UNIT – III

DESIGN AND ANALYSIS OF ALGORITHMS

GREEDY METHOD

BASIC METHOD

- The Greedy method is most straight forward design technique
- It can be applied to a wide variety of problems, most of these problems have n-inputs and require us to obtain a subset that satisfies some constraints.
- Any subset that satisfies these constraints is called feasible solution.
- We need to find a feasible solution that either maximizes or minimizes a given objective function.
- A feasible solution that does this is called Optimal solution
- Greedy method works in stages, consider one input at a time.
- At each stage, a decision is made regarding whether a particular input is in an optimal solution.
- This is done by considering inputs in an order by some selection procedure.

BASIC METHOD

A greedy algorithm has five components:

- A set of candidates, from which to create solutions.
- A selection function, to select the best candidate to add to the solution.
- A feasible function is used to decide if a candidate can be used to build a solution.
- An objective function, fixing the value of a solution or an incomplete solution.
- An evaluation function, indicating when you find a complete solution.

BASIC METHOD

- Algorithm Greedy(a, n)

```
{
    solution = φ;
    for i := 1 to n do
        {
            x := Select(a);
            if Feasible(solution, x);
            solution := union(solution, x);
        }
    return solution;
}
```

KNAPSACK PROBLEM

- We are given n objects and a knapsack(bag). Object i has a weight w_i and capacity m.
- If a fraction $x_i, 0 \leq x_i \leq 1$, of object i is placed into the knapsack, then the profit $p_i x_i$ is earned.
- The objective is to obtain a filling of the knapsack that maximizes the total profit earned.
- We require the total weight of the chosen objects to be at most m.
- Hence, the objective of this algorithm is to

$$\text{Maximize } \sum_{1 \leq i \leq n} p_i x_i$$

$$\text{Subject to } \sum_{1 \leq i \leq n} w_i x_i <= \text{m}$$

$$\text{and } 0 \leq x_i \leq 1, 1 \leq i \leq n$$

The profits and weights are positive numbers.

KnapSack Problem

KnapSack Problem

Q: Consider the following instance of the knapSack problem:
$n = 3$, $m = 20$, $(P_1, P_2, P_3) = (25, 24, 15)$, $(W_1, W_2, W_3) = (18, 15, 10)$
Find the solution x_1, x_2, x_3 and Optimal profit

Ans: Four feasible solutions are

S.NO	(x_1, x_2, x_3)	$\sum w_i x_i$	$\sum p_i x_i$
1	$(\frac{1}{2}, \frac{1}{3}, \frac{1}{4})$	16.5	24.25
2	$(1, \frac{2}{15}, 0)$	20	28.2
3	$(0, \frac{2}{3}, 1)$	20	31
4	$(0, 1, 1/2)$	20	31.5

Solution-4 yields the maximum profit

KnapSack Problem

straight forward Greedy approach

- method selects highest profit item first. So item 1 is selected and we obtain profit as 24. and $x_1 = 1$. weight of the item $w_1 = 18$. Remaining Capacity is $20 - 18 = 2$.
- Next max profit item is item-2 with Profit 24. But the weight of that item is 15. Remaining Capacity is 2 only So we can place only 2 units into knapsack. $x_2 = \frac{2}{15}$. Now knapsack is full.
- we cannot select item 3.
- Maximum Profit we obtained is $= P_1x_1 + P_2x_2 + P_3x_3$
  $= 25 \times 1 + 24 \times \frac{2}{15} + 15 \times 0 = 25 + 3.2 = \underline{28.2}$
- Total weight included is $\sum w_i x_i = 20$

Knapsack Problem

* Arrange elements in profit/weight ratio decreasing order

$\frac{P_1}{w_1} = \frac{25}{18} = 1.3$ $\frac{P_2}{w_2} = \frac{24}{15} = 1.6$ $\frac{P_3}{w_3} = \frac{15}{10} = 1.5$

- Now arrange elements in profit decreasing order

Item	weight	Profit
2	15	24
3	10	15
1	18	25

- Now Greedy method selects first item first. Its weight is 15. So $x_2 = 1$. Remaining Capacity is $20 - 15 = 5$. Profit obtained is 24.

KnapSack Problem

- Next method selects 3rd item. Its weight is 10. But capacity of the knapsack is 5 only.

So 5 units can be extracted from 10 units item.

$x_3 = \frac{5}{10} = \frac{1}{2}$. Profit can be obtained is $15 \times \frac{1}{2}$.

So from 3rd item we get ~~Profit~~ as 7.5.

Total Profit obtained is = $24 + 7.5 = 31.5$

Now KnapSack is full. $x_3 = 1$

So $x_1 = 0$

- Total weight included $= \sum w_i x_i = w_1x_1 + w_2x_2 + w_3x_3 = 18 \times 0 + 15 \times 1 + 10 \times \frac{1}{2}$
  $= 0 + 15 + 5 = 20$ [full knapsack]

- Total Profit gained $= \sum P_i x_i = P_1x_1 + P_2x_2 + P_3x_3 = 25 \times 0 + 24 \times 1 + 15 \times \frac{1}{2}$
  $= 0 + 24 + 7.5$
  $= 31.5$

KNAPSACK PROBLEM

- Example:
- For the given set of items and knapsack capacity = 15 kg, find the optimal solution for the fractional knapsack problem making use of the greedy approach.

Item number	1	2	3	4	5	6	7
Profit(Pi)	10	15	7	9	6	20	30
Weight(Wi)	2	5	2	4	1	8	6

- **Solution:** **Straight Forward Greedy Approach:**
- Straight forward Greedy approach selects Highest profit item first. Item 7 has the highest profit with profit=30. weight is 6. total profit gaining after selecting item is 30. remaining knapsack capacity is 15-6=9. so x7=1.
- Next maximum profit item is item 6 with profit 20.Total profit gained after selecting item is 30+20=50. The remaining knapsack capacity is 9- 8=1. and solution is x6=1.
- Next maximum profit item is item 2 with profit 15. but the remaining capacity of knapsack is 1 unit. So extract 1 unit from item 2 (out of 5). So solution is x2=1/5. so the total profit gained is 50+15*1/5=53.
- The solution is x1=0, x2=1/5, x3=0, x4=0, x5=0, x6=1, x7=1.
- Total weight consumed in knapsack=15
- Total profit obtained from the straight forward greedy

- Solution: Find out profit per weight Pi/Wi

Item number	1	2	3	4	5	6	7
Profit(Pi)	10	15	7	9	6	20	30
Weight(Wi)	2	5	2	4	1	8	6
Pi / wi	5	3	3.5	2.25	6	2.5	5

Item number	5	7	1	3	2	6	4
Profit(Pi)	6	30	10	7	15	20	9
Weight(Wi)	1	6	2	2	5	8	4
Selection(Xi)	1	1	1	1	4/5=0.8	0	0

- Selection (Xi) Steps: let's have the capacity m=15
- The first profitable item we have are item no.5, so we select is 15-1=14. Now the remaining knapsack capacity is 14 and our selection is 1(means selected)
- Then we have the next profitable item is item no .7 so we select 14-6. Now the remaining knapsack capacity is 8 and our selection is 1(means selected)
- Then we have the next profitable item is item no .1 so we select 8-2. Now the remaining knapsack capacity is 6 and our selection is 1(means selected)
- Then we have the next profitable item is item no .3 so we select 6-2. Now the remaining knapsack capacity is 4 and our selection is 1(means selected)
- Then we have the next profitable item is item no .2. Its weight is 5 and our knapsack remaining capacity is 4, so now we are dealing with a greedy approach and select 4/5 items. (like take as we can)
- So our selection is 4/5.
- Now we don't have any remaining capacity so we can't take any more items, so it's selection is made 0 for other items.

