4)= 1 C(4,4)=0 R(4,4) =0
j-i=1	W(0,1)=8 C(0,1)= 8 R(0,1)= 1	W(1,2)= 17 C(1,2)=7 R(1,2) =2	W(2,3)= 3 C(2,3)= 3 R(2,3) =3	W(3,4)=3 C(3,4)= 3 R(3,4) =4	
j-i=2	W(0,2)=12 C(0,2)= 19 R(0,2)= 1	W(1,3)= 9 C(1,3)= 12 R(1,3) =2	W(2,4)= 5 C(2,4)= 8 R(2,4) =3		
j-i=3	W(0,3)=14 C(0,3)= 25 R(0,3)=2	W(1,4)= 11 C(1,4)= 19 R(1,4) = 2			
j-i=4	W(0,4)=16 C(0,4)= 32 R(0,4)=2				

Initially,

W(0,0) = Q0	C(0,0) = 0	R(0,0) = 0
W(1,1) = Q1	C(1,1) = 0	R(1,1) = 0
W(2,2) = Q2	C(2,2) = 0	R(2,2) = 0
W(3,3) = Q3	C(3,3) = 0	R(3,3) = 0
W(4,4) = Q4	C(4,4) = 0	R(4,4) = 0

W(0,1) = w(0,0) + Q1 + P1 = 2+3+3 =8

W(1,2) = W(1,1) + Q2+ P2 = 3+3+1= 7

W(2,3) = W(2,2) + Q3+ P3 = 1+1+1 = 3

W(3,4) = W(3,3) + Q4+ P4 = 1+1+1 = 3

W(0,2) = W(0,1) + Q2 + P2 = 8+3+1 = 12

W(1,3) = W(1,2) + Q3+ P3 = 7+1+1 = 9

W(2,4) = W(2,3) + Q4+ P4 = 3+1+1 = 5

W(0,3) = W(0,2) + Q3 + P3 = 12+1+1 = 14

W(1,4) = W(1,3) + Q4+ P4 = 9+1+1 = 11

W(0,4) = W(0,3) + Q4 + P4 = 14+1+1 = 16

find out the cost matrix as follows;

C(I,J) = Min $_{i<k<=j}$ {c(i,k-1)+c(K,j)}+W(i, j)

when k = 1, j=1

C(i, j) = Min $_{K=1}$ c(i,k-1)+c(K,j)}+W(i, j)

C(0,1) = (C(0,0) + C(1,1)) + W(0,1) = 8

C(1,2) = (C(1,1) + C(2,2)) + W(1,2) = 7

C(2,3)) = (C(2,2) + C(3,3)) + W(2,3) = 3

C(3,4) = (C(3,3) + C(4,4)) + W(3,4) = 3

C(i, j) = Min $_{K=2}$ {c(i,k-1)+c(K,j)}+W(i, j)

C (0, 2) = min{C(0, 0)+C(1,2), C(0,1) + C(2,2)} + W(0, 2)

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

= 7+12

= 19 (**K=1)**

C(1, 3) = min{C(1,1) +C(2,3), C(1,2)+C(3,3)} + W(1,3)

= min(0+3, 7+0) + 9

= 3 + 9

= 12 **(K= 2)**

C(2, 4) = min{C(2,2) +C(3,4), C(2,3)+C(4,4)} + W(2,4)

= min(0+3, 3+0) + 5

= 3 + 5

= 8 **(K= 3)**

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)

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

= 11 + 14

= 25 **(K= 2)**

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)

