

# UNIT# VI

## Greedy Algorithms

**Prof. Ritesh Upadhyay**

Faculty of Engineering and Technology



# Introduction

Let us start our discussion with simple theory that will give us an understanding of the Greedy technique. In the game of

Chess, every time we make a decision about a move, we have to also think about the future consequences. Whereas, in the

game of Tennis (or Volleyball), our action is based on the immediate situation. This means that in some cases making a

decision that looks right at that moment gives the best solution (Greedy), but in other cases it doesn't.

The Greedy technique is best suited for looking at the immediate situation.

**Greedy Strategy :** Greedy algorithms work in stages. In each stage, a decision is made that is good at that point, without

bothering about the future. This means that some local best is chosen. It assumes that a local good selection makes for a

global optimal solution.

**OR**

Greedy is an algorithmic paradigm that builds up a solution piece by piece, always choosing the next piece that offers the

most obvious and immediate benefit.

# Cont..

Most of the problem in Greedy Technique contains n-inputs. Our object is to find subset of given input which will satisfy our requirement.

- **Problem Definition:** Understanding the Problem clearly.
- **Finding Solution Space:** All Possible solution for the given n- input.
- **Feasible Solution:** Feasible solution is one of the solution in solution space, which will satisfy our condition(Maximum profit or Minimum Cost)
- **Optimal Solution:** Optimal solution is one of the feasible solution , which will give maximum profit or minimum cost.

**Advantages and Disadvantages of Greedy Method:** The main advantage of the Greedy method is that it is straightforward,

easy to understand and easy to code. In Greedy algorithms, once we make a decision, we do not have to spend time

reexamining the already computed values. Its main disadvantage is that for many problems there is no greedy algorithm.

That means, in many cases there is no guarantee that making locally optimal improvements in a locally optimal solution

gives the optimal global solution.

# Application of Greedy Technique

1. Prim's and Kruskal's algorithms for Minimum Spanning Tree
2. Shortest path in Weighted Graph [Dijkstra's and Bellman Ford Algorithm]
3. Coin change problem
4. Fractional Knapsack problem
5. Job scheduling algorithm

# 1. Fractional Knapsack problem/Knapsack problem

**One wants to pack  $n$  items in a Bag:**

**The  $i$ th item is worth  $v_i$  rupees and weighs  $w_i$  kg**

**Maximize the value but cannot exceed  $W$  kg**

**$v_i, w_i, W$  are integers**

**0-1 knapsack  $\rightarrow$  each item is taken or not taken**

**Fractional knapsack  $\rightarrow$  fractions of items can be taken**

# Knapsack Problem-1

Example:

I(item)	I1	I2	I3	I4	I5
W(weigh)	5	10	20	30	40
V(value)	30	20	100	90	160
V/W	6	2	5	3	4

Weight capacity = 60

Max. Value first:

$$\text{Total weight} = 40 + 20 = 60$$

$$\text{Total value} = 160 + 100 = 260$$

Min Weight first:

$$\text{Total weight} = 5 + 10 + 20 + 30 (5/6) = 60$$

$$\text{Total value} = 30 + 20 + 100 + 90 (5/6) = 260$$

Maximum value/Weight first:

$$\text{Total weight} = 5 + 20 + 40 (7/8) = 60$$

$$\text{Total value} = 30 + 100 + 160 (7/8) = 270$$

Note: Greedy knapsack will give optimal sol. Every time when we give priority to both profit/value and weight.

# Knapsack Problem-2

Given; capacity(M) = 12

Number of object(N) = 5

Object	Obj1	obj2	obj3	obj4	obj5
Profit(p)	5	2	2	4	5
Weight(w)	5	4	6	2	1

Find out the maximize the total profit considering that we are subject to an absolute weight limit(M)?

**Solution: Step-1**

Object	Profit	Weight	Profit/weight
Obj1	5	5	5/5=1
Obj2	2	4	2/4=0.5
Obj3	2	6	2/6=0.33
Obj4	4	2	4/2=2
Obj5	5	1	5/1=5

# Knapsack Problem-2

**Step-2:** Now arranging the profit/weight ratio of objects in Descending order, we get the following sequence:

Obj5>Obj4> Obj1>Obj2>Obj3


**Step-3:** Now take as much as possible of the density item which is not in the bag

Des.order	Obj5	Obj4	Obj1	Obj2	Obj3
Fractional Ratio	1	1	1	1	0
Profit*Ratio	5*1=5	4*1=4	5*1=5	2*1=2	6*0=0
Maximize Profit = 5+4+5+2+0=16					



# Algorithm of Fractional Knapsack problem

1) Compute value per size density for each item

For( $i=1;i \leq n;i++$ )   $O(n)$

{

$D[i] = p_i/w_i$

}

2) Sort each item by its value density in descending order   $O(n \log n)$

3) Take one by one object from the above array until capacity of bag becomes zero  
→  $O(n)$

Time Complexity:  $O(n \log n)$  for sorting and  $O(n)$  for greedy selections.

