11/2/22

MODULE - 3

Greedy method and dynamic Programming.

Greedy method:- It is a problem solving Technique. This method is used to determine all possible Solutions of the problem is called feasible Solutions. Among all feasible Solutions we select one solution as the best Solution called the optimal Sol.

> Feasible Solution is the one which satisfies the Constraints of the problem.

> optimal solution is one which satisfies the objectives of the problem.

Control Abstraction of greedy method:-

Algorithm Greedy (a, n)

An array a[1:n] Contains the 'n' inputs

```
Solution = ∅;        // initialize the Solution
for i = 1 to n do
{
  x = select(a);     // Selects an input from array
  if feasible (Solution, x) then
  Solution = Union (Solution, x);
}
return Solution;
}
```

Applications of greedy method:-

1. Job Sequencing with dead lines
2. knapsack problem
3. Minimum spanning Tree
4. Single Source shortest path

1. Job Sequencing with dead lines:-

n - jobs
P_i - Profit
d_i - deadline.

=> Time taken to execute a job is called deadline.

=> if all Jobs are executed within the deadline then we will get Profit.

Jobs	J_1	J_2	J_3	J_4	J_5
profit	20	15	10	5	1
deadline	2	2	1	3	3

> Let $n = 5$ $(P_1, P_2, P_3, P_4, P_5)$

$(20, 15, 10, 5, 1)$

$(d_1, d_2, d_3, d_4, d_5) = (2, 2, 1, 3, 3)$

Assume that there is a machine in which each job is executed in 1 unit of time.

$J_1 \rightarrow 0-1, 1-2$

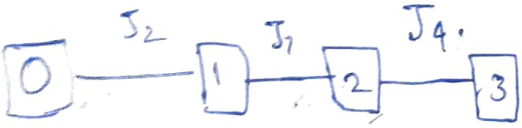
$J_2 \rightarrow 0-1, 1-2$

$J_3 \rightarrow 0-1$

$J_4 \rightarrow 0-1, 1-2, 2-3$

$J_5 \rightarrow 0-1, 1-2, 2-3.$

Jobs	slots-assigned	Solution	Profit
-	-	$\emptyset$	0
J_1	[1, 2)	J_1	20
J_2	[0, 1] (1, 2)	J_1, J_2	20+15
J_3	not allowed		
J_4	[0, 1] [1, 2] [2, 3)	J_1, J_2, J_3	20+15+5 = 40
J_5	not allowed		



Max profit = 40.

Suppose J_1 uses the slot 1-2 then J_2 cannot use the same slot so J_2 will use slot 0-1. Similarly, here J_3 and J_5 are ignored or cancelled since the slots are already filled.

Profit = $J_2 + J_1 + J_4$

= 20 + 15 + 5

= 40//.

2)

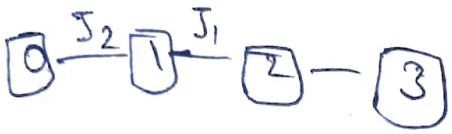
2> There are 4 jobs whose deadlines and profits are given as $(d_1, d_2, d_3, d_4) = (2, 1, 3, 2)$ and $(P_1, P_2, P_3, P_4) = (70, 12, 18, 35)$. find sequence of jobs that can be executed on a machine when its deadline is with maximum profit.

$J_1 \rightarrow 0-1, 1-2$

$J_2 \rightarrow 0-1$

$J_3 \rightarrow 0-1, 1-2$

$J_4 \rightarrow 0-1$.



$P_1 + P_2 = 70 + 12$

max Profit = 82

=> Knapsack problem.

n - no. of objects

Pi - Profit

wi : weight of the object

m - knapsack Capacity

Objective of knapsack problem is to place the objects in a knapsack to get a maximum profit. Here total weights of the objects are less than the capacity of the knapsack.

The objects are included or not included or a fractional part of the objects in a knapsack depends on x values.

Here x values are either 0 to 1 i.e $0 \le x \le 1$.

$$x = \begin{cases} 1 & - \text{ complete object is placed in knapsack} \\ 0 & - \text{ object not placed in u} \\ \text{fraction} & - \text{ fractional part of the object is placed in knapsack.} \end{cases}$$

Example:-

objects	1	2	3	4	5	6	7
Profit (p)	10	5	15	7	6	18	3
weights (w)	2	3	5	7	1	4	1

n = 7, m = 15

P/W	5	1.3	3	1	6	4.5	3
x	1	2/3	1	0	1	1	1
	x_1	x_2	x_3	x_4	x_5	x_6	x_7

$0 \le x \le 1$

knapsack capacity m = 15
15-1=14
14-2=12
12-4=8
8-5=3
3-1=2
2-2=0

Constraints:- $\sum w_i x_i \le m$

max profit $\sum p_i x_i$

$\sum w_i x_i = w_1x_1 + w_2x_2 + \cdots + w_7x_2$

$= 2\times1 + 3\times2/3 + 5\times1 + 7\times0 + 1\times1 + 4\times1 + 1\times1$

$= 2+2+5+0+1+4+1$

$\sum w_i x_i = 15.$

$\therefore \sum w_i x_i \leq m.$

$\sum p_i x_i = 10\times1 + 5\times2/3 + 15\times1 + 7\times0 + 6\times1 + 18\times1 + 3\times1$

$= 10 + 10/3 + 15 + 0 + 6 + 18 + 3$

$= 55.3$

$\therefore \sum p_i x_i = 55.3$

Example:-

$n = 7, \quad m = 15.$

$(p_1, p_2 \cdots p_7) = (10, 5, 15, 7, 6, 18, 3)$

$(w_1, w_2 \cdots w_7) = (2, 3, 5, 7, 1, 4, 1)$

done already.