- Formula:

Maximum profit = $\sum_{i=1}^{n} \binom{n}{i} \text{Pi Xi}$

Constraint subject to = $\sum_{i=1}^{n} \binom{n}{i} \text{Wi Xi} \leq \text{m}$

Where $0 \leq \text{Xi} \leq 1$

Xi = selection value of item i

⇨ 6*1 + 30*1 + 10*1+ 7*1 + 15*4/5 + 20*0 + 9*0

⇨ 6+30+10+7+12

⇨ 65 Ans.

- The knapsack would contain the following items- < 5,7,1,3,2 >.
- Knapsack's total profit would be 65 units.

```
Algorithm Greedy knapsack (m, n)
{
  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]
}
```

for the above algorithm, all the items are arranged in profit and weight ratio decreasing order.

Job Sequencing with Deadlines Problem

- Given a set of tasks with deadlines and total profit earned on completion of a task, find maximum profit earned by executing the tasks within the specified deadlines. Assume any task will take one unit of time to execute and any task can't execute beyond its deadline. Also, one task can be executed at a time.

Example: Given the jobs, their deadlines and associated profits as shown below:

Jobs	**J1**	**J2**	**J3**	**J4**	**J5**	**J6**
Deadlines	5	3	3	2	4	2
Profits	200	180	190	300	120	100

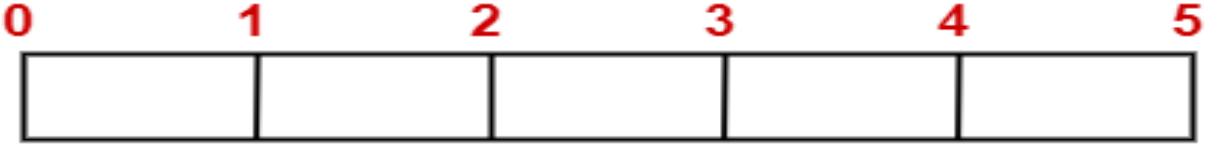
Step-01:

- Sort all the given jobs in decreasing order of their profit-

Jobs	J4	J1	J3	J2	J5	J6
Deadlines	2	5	3	3	4	2
Profits	300	200	190	180	120	100

Step-02:

- Value of maximum deadline = 5. So, draw a Gantt chart with maximu

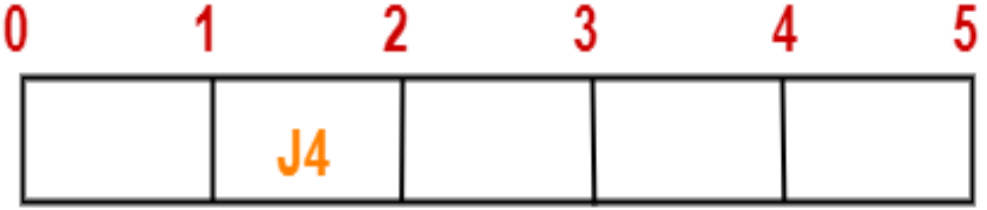


Gantt Chart

- Now, we take each job one by one in the order they appear in Step-01.We place the job on Gantt chart as far as possible from 0.

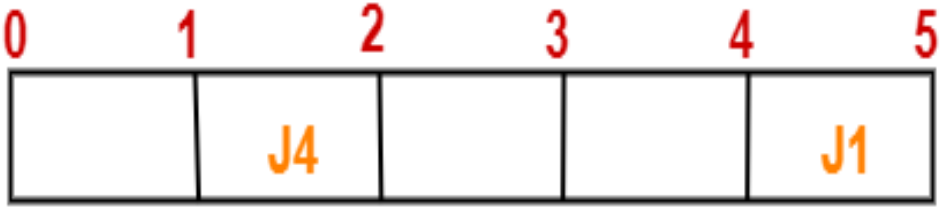
Step-03:

- We take job J4. Since its deadline is 2, so we place it in the first empty cell before deadline 2 as-



Step-04:

- We take job J1. Since its deadline is 5, so we place it in the first empty cell before deadline 5 as-



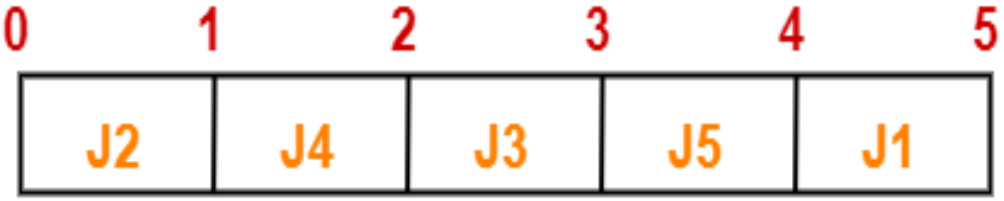
- **Step-05:**
- We take job J3. Since its deadline is 3, so we place it in the first empty cell before deadline 3 as-

Step-06:

- We take job J2. Since its deadline is 3, so we place it in the first empty cell before deadline 3. Since the second and third cells are already filled, so we place job J2 in the first cell as-

Step-07:

- Now, we take job J5.Since its deadline is 4, so we place it in the first empty cell before deadline 4 as-



- Now, The only job left is job J6 whose deadline is 2. All the slots before deadline 2 are already occupied. Thus, job J6 can not be completed.