So the time complexity of Greedy knapsack problem is  $O(n \log n)$

# Knapsack Problem-3

Given; capacity(M) = 15

Number of object(N) = 7

Object	Obj1	Obj2	Obj3	Obj4	Obj5	Obj6	Obj7
Profit	10	5	15	7	6	18	3
Weight	2	3	5	7	1	4	1

## 2. Coin Change Problem

Given a value  $V$ , if we want to make a change for  $V$  Rs, and we have an infinite supply of each of the denominations in

Indian currency, i.e., we have an infinite supply of  $\{ 1, 2, 5, 10, 20, 50, 100, 500, 1000 \}$  valued coins/notes, what is the minimum number of coins and/or notes needed to make the change?

Examples:

Input:  $V = 70$

Output: 2 We need a 50 Rs note and a 20 Rs note.

Input:  $V = 121$

Output: 3 We need a 100 Rs note, a 20 Rs note and a 1 Rs coin.

# Coin Change Problem

Solution: Greedy Approach.

Approach: A common intuition would be to take coins with greater value first. This can reduce the total number of coins needed. Start from the largest possible denomination and keep adding denominations while the remaining value is greater than 0.

## Algorithm:

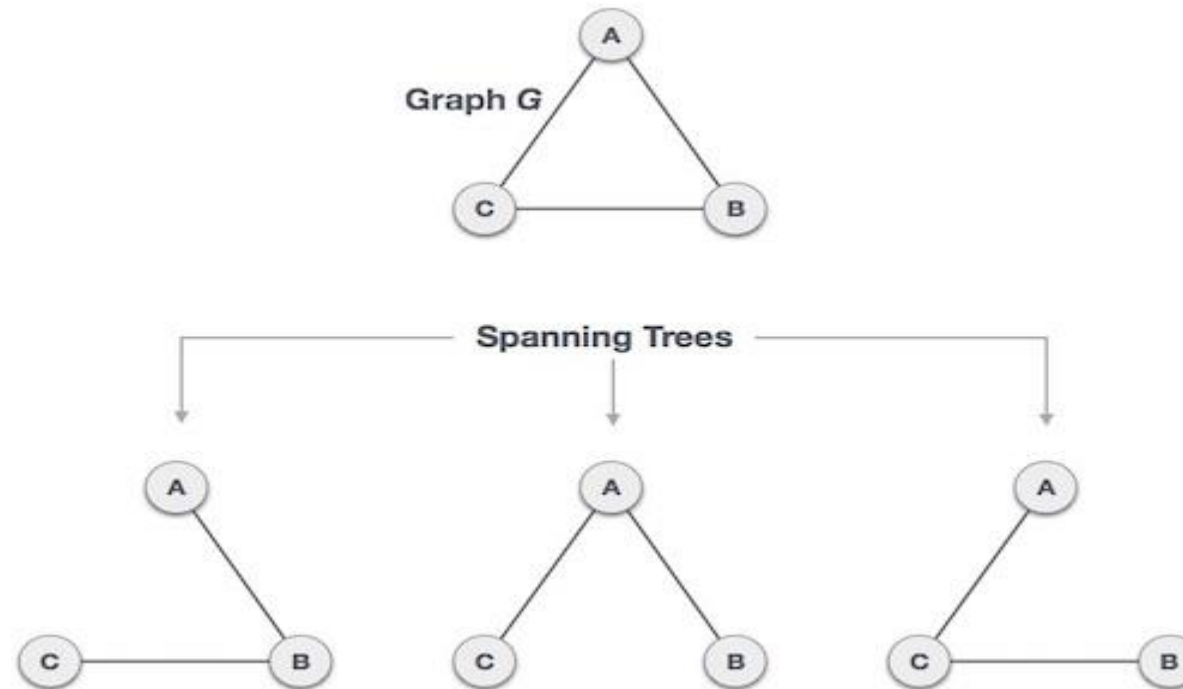
1. Sort the array of coins in decreasing order.
2. Initialize result as empty.
3. Find the largest denomination that is smaller than current amount.
4. Add found denomination to result. Subtract value of found denomination from amount.
5. If amount becomes 0, then print result.
6. Else repeat steps 3 and 4 for new value of V.

# Coin Change Problem

```
Function Coin Change(n){  
  Const C = {500, 100, 50, 20, 10, 5, 2, 1}  
  S =  $\Phi$   
  Sum = 0  
  while Sum  $\neq$  n do  
    x = largest item in C, such that Sum + x  $\leq$  n  
    if there is no such item  
      then return "No Solution"  
    else  
      S = S U { a coin of value x}  
      Sum = Sum + x  
  End while  
  return S }
```

# 3. Spanning Tree

A spanning tree is a subset of Graph  $G$ , which has all the vertices covered with minimum possible number of edges. Hence, a spanning tree does not have cycles and it cannot be disconnected..



# Cont...

We found three spanning trees off one complete graph. A complete undirected graph can have maximum  $n^{n-2}$  number of spanning trees, where  $n$  is the number of nodes. In the above addressed example,  $n$  is 3, hence  $3^{3-2} = 3$  spanning trees are possible.