Example:-

$n = 7 \qquad m = 15$

$(p_1, p_2 \cdots p_7) = (5, 10, 15, 7, 8, 9, 6)$

$(w_1, w_2 \cdots w_7) = (1, 3, 5, 4, 1, 2, 2).$

find the optimal solution of the Problem.

Sol:- $p/w = 5 \quad 10/3 \quad 3 \quad 7/4 \quad 8 \quad 9/2 \quad 3$

$p/w = 5 \quad 3.3 \quad 3 \quad 1.7 \quad 8 \quad 4.5 \quad 3.$

$x = x_1 \; x_2 \; x_3 \; x_4 \; x_5 \; x_6 \; x_7$

$= 1 \; 1 \; 1 \; 1/4 \; 1 \; 0 \; 1$

$15-7=14$
$14-1=13$
$13-2=11$
$11-3=8$
$8-5=3$
$3-2=1$
$1-1=0$

$m = 15.$

$\sum w_i x_i = 1\times1 + 3\times1 + 5\times1 + 1/4\times1 + 1\times1 + 2\times1 + 2\times1$

$= 1+3+5+1+.1+.2+2$

$= 15.$

$\therefore \sum w_i x_i \leq m$

$\sum p_i x_i = 5x1 + 10x1 + 15x1 + 7x1/4 + 8x1 + 9x1 + 6x1.$

$= 5 + 10 + 15 + 8 + 9 + 6 + 7/4$

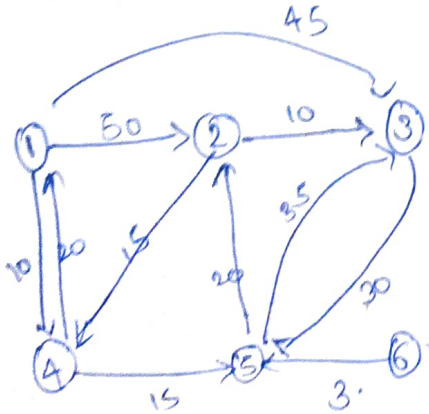
$= 53 + 1.7$

$\therefore \sum p_i x_i = 54.7.$

4)

Single Source shortest path Problem (or) Dijkstra algorithm:

Let $G(V, E)$ be a graph. Then from starting vertex to all remaining vertices are determined. The starting vertex is called source and the last vertex is called destination.



The no's on the edges are the weights.

If node 1 is the source vertex. Then the shortest path from node 1 to 2 is

1 -> 2 - 50

1 -> 4 -> 5 -> 2 — 45 ✓

1 -> 3 -> 5 -> 2 — 95.

1 -> 4 -> 5 -> 2 i.e the length is

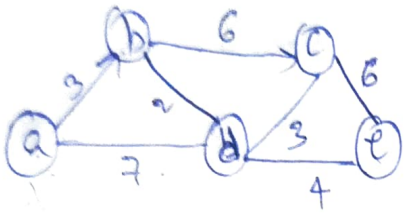
10 + 15 + 20 = 45.

The following tables gives the shortest paths from nodes 1 to nodes 4, 5, 2, 3 respectively.

S.No	path	shortest path	length
1	1, 4	1 -> 4	10
2	1, 5	1 -> 4 -> 5	25.
3	1, 2	1 -> 4 -> 5 -> 2	45
4	1, 3	1 -> 3	45
5	1, 6	no path.	

for the graph nearest vertex to starting vertex is 4 and its length = 10.

⇒ Identify the shortest path from source 'a' to all other vertices in the graph shown in the below figure. Using greedy method.



Source → a.

S.No.	path	shortest path	length.
1.	a→b	a→b	3.
2.	a, c	a→b→d→c	8.
3.	a, d	a→b→d	5.
4.	a, e	a→b→d→e	9.

Shortest path a→b of length 3.

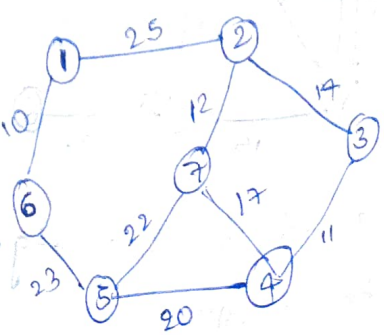
⇒ Minimum Cost Spanning Trees:-

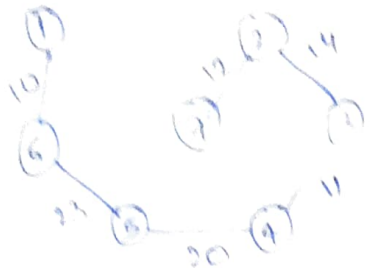
A minimum Cost Spanning Tree of a graph is a Spanning Tree that has least total Cost. There are two different algorithms for minimum Cost Spanning Trees. Both algorithms are similar in that they build one edge at a time. These algorithm's are

1) Prim's algorithm and
2) Kruskhchal's algorithm.

① Prim's algorithm:- This algorithm finds a minimum Cost Spanning Tree of an edge weighted Connected and Undirected graph G = (V, E). The algorithm Constructs the minimum Cost Spanning tree of a graph by selecting edges from the graph one-by-one and adding those edges to the spanning tree.

Example: Consider the following graph





Now consider all the vertices of the graph first select an edge with minimum weight. This algorithm proceeds by selecting adjacent edges with minimum weight. Care should be taken to avoid a loop.

Step-1

Total weight = 0.

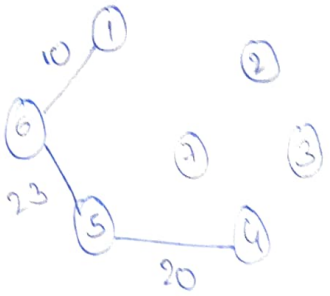
Step-2

Total weight = 10.