- The optimal schedule is- **J2 , J4 , J3 , J5 , J1**
- This is the required order in which the jobs must be completed in order to obtain the maximum profit.
- All the jobs are not completed in optimal schedule. This is because job J6 could not be completed within its deadline.
- Maximum earned profit

  = Sum of profit of all the jobs in optimal schedule

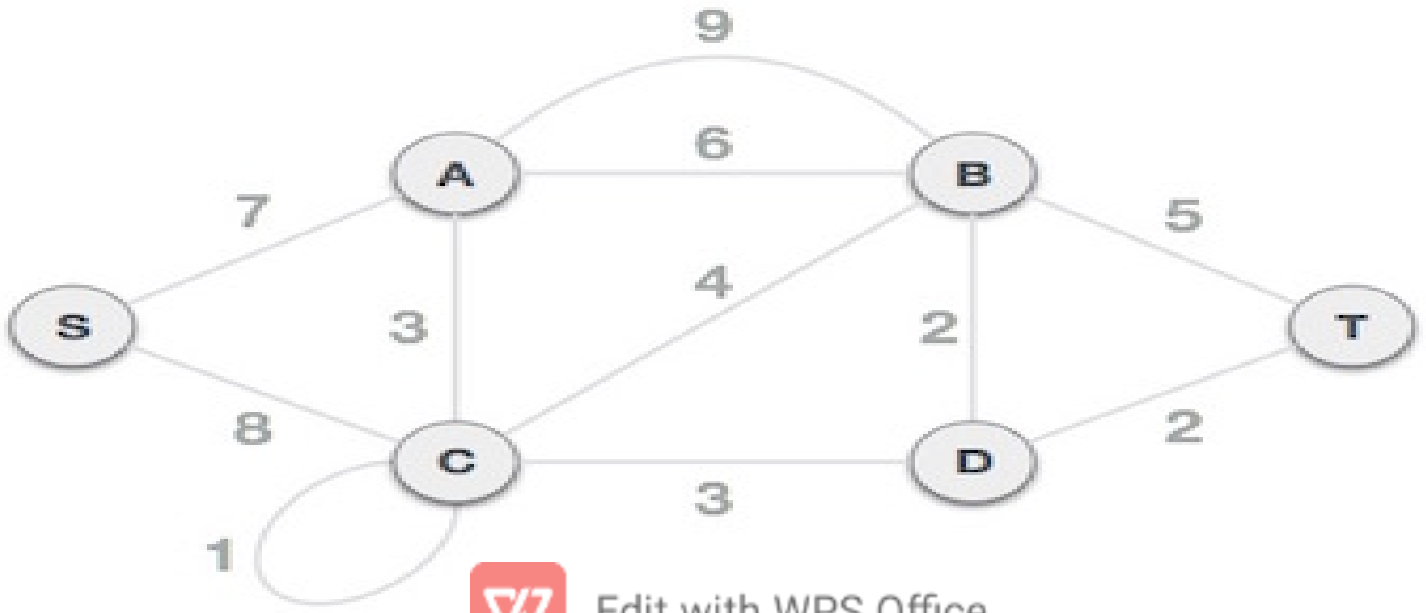
  = Profit of job J2 + Profit of job J4 + Profit of job J3 + Profit of job J5 + Profit of job J1

  = 180 + 300 + 190 + 120 + 200= 990 units

```
Algorithm JS(d, j, n)
    {
    d[0] := J[0] := 0;
    J[1] := 1;    k := 1;
for i := 2 to n do
    {
     r := k;
while ((d[J[r]) > d[i]) and (d[J[r] ≠ r) do
            r := r-1;
If((( d[J[r] ≤ d[i] and (d[i] > r)) then
      {
       for q := k to (r+1) step -1 do
          J[q+1] = J[q];
      J[r+1] = i;
       k := k+1;
       }
}
  return k;
}
```

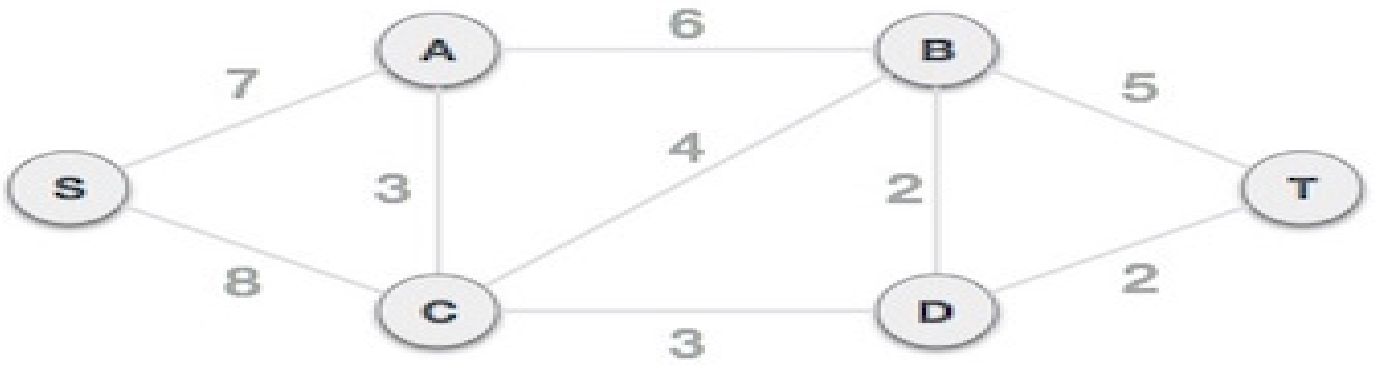
Minimum Cost Spanning Tree

- A **Minimum Spanning Tree (MST)** is a subset of edges of a connected weighted undirected graph that connects all the vertices together with the minimum possible total edge weight. To derive an MST, **Prim's** algorithm or **Kruskal's** algorithm can be used.
- Prims Algorithm: Prim's algorithm is a greedy approach to find the minimum spanning tree. In this algorithm, to form a MST we can start from an arbitrary vertex



Minimum Cost Spanning Tree

Step 1 - **Remove all loops and parallel edges :** Remove all loops and parallel edges from the given graph. In case of parallel edges, keep the one which has the least cost associated and remove all others.