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

= 8 + 11

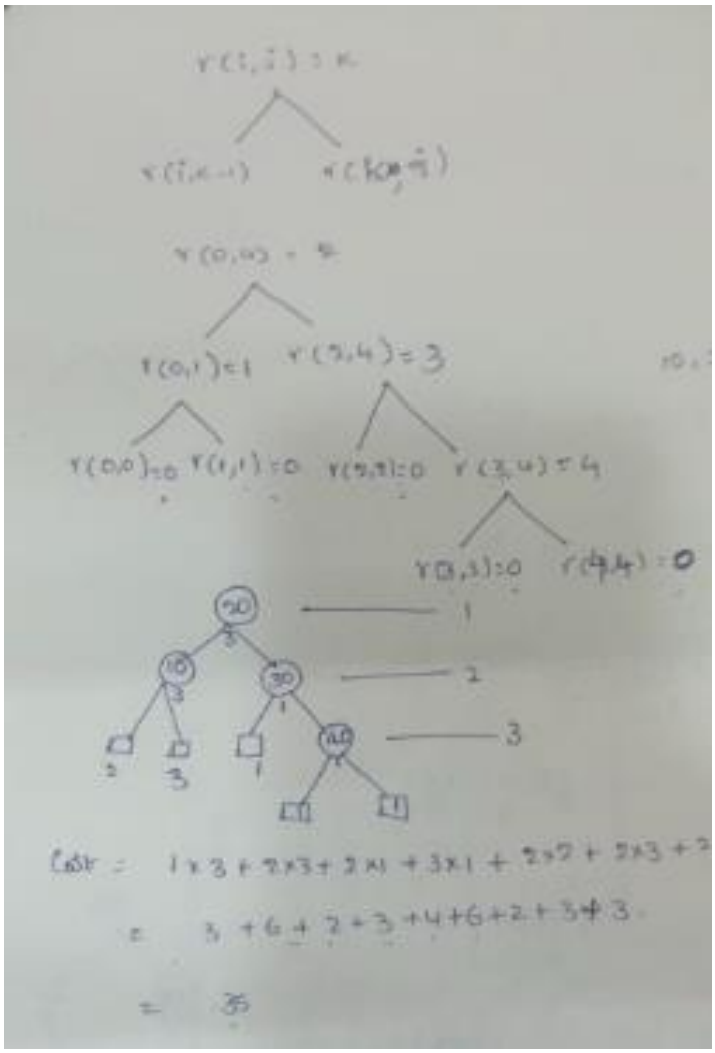
= 19 **(K= 2)**

C(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+16, 8+8,19+3, 25+0} + 16

= 16 + 16

= 32 **(K= 2)**



Exercise:

1. Use function OBST to compute w(i,j),r(i,j) and c(i,j), 0<=I<j<=4, for the identifier set (a1, a2, a3, a4)=(count, float, if, while) with p(1)=1/20, p(2)=1/5 p(3)=1/10, p(4)=1/20, q(0)=1/5, q(1)=1/10, q(2)=1/5, q(3)=1/20, and q(4)=1/20. Using the r(i,j)'s, construct the Optimal Binary Search Tree
2. Construct an optimal binary search tree for the following data: n=4, (a1,a2,a3,a4)=(do, if, int, while), p(1:4)= (3,3,1,1) and q(0:4)= (2,3,1,1,1).
3. Consider 4 elements $a_1 < a_2 < a_3 < a_4$ with $q_0=0.25$, $q_1=3/16$, $q_2=q_3=q_4=1/16$. $p_1=1/4$, $p_2=1/8$ $p_3=p_4=1/16$.

(a) Construct the optimal binary search tree as a minimal cost tree.

(b) Construct the table of values W_{ij}, C_{ij}, V_{ij} computed by the algorithm to compute the roots of optimal subtrees.

Divide and Conquer (DAC) Vs. Greedy Method:

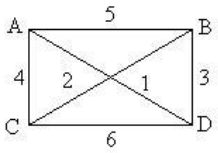
	Divide and Conquer (DAC)	Greedy Method (GM)
1	It is used to obtain a solution for a given problem. It does not aim for optimum solution.	The main aim of GM is to obtain optimum solution.
2	The problem is divided into subproblems and solved these subproblems are independently. Finally all the solutions are combined to get the final result.	In this set of feasible solutions are generated and finally one solution is selected as optimum solution.
3	Less efficient when compared to Greedy Method.	Comparatively efficient when compared to DandC.
4	Space requirement is high when compared to Greedy Method.	Space requirement is less when compared to DandC.
5	Some of the applications are Merge Sort, Quick Sort, Binary Search, etc.	Some of the applications are fractional knapsack, minimum cost spanning tree, optimal merge patterns, single source shortest path, etc.