Step-3

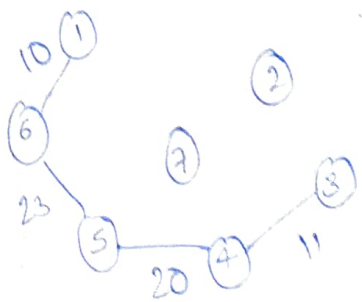
Total weight = 33.

Step-4



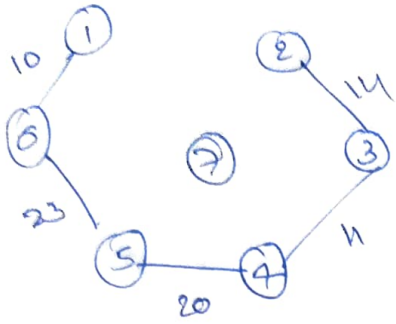
Total weight = 53.

Step-5



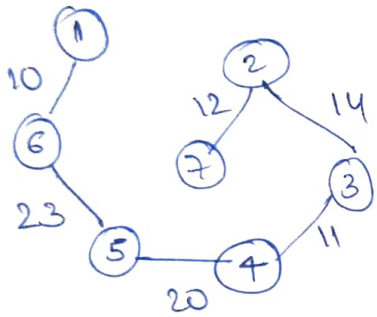
Total weight = 53+11
= 64.

Step-6



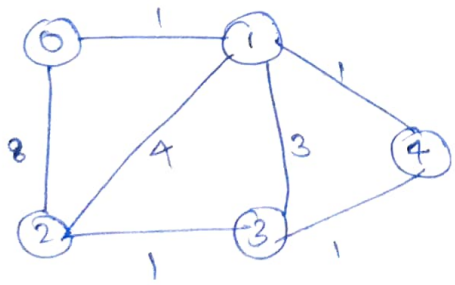
64+14 = 78.

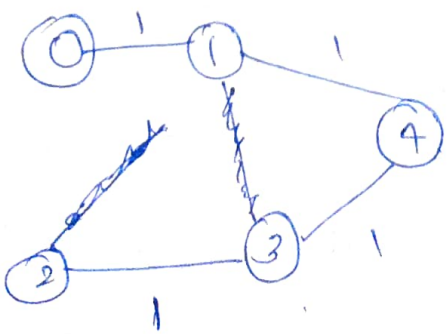
Step-7

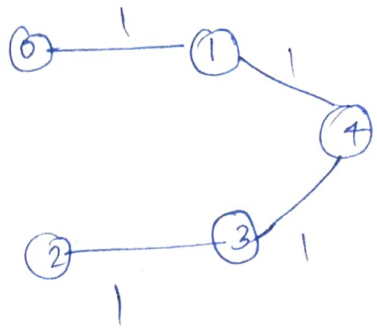


78+12 = 90.

Example







1+1+1+1 = 4.

② ~~Krushkal's~~ Algorithm:-

Algorithm for Prim's Algorithm:-

(n, E, w, t)

// 'E' is an array that stores all edges of the graph.

// 'w' is the weight of each edge.

// 'n' is the no. of edges b/w vertices

// 't' is the array in which min spanning tree is computed.

// t[i,1], t[i,2] is an edge in minimum spanning tree.

{ m ← w(u,v) // (u,v) is ~~weight~~ edge of minimum cost.

t(1,1) ← u , t(1,2) ← v.

```
for i ← 1 to n do
if [w(i,u) > w(i,v)] then
    adj[i] ← v else
    adj[j] ← u
    adj[u) ← adj(v) ← 0
for i ← 2 to (n-1) do
{
// find remaining (n-2) edges for t
// let j be the index such that
    adj[j] ≠ 0 and w[j, adj[j]) is min.
t(i,1) ← j
t(i,2) ← adj[j]
minCost ← minCost + w[j, adj(j)]
adj(j) ← 0
for k ← 1 to n do
```

a) Kruskal's Algorithm:- Kruskals alg is used for finding the min. Cost of a spanning tree of a graph.

Input:- A Connected graph 'G' with non-negative values assigned to each edge.

Output:-

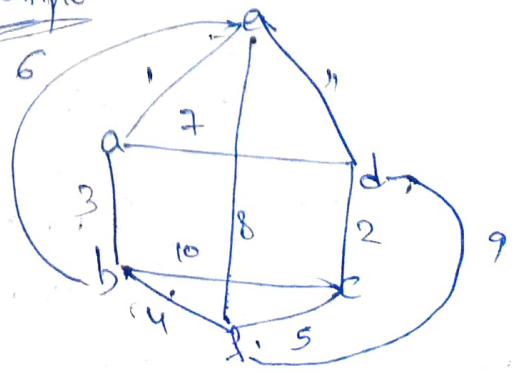
Algorithm:-

(i) Select any edge of minimal value ie not a loop. This is first edge of 'T'

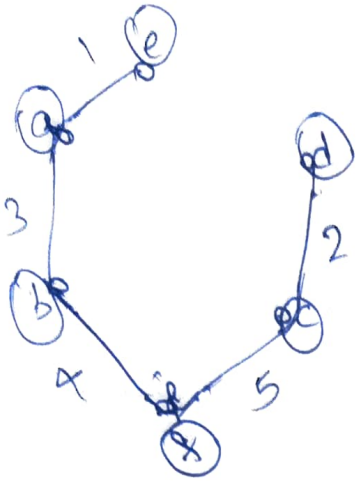
(ii) Select remaining edge of 'G' having next minimal value.