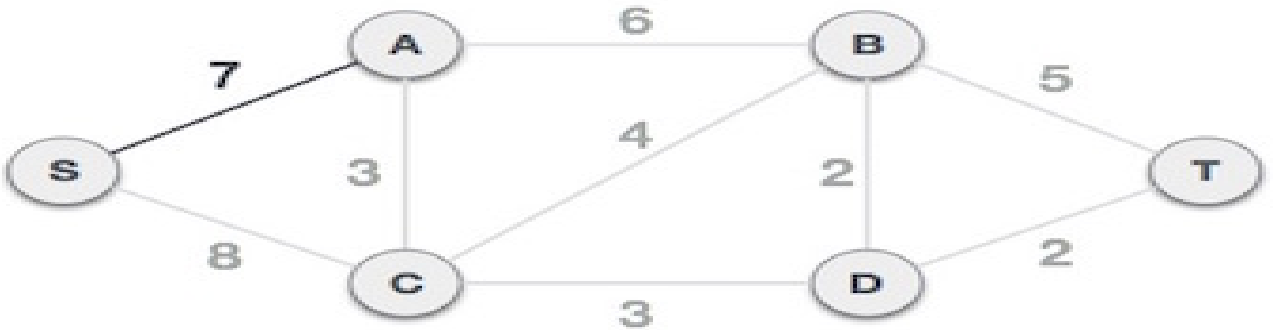


Step 2 - **Choose any arbitrary node as root node**

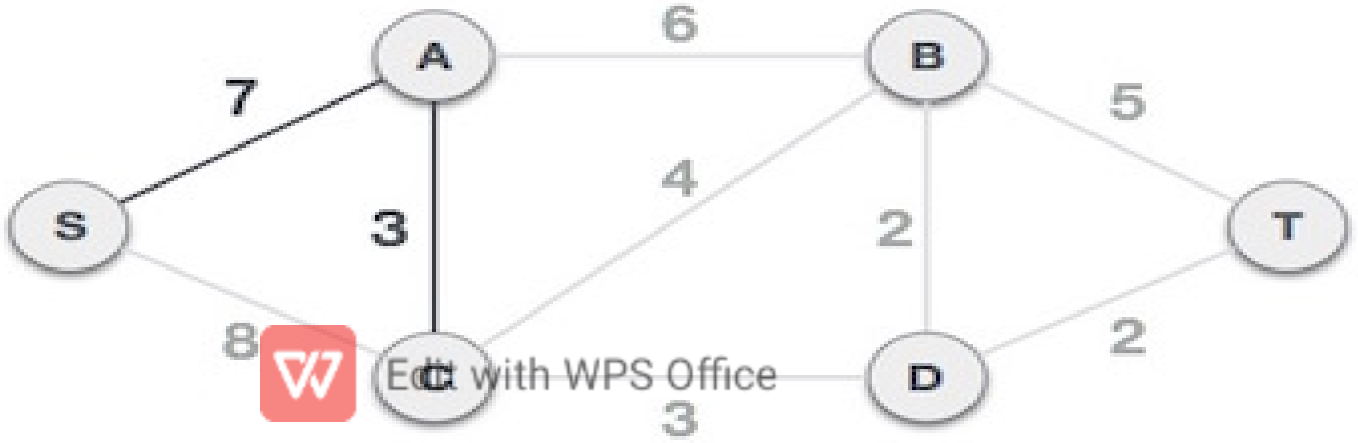
- In this case, we choose **S** node as the root node of Prim's spanning tree. This node is arbitrarily chosen, so any node can be the root node. One may wonder why any video can be a root node. So the answer is, in the spanning tree all the nodes of a graph are included and because it is connected then there must be at least one edge, which will join it to the rest of the tree.

Minimum Cost Spanning Tree

Step 3 - Check outgoing edges and select the one with less cost : After choosing the root node **S**, we see that S,A and S,C are two edges with weight 7 and 8, respectively. We choose the edge S,A as it is lesser than the other.

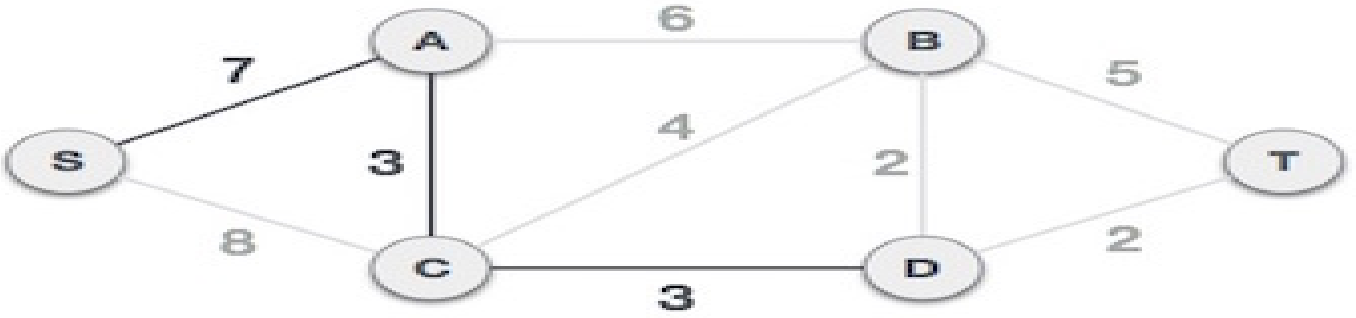


Now, the tree S-7-A is treated as one node and we check for all edges going out from it. We select the one which has the lowest cost and include it in the tree.



Minimum Cost Spanning Tree

- After this step, S-7-A-3-C tree is formed. Now we'll again treat it as a node and will check all the edges again. However, we will choose only the least cost edge. In this case, C-3-D is the new edge, which is less than other edge



- After adding node **D** to the spanning tree, we now have two edges going out of it having the same cost, i.e. D-2-T and D-2-B. Thus, we can add either one. But the next step will again yield edge 2 as the least cost. Hence, we are showing a spanning tree with both edges included.

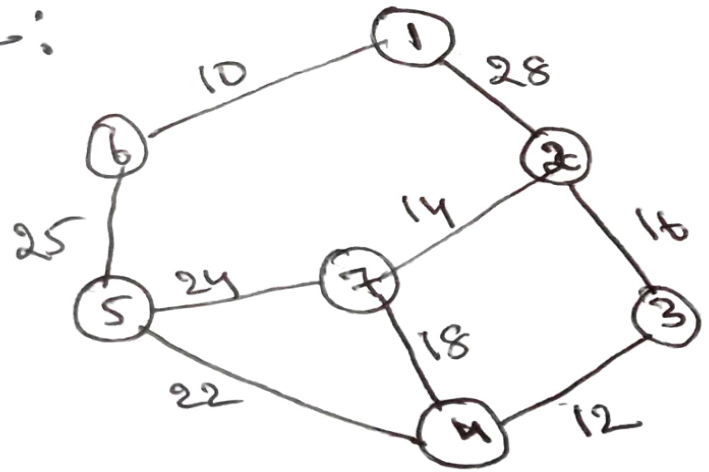
Minimum Cost Spanning Tree

Prim's Algorithm

A greedy method to obtain a minimum cost Spanning tree builds the tree edge by edge. The next edge to be included is choosen according to some optimization Criteria

- If A is the set of edges selected so far, then A forms a tree. The next edge (u, v) is to be included in A is a minimum cost edge not in A, with the property that $A \cup \{(u, v)\}$ is also a tree.

Ex:



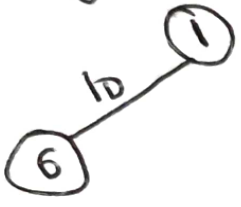
Minimum Cost Spanning Tree

- The prims algorithm will start with a tree that includes only a minimum cost edge of G.
- Then edges are added to this tree one by one.
- The next edge (i,j) to be added is such that i is a vertex already included in the tree, j is a vertex not yet included and cost of i,j, i.e cost(i,j) is min among all edges [k,l] such that k is in tree and l is not in the tree.

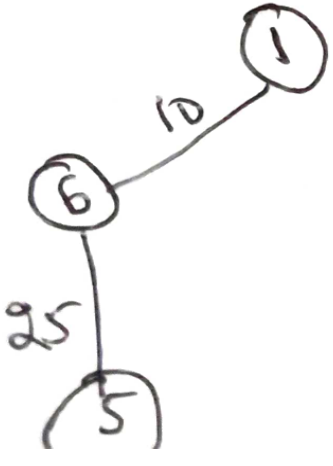
- Every time after selecting min-cost edge, check whether the inclusion of this edge making cycle in the Spanning tree of it makes cycle, then discard it and select next minimum cost edge. Otherwise add this edge into Spanning tree.