Greedy Method Vs. Dynamic Programming:

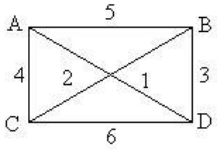
	Greedy Method (GM)	Dynamic Programming
1	It does not guarantees optimal solution in all cases.	It is guaranteed that this technical generate optimal solution as it generally considers all possible cases and then chooses the best.
2	Memory requirement is less when compared to the Dynamic Programming.	Memory requirement is high when compared to the Greedy Method.
3	It is not applicable for multistage problems.	It is applicable for multistage problems.
4	Simple and faster to implement.	Complex and difficult to implement.
5	Some of the applications are fractional knapsack, minimum cost spanning tree, optimal merge patterns, single source shortest path, etc.	Some of the applications are 0/1 knapsack, all pairs shortest path, optimal binary search tree, string editing etc.

Important Questions

1. Explain about Greedy method and its applications.
2. Use the greedy algorithm for sequencing unit time jobs with deadlines and profitsto generate the solution when n=7, (p1, p2,…p7)=(3, 5, 20, 18, 1, 6, 30), and (d1, d2,…, d7)=(1, 3, 4, 3, 2, 1, 2)
3. Solve the following instance of knapsack problem using greedy method. [n=7(objects), m=15, profits are (P1,P2,P3,P4,P5,P6,P7)=(10,5,15,7,6,18,3) and its corresponding weights are (W1,W2,W3,W4,W5,W6,W7)=(2,3,5,7,1,4,1).
4. What is a Minimum Cost Spanning tree? Explain Kruskal's Minimum cost spanning tree algorithm with a suitable example.
5. Write and explain Prim's algorithm for finding minimum cost spanning tree of a graph with an example.
6. Discuss the Dijsktra's single source shortest path with an example.
7. Explain the general concept of dynamic programming and Construct an optimal binary search tree for the following data: n=4, (a1,a2,a3,a4)=(do, if, int, while), p(1:4)= (3,3,1,1) and q(0:4)= (2,3,1,1,1).
8. Find the solution for the0/1 Knapsack problem. When n=3, (W_1, W_2, W_3)=(18, 15, 10,) . (P1, P2, P3)= (25, 24, 15) and m=20.
9. (a) Write an algorithm for all pairs shortest path. Find the shortest paths between all pairs of nodes in the following graph



   (b) What are the advantages of finding shortest paths and explain the application areas.
10. Find the shortest tour off TSP for the following graph using dynamic programming



11. Define dynamic programming. Give the solution string editing with an example.
12. Differentiate between Greedy Method and Dynamic Programming.
13. Differentiate between Greedy Method and Divide and Conquer.

Short Answer Questions

1. When Greedy method is used?
2. Name any four applications of Greedy method.
3. What is feasible solution in the Greedy method.
4. What is constraint in the job sequencing with deadlines?
5. What is the objective of job sequencing with deadlines?
6. How does Greedy method solve the fractional knapsack problem?
7. What is time complexity of fractional knapsack problem using Greedy method.
8. What is the objective of minimum cost spanning trees?
9. Name any two Greedy algorithms used to find minimum cost spanning trees. What is the key difference between these two algorithms?
10. What is the goal of single source shortest path problem.
11. What is dynamic programming algorithm design and what is its key principle.
12. Which algorithm is used to finds all pairs shortest path and also write it's time complexity.
13. What is the objective of all pairs shortest paths problem?
14. What is time complexity of the dynamic programming approach to the 0/1 knapsack problem?
15. What is the main constraint of the 0/1 knapsack problem?
16. What is the main constraint of string editing (edit distance) problem?
17. What is the objective of constructing an optimal binary search tree?
18. What is the objective of travelling salesman problem?
19. What is time complexity of Sting editing, OBST and TSP?