(iii) Continue Step (2) Until 'T' Contains (n-1) edges where 'n' is the no. of Vertices of 'G'.

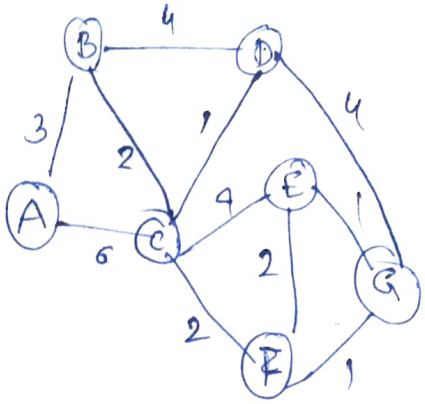
Example:-



Example:- (using Prim's).

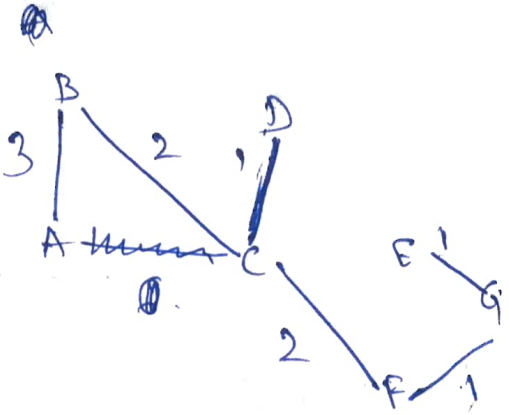


Total weight = 15.



Sol:-

Using prim's.

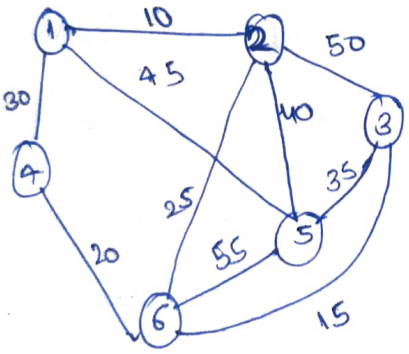


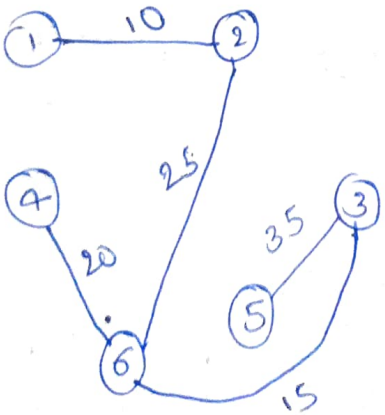
3+2+1+2+1+1+1.

= 10.

∴ Total weight = 10.

Example:-

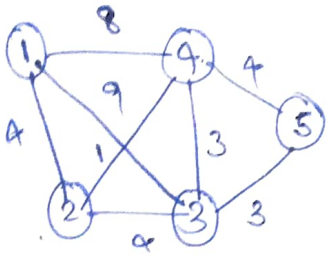




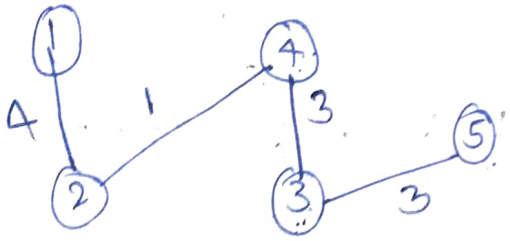
Total weight = 10 + 20 + 25 + 15 + 35.

= 105.

Example:-



Sol:-



Total weight = 4+1+3+3

= 11.

8-2=6
6-1=5
5-2=3
3-3=0

$\sum w_i x_i$ =

$5 \times 3/5 + 0 \times 4 + 1 \times 2 + 1 \times 2 + 1 \times 1$

= 3+0+2+2+1

= 8

$\sum p_i x_i = 10 \times 3/5 + 0 + 8 \times 1 + 5 \times 1 + 3 \times 1$

= 6+8+5+3

= 22.

Knapsack problem:-

Find the optimal solution for the given instance of knapsack problem

$n = 5, \quad (w_1, w_2, w_3, w_4, w_5) = (5, 4, 2, 2, 1)$

$m = 8 \quad (p_1, p_2 \ldots p_5) = (10, 5, 8, 5, 3)$

Problem Constraint

$\sum w_i x_i \leq m$

$8 \leq 8$

objective of the Problem

$\max \sum p_i x_i$

max = 22.

$p/w = 10/5 \;\; 5/4 \;\; 8/2 \;\; 5/2 \;\; 3/1$

= 2 5/4 4 5/2 3

3.5 0 1 1 0

$x_1 \; x_2 \; x_3 \; x_4 \; x_5$

=> Consider the following instance of a knapsack problem

$n = 3, m = 20$

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

$(w_1, w_2, w_3) = (18, 15, 10)$

Sol:

$P/w = 25/18 \quad 24/15 \quad 15/10$

$= 1.3 \quad 1.6 \quad 1.5.$

20 - 15 = 5
5 - 5 = 0

$x_1 = 0 \quad x_2 = 1 \quad x_3 = 5/10$

$x_4 = 0 \quad x_2 = 1 \quad = 1/2$

$\sum P_i x_i = 25 \times 0 + 24 \times 1 + 15 \times 1/2$

$= 24 + 7.5$

$= 31.5$

$\sum w_i x_i = 18 \times 0 + 15 \times 1 + 10 \times 1/2$

$= 15 + 5$

$= 20.$

=> Find the optimal solution of the following using job sequencing using deadlines.