Minimum Cost Spanning Tree

- The min cost edge in the graph is (1,6) with cost 10. So include this edge into spanning tree. Now the tree is:



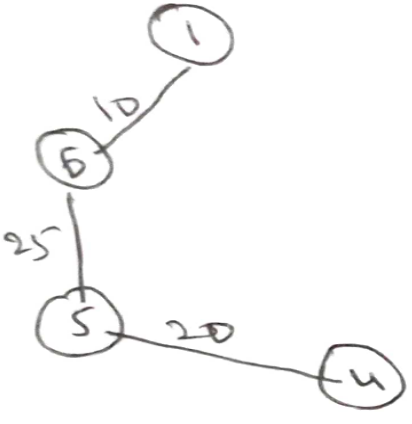
- Now consider nodes 1 and 6 and select nearest node from 1,6 in the graph. 1 has only one node 2 with cost 28 node 6 has nearest node 5 with cost 25.
- So (6,5) edge cost is less than (1,2). So include (6,5) edge into spanning tree.



Minimum Cost Spanning Tree

- now for the next edge, consider nodes 1, 6 and 5 and select nearest nodes from ~~these~~ three nodes and select min edge cost among all.

so edge (5, 4) is included as next edge

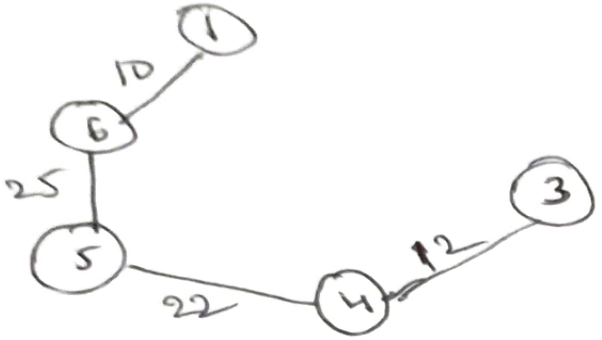


- now consider the nodes in the tree 1, 6, 5 and 4 and apply the same procedure.

So include (4, 3) as ~~next~~ min cost edge into Spanning tree

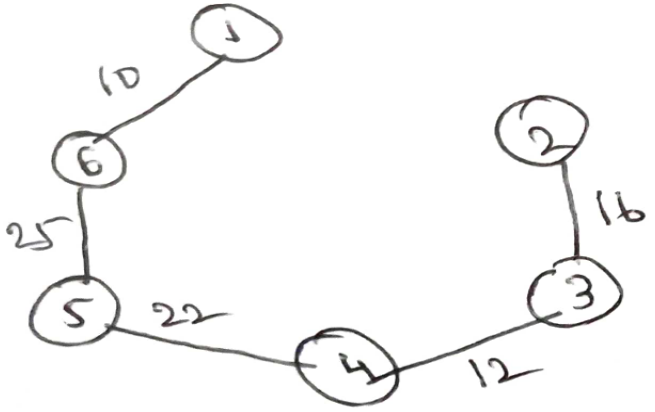
Minimum Cost Spanning Tree

The updated tree is:

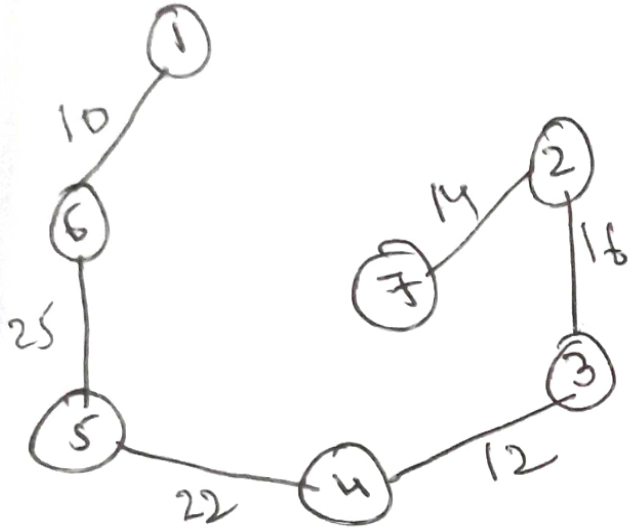


- Next consider included vertices in the tree and select nearest node from all the above vertices and select min cost among all.

min-cost edge is (3,2), add this edge into spanning tree.



- for next edge repeat the same procedure. so the edge (2,7) is included as next min cost edge.



All the vertices are included into the spanning tree.

So this tree is the minimum-cost spanning tree.

The cost of the spanning tree is: 99

Minimum Cost Spanning Tree

Prim's Algorithm

Let (k, l) be an edge of min cost in E;

minCost := cost[k, l].

t[1,1] := k; t[1,2] := l;

for i := 1 to n do // initialize near

if (cost[i, l] < cost[i, k] then near[i] := l;

else

near[i] := k;

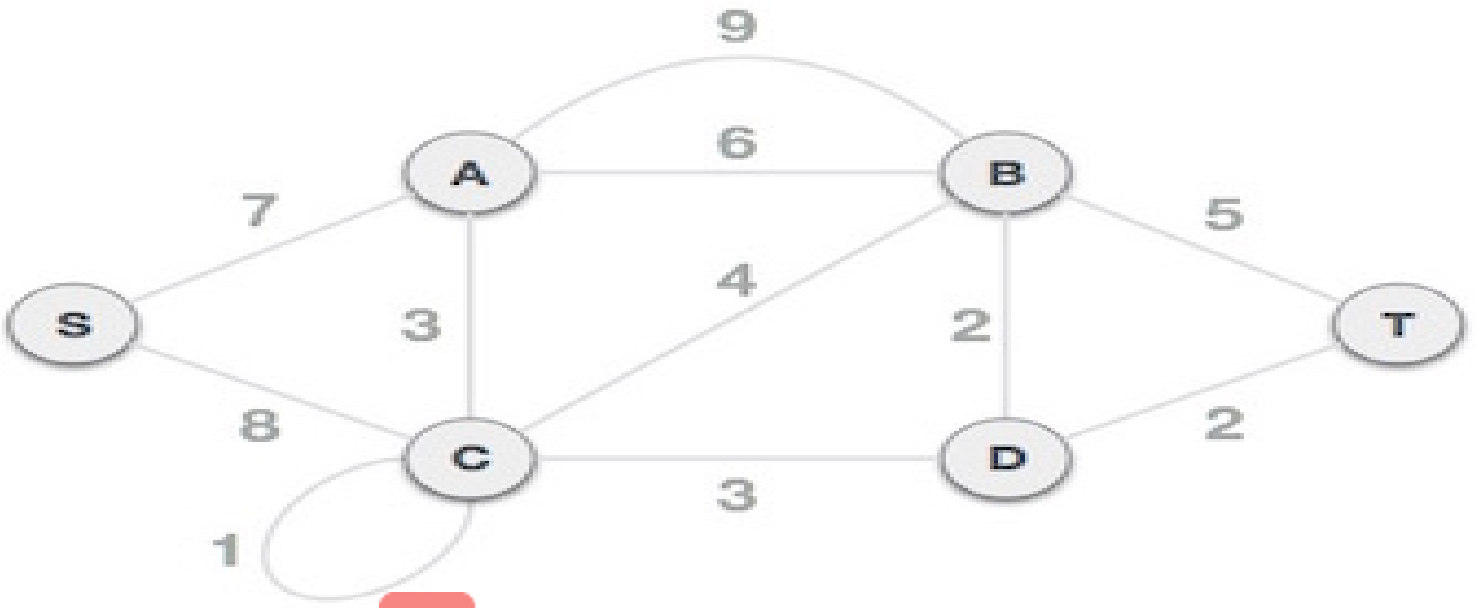
near[k] := near[l] := 0;