$n = 4.$

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

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

$P_1 \rightarrow 0 \rightarrow 1, 1 \rightarrow 2$

$P_2 \rightarrow 0 \rightarrow 1$

$P_3 \rightarrow 0 \rightarrow 1, 1 \rightarrow 2$

$P_4 \rightarrow 0 \rightarrow 1.$

0 — P_2 — 1 — P_1 — 2

Profit $= P_1 + P_2$

$= 100 + 10.$

$= 110$

0 — P_1 — 1 — P_3 — 2

Profit $= P_1 + P_3$

$= 100 + 15$

$= 115$

∴ max profit = 115.

Algorithm for knapsack problem:-

```
Algorithm Greedy knapsack (m, n)
// P[1:n] and the w[1:n] Contain the
   Profit.
// & weight respectively of the n
   objects ordered.
// Such that P[i]/w[i] >= P[i+1]/w[i+1]
// m is the knapsack size and
   x[1:n] is the solution vector
{
  For i:=1 to n do x[i]:=0.0;
  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);
}.
```

Output:-

```
Enter the items: 3
Enter the Profits of
Products: 10
          15
          4.

Enter the weights of the
objects: 2
         5
         7.

Enter the maximum Capacity
of the knapsack : 10.
```

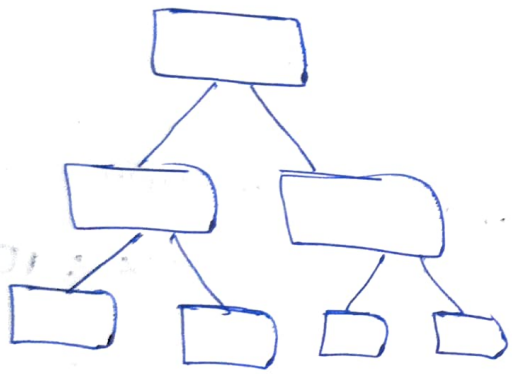
Dynamic Programming:-

Dynamic Programming is a Search method Suitable for optimisation Problems. This technique is invented by us mathematician Richard Bellmann in 1950's. Dynamic programming to the world of Programming stands for Planning and it does not mean by Computer programming. This

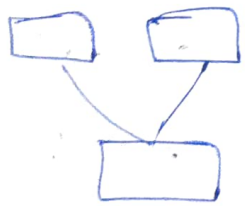
method has a Bottom-up approach that means:

1) It starts with Smallest Sub-Problems.
2) Combining their Solutions
3) Until they arrive the Solution of the original problem.

Top down



Bottom-up:-



All the Solutions are Combined to arrive at the Solution of the main Problem.

Differences b/w greedy method and dynamic Programming:-

Greedy method	Dynamic Programming
> In greedy method from a Set of feasible Solutions we will pick up the optimal Solution	> In Dynamic Pro. there is no specific Set of feasible Solutions.
> In greedy method decision is made at a time.	> In this method more than one decision are made at a time
> Top-down approach	> Bottom-up approach
> less expensive method	> more expensive than greedy method.
> It is used to obtaining optimal Solution.	> This is also used for Solving the optimal Solution.

Applications of Dynamic Programming:

1) Multi stage graphs
2) All pairs shortest Path Problem
3) Optimal Binary Search tree (OBST)
4) 0/1 knapsack Problem
5) Travelling Sales person Problem.

0/1 knapsack problem:-

In this problem there are 'n' no. of objects, each having a Profit 'pi', weight 'wi' and Capacity 'm'.

The objective of 0/1 knapsack Problem is to place the objects in a knapsack to get a maximum Profit. Here total weights of the objects are less than the Capacity of the knapsack.

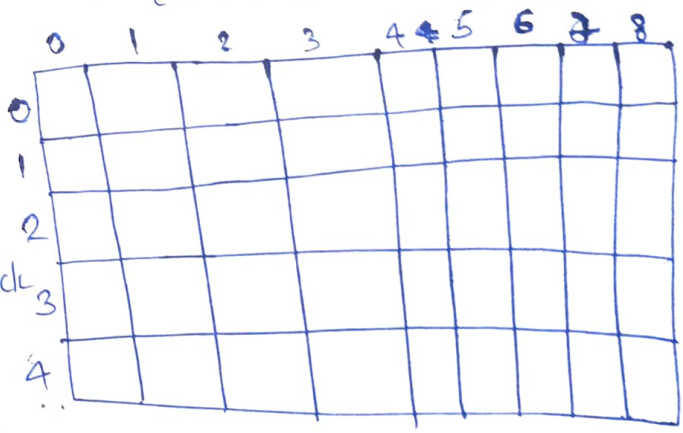
The objects are included or not included in a knapsack. Using 'x' values. Here x=1, Complete objects are included and x=0. object is not included in a knapsack.

This problem can be solved using dynamic Programming by using sequence of decisions and Pick up the optimal Solution.

0/1 knapsack can be solved by using tabulation method as shown below:

for example Consider n=4 and m=8

$P_i = \{1,2,5,6\}$ and $w = \{2,3,4,5\}$.



The objects can be filled in knapsack using the formula

$$v[i,w] = \max\{v[i-1], v[i-1, w-w[i]] + P[i]\}.$$

where i = row number, P = profit, w = weights of objects, w = Column number, V = table of rows & Columns

P_i	W_i		0	1	2	3	4	5	6	7	8
		0	0	0	0	0	0	0	0	0	0
1	2	1	0	0	1	1	1	1	1	1	1
2	3	2	0	0	1	2	2	3	3	3	3
5	4	3	0	0	1	2	5	5	6	7	7
6	5	4	0	0	1	2	5	6	6	7	8

1st object

P_i	W_i
1	2

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

2nd object

P_i	W_i
1	2
2	3

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

3rd object

P_i	W_i
1	2
2	3
5	4

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

4th object

P_i	W_i
1	2
2	3
5	4
6	5

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

↓ max profit

x_1	x_2	x_3	x_4
0	1	0	1

$2 + 6 = 8$

$8 - 6 = 2$ in 3rd and 2nd row

$2 - 2 = 0$ in 1st row.

⇒ Solve the Solution for 0/1 knapsack problem using dynamic Programming $(P_1, P_2, P_3, P_4) = (11, 21, 31, 33)$, (w_1, w_2, w_3, w_4)

6, 8, 4, 5 = (2, 11, 22, 15)

2, 7, 5, 8

$m = 10$, $n = 4$.

P_i	w_i	w_i	0	1	2	3	4	5	6	7	8	9	10
"		0	0	0	0	0	0	0	0	0	0	0	0
6	2	1	0										
8	7	2	0										
4	5	3	0										
5	8	4	0										

1st object

P_i	w_i
6	2

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

2nd object

P_i	w_i
6	2
8	7

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

max

3rd object

P_i	w_i
6	2
8	7
4	5

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

4th object

P_i	w_i
6	2
8	7
4	5
5	8

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

=> Solve 0/1 knapsack problem using dynamic programming

$n = 3$, $m = 6$, $(w_1, w_2, w_3) = (2, 3, 4)$

$(P_1, P_2, P_3) = (1, 2, 5)$

P_i	w_i		0	1	2	3	4	5	6
		0	0	0	0	0	0	0	0
1	2	1	0	0	1	1	1	1	1
2	3	2	0	0	1	2	2	3	3
5	4	3	0	0	1	2	5	5	6

1st object

P_i	w_i
1	2

0	1	2	3	4	5	6
		1				

2nd object

P_i	w_i
1	2
2	3

0	1	2	3	4	5	6
			2		3	

Third

P_i	w_i
1	2
2	3
5	4

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

x_1	x_2	x_3
1	0	1

$1 + 5 = 6$

=> All pairs shortest Path Algorithm:-

All pairs shortest path problem is used to find the shortest path between every pair of vertices of the graph.

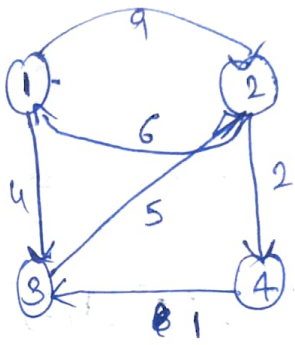
This problem can be used to solve using dynamic. In dynamic programming problems should be solved by taking a sequence of decisions in each stage.

In this method first select vertex 1 as the middle vertex where to find out the shortest path between each pair of vertices of the graph.

Similarly the vertices 2, 3 & 4 are middle vertices then find the shortest path b/w each pair of vertices also.

Now consider the following example:

Find all pairs shortest path Problem using floyd warshall algorithm.



Vertex 1 is the middle vertex. find shortest path from

1-2, 1-3, 1-4.

The adjacency matrix of the above graph is.

A^0,

	1	2	3	4
1	0	9	4	∞
2	6	0	∞	2
3	∞	5	0	∞
4	∞	∞	1	0

If there is no direct path ∞ will be indicated in the adjacency matrix.

=> Here 'i' is the middle element.

$$A^1 = \begin{bmatrix} 0 & 9 & 4 & \infty \\ 6 & 0 & & \\ \infty & & 0 & \\ \infty & & & 0 \end{bmatrix}$$

$A^0[2,3]$ Via Vertex 1 is.

$A^0[2,1] + A^0[1,3]$

$= 6 + 4$

$= 10.$

$A^0[2,3] = \infty.$

$\therefore \infty > 10$ ✓

$A^0[2,3] = 10$ ✓

$A^0[2,4] = A^0[2,1] + A^0[1,4]$

$2 = 6 + \infty.$

✓ $2 < \infty$

$A^0[3,2] = A^0[3,1] + A^0[1,2]$

$5 = \infty + 9.$

✓ $5 < \infty$

$A^0[3,4] = A^0[3,1] + A^0$

$= \infty + \infty$

$\infty = \infty$ ✓

$A^0[4,2] = A^0[4,1] + A^0[1$

$\infty = \infty + 9.$

$\infty = \infty$ ✓

$A^0[4,3] = A^0[4,1] + A^0[1$

$1 = \infty + 4.$

✓ $1 = \infty$

$$A^1 = \begin{bmatrix} 0 & 9 & 4 & \infty \\ 6 & 0 & 10 & 2 \\ \infty & 5 & 0 & \infty \\ \infty & \infty & 1 & 0 \end{bmatrix}$$

$$A^2 = \begin{bmatrix} 0 & 9 & & \\ 6 & 0 & 10 & 2 \\ & 5 & 0 & \\ & \infty & & 0 \end{bmatrix}$$

Via Vertex 2.