Minimum Cost Spanning Tree

```
for i := 2 to n-1 do
    {
Let j be the index such that near[j] ≠ 0 and
(cost[j], near[j]) is minimum;
t[i,1] = j;  t[i,2] := near[j];
mincost := mincost + cost[j, near[j]];
  near[j] := 0;
for k := 1 to n do  // update near
if (near[k] ≠ 0) and cost[k, near[k]] > cost[k,j]
then  near[k] := j;
    }
return mincost;
}
```

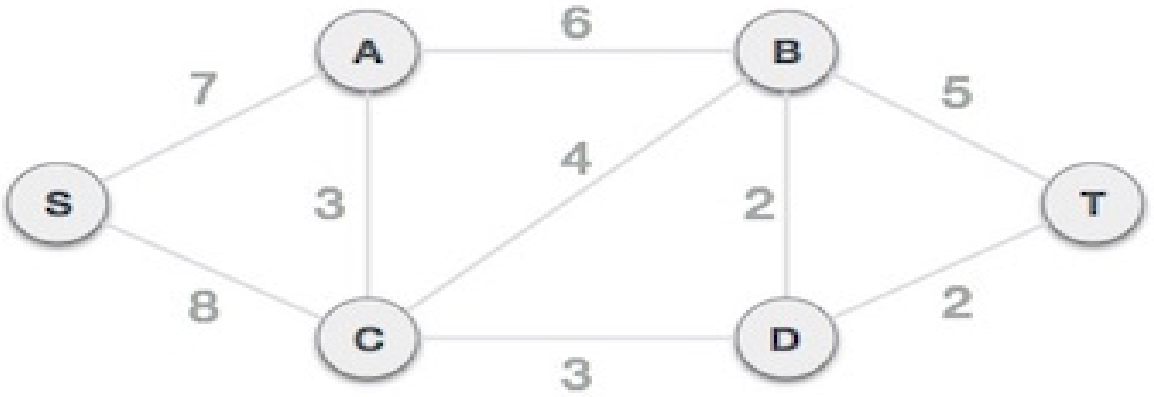
Minimum Cost Spanning Tree

- **Kruskal's Algorithm:** Kruskal's algorithm to find the minimum cost spanning tree uses the greedy approach. This algorithm treats the graph as a forest and every node it has as an individual tree. A tree connects to another only and only if, it has the least cost among all available options and does not violate MST properties.
- To understand Kruskal's algorithm let us consider the following example



Minimum Cost Spanning Tree

Step 1 - Remove all loops and Parallel Edges : Remove all loops and parallel edges from the given graph.

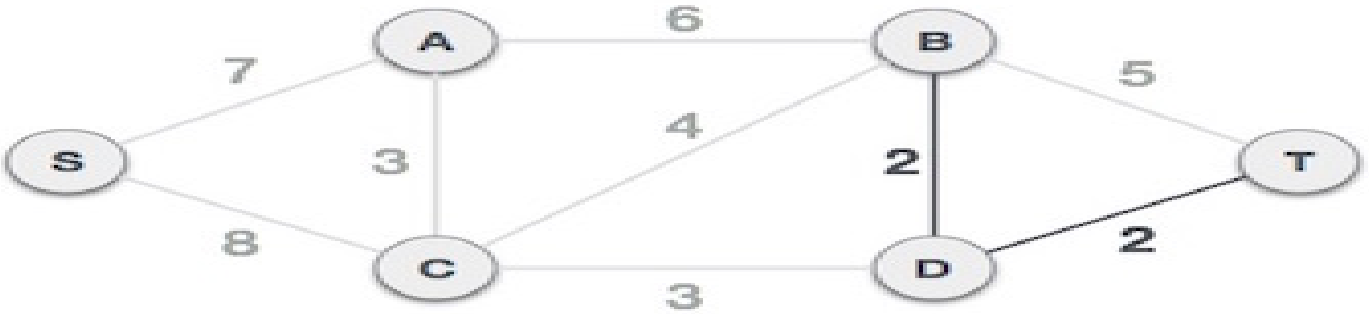


Step 2 - Arrange all edges in their increasing order of weight : The next step is to create a set of edges and weight, and arrange them in an ascending order of weightage (cost).

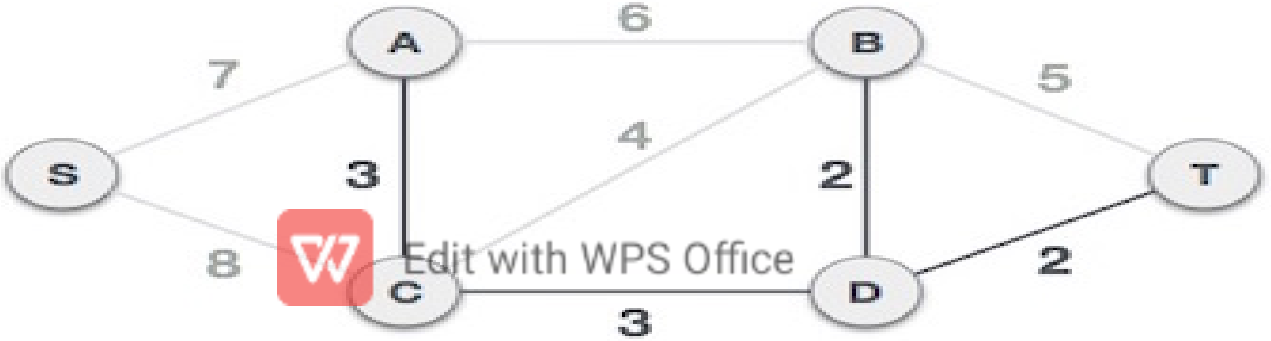
B, D	D, T	A, C	C, D	C, B	B, T	A, B	S, A	S, C
2	2	3	3	4	5	6	7	8

Minimum Cost Spanning Tree

Step 3 - Add the edge which has the least weightage : Now we start adding edges to the graph beginning from the one which has the least weight. Throughout, we shall keep checking that the spanning properties remain intact. In case, by adding one edge, the spanning tree property does not hold then we shall consider not to include the edge in the graph.