$A^0[1,3] = A^0[1,2] + A^0[2,3]$

$4 = 9 + 10$

$4 < 19$ ✓

$A^2 =$

0	9	4	11
6	0	10	2
11	5	0	7
∞	∞	1	0

$A^0[1,4] = A^0[1,2] + A^0[2,4]$

$\infty = 9 + 2$

$\infty \geq 11$ ✓

$A^0[3,1] = A^0[3,2] + A^0[2,1]$

$\infty = 5 + 6$

$\infty > 11$ ✓

$A^0[3,4] = A^0[3,2] + A^0[2,4]$

$\infty = 5 + 2$

$\infty \geq 7$ ✓

$A^0[4,1] = A^0[4,2] + A^0[2,1]$

$\infty = A^0[\infty + 6$

$\infty = \infty$ ✓

$A^0[4,3] = A^0[4,2] + A^0[2,3]$

$1 = \infty + 10$

$A^3 =$ Via Vertex 3

0		4	
	0	10	
11	5	0	7
		1	0

$A^3[1,2] = A^3[1,3] + A^3[3,2]$

$9 = 4 + 5$

$9 = 9$ ✓

$A^3[1,4] = A^3[1,3] + A^3[3,4]$

$11 = 4 + 7$

$11 = 11$ ✓

$A^3[2,1] = A^3[2,3] + A^3[3,1]$

$6 = 10 + 11$

$6 < 21$ ✓

$A^3[2,4] = A^3[2,3] + A^2[3,4]$

$2 = 10 + 7$

$2 < 17$ ✓

$A^3[4,1] = A^3[4,3] + A^3[3,1]$

$\infty = 1 + 11$

$\infty \geq 12$ ✓

$A^3[4,2] = A^3[4,3] + A^3[3,2]$

$\infty = 1 + 5$

$\infty \geq 6$ ✓

$$A^3 = \begin{bmatrix} 0 & 9 & 4 & 11 \\ 6 & 0 & 10 & 2 \\ 4 & 5 & 0 & 7 \\ 12 & 6 & 1 & 0 \end{bmatrix}$$

Via Vertex 4.

$$A^4 = \begin{bmatrix} 0 & & & 11 \\ & 0 & & 2 \\ & & 0 & 7 \\ 12 & 6 & 1 & 0 \end{bmatrix}$$

$A^4[1,2] = A^4[1,4] + A^4[4,2]$

$9 = 11 + 6$

$9 \leq 17$ ✓

$A^4[1,3] = A^4[1,4] + A^4[4,3]$

$4 = 11 + 3$

$4 \leq 14$ ✓

$A^4[2,1] = A^4[2,4] + A^4[4,1]$

$6 = 2 + 12$

$6 \leq 14$ ✓

$A^4[2,3] = A^4[2,4] + A^4[4,3]$

$10 = 2 + 1$

$10 \geq 3$ ✓

$A^4[3,1] = A^4[3,4] + A^4[4,1]$

$11 = 7 + 12$

$11 \leq 19$ ✓

$A^4[3,2] = A^4[3,4] + A^4[4,2]$

$5 = 7 + 6$

$5 \leq 13$ ✓

$$A^4 = \begin{bmatrix} 0 & 9 & 4 & 11 \\ 6 & 0 & 3 & 2 \\ 11 & 5 & 0 & 7 \\ 12 & 6 & 1 & 0 \end{bmatrix}$$

Problem:

The adjacency matrix of the following graph is:

A⁰	1	2	3
1	0	4	3
2	6	0	2
3	1	∞	0

$$A^0 = \begin{matrix} \\ 1 \\ 2 \\ 3 \end{matrix} \begin{matrix} \begin{matrix} 1 & 2 & 3 \end{matrix} \\ \begin{bmatrix} 0 & 4 & 3 \\ 6 & 0 & 2 \\ 1 & \infty & 0 \end{bmatrix} \end{matrix}$$

Via Vertex 1.

$$A^1 = \begin{bmatrix} 0 & 4 & 3 \\ 6 & 0 & \\ 1 & & 0 \end{bmatrix}$$

$A^1[2,3] = A^1[2,1] + A^1[1,3]$

$2 \neq 6 + 3$

$\checkmark 2 \leq 9$

$A^1[3,2] = A^1[3,1] + A^1[1,2]$

$\infty = 1 + 4$

$\infty \leq 5$ ✓

$$A^1 = \begin{bmatrix} 0 & 4 & 3 \\ 6 & 0 & 2 \\ 1 & 5 & 0 \end{bmatrix}$$

$$A^2 = \begin{bmatrix} 0 & 4 & \\ 6 & 0 & 2 \\ & 5 & 0 \end{bmatrix}$$

Via Vertex 2.

$A^1[1,1] = A^1[1,2] + A^1[2,1]$

$0 = 4 + 6$

$\checkmark 0 \leq 10$

$A^1[1,3] = A^1[1,2] + A^1[2,3]$

$3 = 4 + 2$

$\checkmark 3 \leq 5$

$A^1[3,1] = A^1[3,2] + A^1[2,1]$

$1 = 5 + 6$

$\checkmark 1 \leq 11$

$A^2 =$ $\begin{bmatrix} 0 & 4 & 3 \\ 6 & 0 & 2 \\ 1 & 5 & 0 \end{bmatrix}$.

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

Via Vertex 3.

$A^3[1,2] = A^3[1,3] + A^3[3,2]$

$4 = 3 + 5$

$4 \neq 8$ ✓

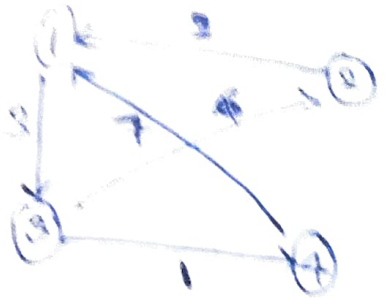
$A^3[2,1] = A^3[2,3] + A^3[3,1]$

$6 = 2 + 4$

$\times 6 = 8$ ✓

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

∴ Vertex 3 is the shortest path.



Adjacency matrix for the above graph.

$A^0 =$

	1	2	3	4
1	0	∞	3	∞
2	2	0	∞	∞
3	∞	6	0	1
4	7	∞	∞	0

Optimal Binary Search Tree (OBST)

For example consider n = 4,

key = $\{a_1, a_2, a_3, a_4\}$

$p(1:4) = (3, 3, 1, 1)$ and

$q(0:4) = (2, 3, 1, 1, 1)$ where $a_1, \ldots a_4$ are identifiers, $p(1:4)$ is the Successful probabilities and $q(0:4)$ is the unsuccesful probabilities stasts with 0's. Now Construct optimal Binary Search Tree.