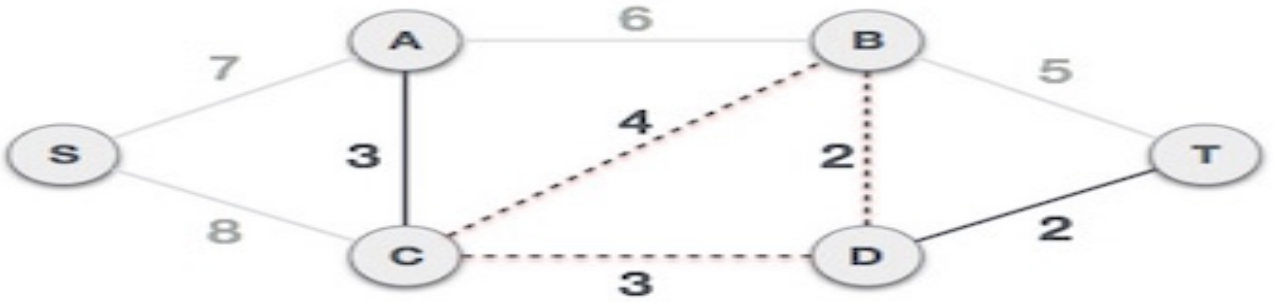


- The least cost is 2 and edges involved are B,D and D,T. We add them. Adding them does not violate spanning tree properties, so we continue to our next edge selection.
- Next cost is 3, and associated edges are A,C and C,D. We add them again –



Minimum Cost Spanning Tree

- Next cost in the table is 4, and we observe that adding it will create a circuit in th

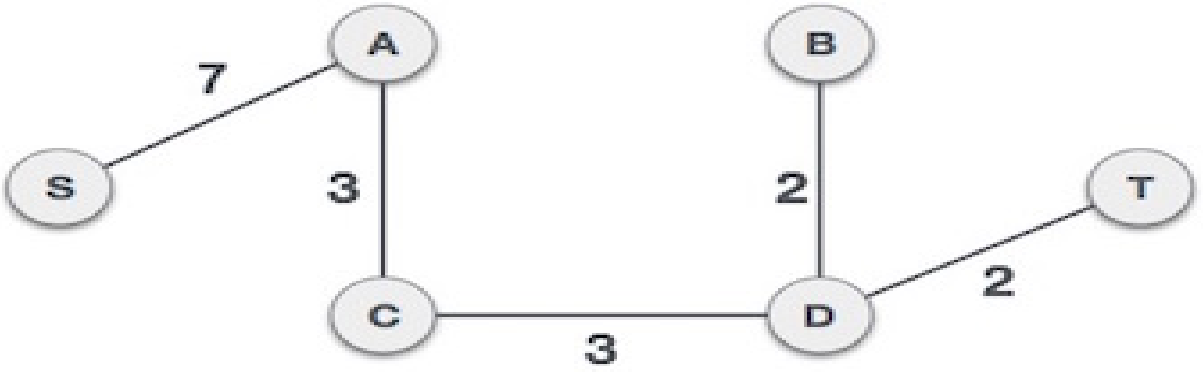


- We ignore it. In the process we shall ignore/avoid all edges that create a circuit.

- We observe that edges with cost 5 and 6 also create circuits. We ignore them and move

Minimum Cost Spanning Tree

- Now we are left with only one node to be added. Between the two least cost edges available 7 and 8, we shall add the edge with cost 7.



- By adding edge S,A we have included all the nodes of the graph and we now have minimum cost spanning tree.

Minimum Cost Spanning Tree

Kruskal's Algorithm

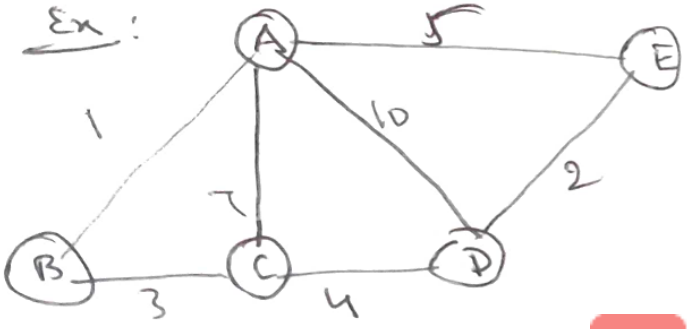
In this method, the edges of the graph are considered as non-decreasing order of edge costs. So arrange all the edges in decreasing order of its cost.

- Select minimum cost edge, verify whether the inclusion of this edge into spanning tree makes cycle. If not include it, otherwise discard it from consideration.
- Next select next minimum cost edge and do the same as above.
- Repeat the above selection procedure until including all the vertices (n-1 edges of n-vertex graph) into spanning tree.
- Before constructing spanning tree, first create forest with all the nodes and treat these nodes as individual sets.

Minimum Cost Spanning Tree

- After selecting minimum cost edge, verify these two vertices of edge are there in same set or different sets
  - If these two nodes are there in the same set, then adding this node makes cycle in the spanning tree.
  - If both the nodes are there in the different sets then add this edge into spanning tree and also combine the two sets which are having the above two nodes.
- Repeat the procedure until adding n-1 edges into spanning tree (until adding all the nodes into the spanning tree.

Ex:



Minimum Cost Spanning Tree

The weight of the edges are given as

Edge	AE	AD	AC	AB	BC	CD	DE
weight	5	10	7	1	3	4	2

Sort the edges according to their weights

Edge	AB	DE	BC	CD	AE	AC	AD
weight	1	2	3	4	5	7	10

and Consider all the nodes as forest.

{A} {B} {C} {D} {E}

Minimum Cost Spanning Tree

- First min cost edge is (A, B) with edge cost 1. Both the nodes are there in two different sets. So add this edge into spanning tree and combine those two sets

{A, B} {C} {D} {E}

- Next min cost edge is (D, E) with edge cost 2. Both D and E are there in two different sets. So add (D, E) into spanning tree and combine these two sets

{A, B} {C} {D, E}

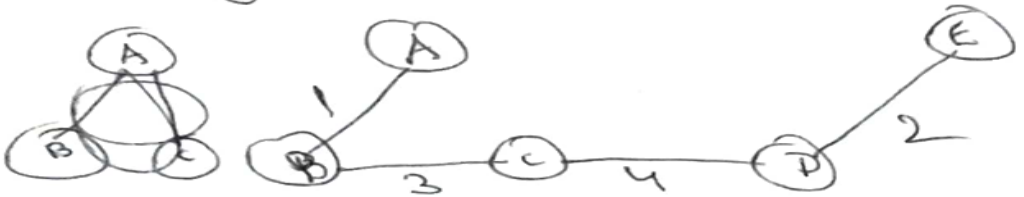
- Next min cost edge is (B, C). Both the nodes are there in two different sets, add them (combine two sets and add (B, C) into spanning tree

Minimum Cost Spanning Tree

{A, B, C} {D, E}

4

- Next minimum cost edge is (C, D), Both C and D are there in different sets, so add (C, D) into the spanning tree and combine these two sets