$p(1:4) = (3, 3, 1, 1)$ and

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

initially,

$w[i,i] = q[i]$

$c[i,i] = 0$

$r[i,i] = 0$.

when $i = 0$.

$\Rightarrow w[0,0] = q[0] = 2$

$c[0,0] = 0$

$r[0,0] = 0$.

when $i = 1$,

$w[1,1] = q[1] = 3$.

$c[1,1] = 0$

$r[1,1] = 0$

Similarly:

$w[2,2] = 1$, $w[3,3] = 1$,

$w[4,4] = 1$.

$|j - i| = 1$.

$w[i,j] = w[i, j-1] + p[j] + q[j]$

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

$\rightarrow i$

$\downarrow j-i$	0	1	2	3	4
0	$w[0,0]=2$ $c[0,0]=0$ $r[0,0]=0$	$w[1,1]=3$ $c[1,1]=0$ $r[1,1]=0$	$w[2,2]=1$ $c[2,2]=0$ $r[2,2]=0$	$w[3,3]=1$ $c[3,3]=0$ $r[3,3]=0$	$w[4,4]=1$ $c[4,4]=0$ $r[4,4]=0$
1	$w[0,1]=8$ $c[0,1]=8$ $r[0,1]=1$	$w[1,2]=7$ $c[1,2]=7$ $r[1,2]=2$	$w[2,3]=3$ $c[2,3]=3$ $r[2,3]=3$	$w[3,4]=3$ $c[3,4]=3$ $r[3,4]=4$	
2	$w[0,2]=$ $c[0,2]=$ $r[0,2]=$	$w[1,3]=$ $c[1,3]=$ $r[1,3]=$	$w[2,4]=$ $c[2,4]=$ $r[2,4]=$		
3	$w[0,3]=$ $c[0,3]=$ $r[0,3]=$	$w[1,4]=$ $c[1,4]=$ $r[1,4]=$			
4	$w[0,4]=$ $c[0,4]=$ $r[0,4]=$				

$r[i,j] = k$.

From these 3 formulae we can find out $|j-i| = 1, 2, 3, 4$.

$$w[i,j] = w[i,j-1] + p[j] + q[j]$$

$$w[0,1] = w[0,1-1] + p[1] + q[1]$$

$$= w[0,0] + p[1] + q[1]$$

$$= 2 + 3 + 3$$

$$= 8.$$

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

$$c[0,1] = \min_{0<k\le 1} \{c[0,1-1] + c[1,1]\} + w[0,1].$$

$k=1$

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

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

$$= 0 + 8 = 8.$$

$r[i,j] = k$.

$r[0,1] = 1$.

$$w[0,2] = w[0,2-1] + p[2].$$

$$= w[0,1] + p[2].$$

$$= 3 + 3 + 1$$

$$= 7$$

$$c[0,2] = \min_{0<k\le 2} \{c[0,2-1] \ldots$$

$k=2$ +

$$= 7 \ldots$$

$r[1,2] = 2$.

Similarly, solve the ...

$w[2,3]$, $c[2,3]$, $r[2,3]$ ...

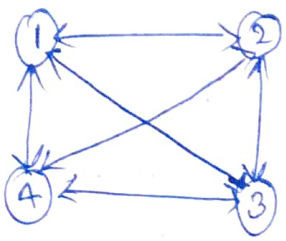
$w[3,4]$, $c[3,4]$, $r[3,4]$ ...

Travelling Sales person problem:-

let the graph $G=(V,E)$ consisting of a set of edges and a set of Vertices.

Travelling sales person problem means start with starting Vertex and visit all the remaining Vertices exactly once then come to the starting Vertex with minimum cost.

Suppose the graph has 4 Vertices then start with Vertex 1 and visit all the vertices: 2,3,4 exactly once and then come back to the Starting Vertex with minimum cost. This is called travelling Sales Person's Problem.

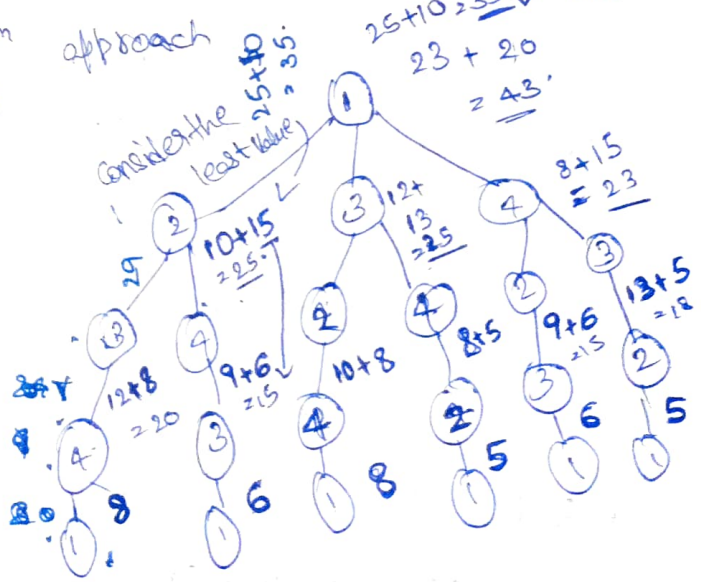


Consider the above direct edged graph

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

Assume the weights since the graph is Unweighted.

> Dynamic programming says that from all possible Solutions select the best Solution as the optimal solution This is a time consuming process. Here we use the process called Brute force approach



1 -> 2 -> 4 -> 3 = 10+25
= 35.

∴ It is the shortest path.