{A, B, C, D, E}

- Next min cost edge is (A, E), But both A and E are there in the Same ~~set~~ set so discard (A, E)
- Next min-cost edge is (A, C), Both A and C are in the Same set, inclusion of which makes cycle in the Spanning Tree.
- Next min-cost edge is (A, D), Both A and D are there in the Same set, discard it.
- No other edge is available in the graph and all nodes have included into Spanning Tree
- So the Cost of the Spanning Tree is 10

Minimum Cost Spanning Tree

```
Algorithm Kruskal (E, Cost, n, t)
{
Construct a heap out of the edge costs using Heapify;
for i := 1 to n do
  Parent[i] := -1;
i := 0; mincost := 0.0;
while (i < n-1) and (heap not empty) do
{
Delete a mincost edge (u, v) from the heap and
  reheapify using adjust.
j := Find(u); k := Find(v);
```

Minimum Cost Spanning Tree

```
    if (j ≠ k) then
        {
        i := i+1;
    t[i,1] := u; t[i,2] := v;
    mincost := mincost + cost[u,v];
    union (j, k);
        }
}
if (i ≠ n-1) then write ("No Spanning Tree");
else
return mincost;
}
```

Single Source Shortest Path Problem

Shortest distance from Source node to any node v is

$d[v] = d[u] + cost[u,v]$ or $d[v]$ which ever is min

is the result of $d[v]$ $\boxed{u \in S}$

$$d[v] = \min\{d[v], d[u] + cost[u,v]\}$$

Inorder to generate the shortest paths, we need to be able to determine

(a) the next vertex to which a shortest path must be generated and

(b) a shortest path to this vertex.

- Let S denote the set of vertices (including Source node) to which the shortest paths have already been generated from Source. For 'w' not in S, Let d[w] be the length of the shortest path starting from Source, going through only those vertices that are in S, and ending at w are observe that:

(1) If the next shortest path is to vertex u, then the path begin at Source node (v_0) and end at u, and goes through only those vertices that are in S.

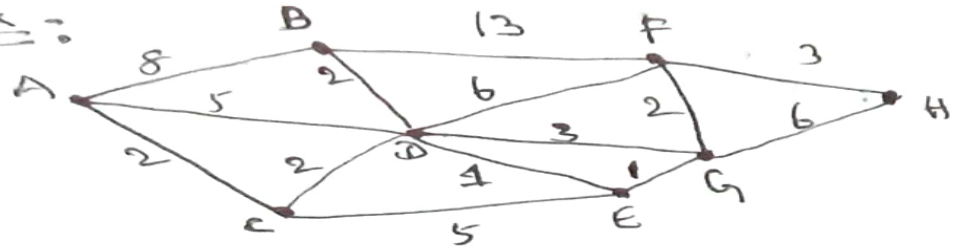
Single Source Shortest Path Problem

The shortest paths are being generated in non-decreasing order of path length.

Assume that there is a vertex w on the path that is not in S. Then, the V_0 to u path also contain a path from V_0 to w that is of length less than the V_0 to u path. By assumption the shortest paths are being generated in non-decreasing order of path length, and so the shortest path V_0 to w must already have been generated.

(2) The destination of the next path generated must be that of vertex u which has the min distance $d[u]$, among all vertices not in S

Ex:



Take Source node is A, and find the Shortest distance from node A to all the remaining nodes.

$$d[v] = \min \{ d[v],\ d[u] + \text{cost}[u,v] \}$$

Selected node	A	B	C	D	E	F	G	H
A	0_A	8_A	2_A	5_A	∞	∞	∞	∞
C		8_A	2_A	4_C	7_C	∞	∞	∞
D		6_D		4_C	5_D	10_D	7_D	∞
E		6_D			5_D	10_D	6_E	∞
B		6_D				10_D	6_E	∞
G						8_G	6_E	12_G
F						8_G		11_F
H								11_F

- Initially consider A as selected node and treat it as intermediate node and calculate the distance.

- From the first row select min value. Min value is 2, that is node C. So for the next row Consider node C as inter mediate node and update the distance

- $d[B] = \min\{d[B], d[C] + \text{Cost}[C,B]\}$
  $= \min\{8, 2+6\}$
  $= \min\{8, 8\} = 8$

- $d[D] = \min\{d[D], d[C] + \text{Cost}[C,D]\}$
  $= \min\{5, 2+2\}$
  $= \min\{5, 4\}$
  $= 4_C$

- $d[E] = \min\{d[E], d[C] + \text{Cost}[C,E]\}$
  $= \min\{\infty, 2+5\}$
  $= \min\{\infty, 7\}$
  $= 7_C$

- $d[F] = \min\{d[F], d[C] + \text{Cost}[C,F]\}$
  $= \min\{\infty, 2+\infty\} = \min\{\infty, \infty\}$
  $= \infty$

- $d[G] = \min\{d[G], d[C] + \text{Cost}[C,G]\}$
  $= \min\{\infty, 2+\infty\} = \min\{\infty, \infty\}$
  $= \infty$

- $d[H] = \min\{d[H], d[C] + \text{Cost}[C,H]\}$
  $= \min\{\infty, 2+\infty\} = \min\{\infty, \infty\}$
  $= \infty$

* From the second row select min value i.e 4. So next for row 3, node D as intermediate node and update distance

- $d[B] = \min\{d[B], d[D] + cost[D,B]\}$
  $= \min\{8, 4+2\} = \min\{8, 6\}$
  $= 6_D$
- $d[E] = \min\{d[E], d[D] + cost[D,E]\}$
  $= \min\{7, 4+1\} = \min\{7, 5\}$
  $= 5_D$
- $d[F] = \min\{d[F], d[D] + cost[D,F]\}$
  $= \min\{\infty, 4+6\} = \min\{\infty, 10\}$
  $= 10_D$
- $d[G] = \min\{d[G], d[D] + cost[D,G]\}$
  $= \min\{\infty, 4+3\} = \min\{\infty, 7\}$
  $= 7_D$
- $d[H] = \min\{d[H], d[D] + cost[D,H]\}$
  $= \min\{\infty, 4+\infty\} = \min\{\infty, \infty\}$
  $= \infty$

For the fourth row select min value in 3rd row & 5. So node E will be the next intermediate node. update row E.

Repeat the same for all nodes as intermediate node as above.

* Shortest Path Construction from A to all the remaining nodes:

(1) Path A to B :-

- Goto row which represented with B, and Select the min value represented in Square box, 6, and the Sub Script represented is D, So the previous node of B is D

D — 2 — B

- Next go to the row which is represented with D, select min value, i.e 4, the Subscript of 4 is C, So the previous node of D is C. The path upto this step is

C — 2 — D — 2 — B

- Next goto the row represented by C, and Select min value, i.e 2, and its Subscript is A, So the previous node of C is A, and A is the Source node. So the find shortest path from Source node A to B is

A — 2 — C — 2 — D — 2 — B

Path Cost is : 6

Like the above, we can Construct path from Source node A to any other node