## General Properties of Spanning Tree:

1. A connected graph  $G$  can have more than one spanning tree.
2. All possible spanning trees of graph  $G$ , have the same number of edges and vertices.
3. The spanning tree does not have any cycle (loops).
4. Spanning tree has  $n-1$  edges, where  $n$  is the number of nodes (vertices).
5. A complete graph can have maximum  $n^{n-2}$  number of spanning trees.

## Application of Spanning Tree:

1. Civil Network Planning
2. Computer Network Routing Protocol
3. Cluster Analysis

# Minimum Spanning Tree (MST)

In a weighted graph, a minimum spanning tree is a spanning tree that has minimum weight than all other spanning trees of the same graph. In real-world situations, this weight can be measured as distance, congestion, traffic load or any arbitrary value denoted to the edges.

## Minimum Spanning-Tree Algorithm

We shall learn about two most important spanning tree algorithms here –

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

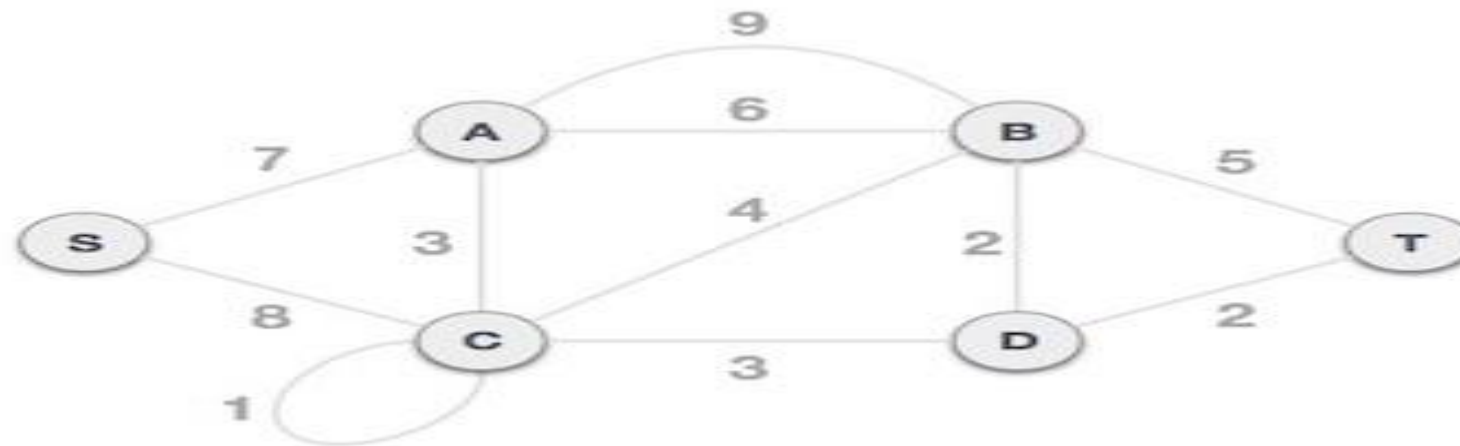
Both are greedy algorithms.



# Kruskal's Spanning Tree Algorithm

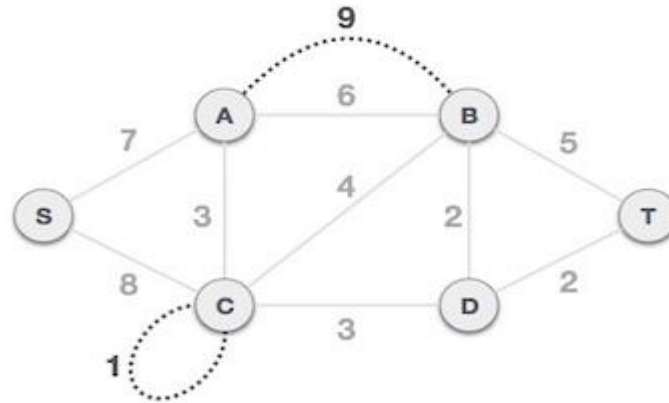
Kruskal's algorithm to find the minimum cost spanning tree uses the greedy approach. This algorithm treats the graph as a forest and every node it has as an individual tree. A tree connects to another only and only if, it has the least cost among all available options and does not violate MST properties.

To understand Kruskal's algorithm let us consider the following example –

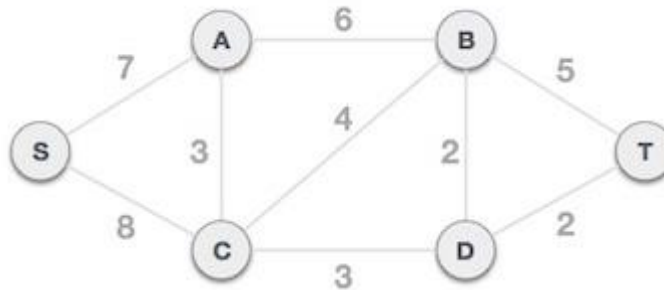


# Kruskal's Spanning Tree Algorithm

## Step 1 - Remove all loops and Parallel Edges



In case of parallel edges, keep the one which has the least cost associated and remove all others.



# Cont...

## 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

## Step 3 - Add the edge which has the least weightage

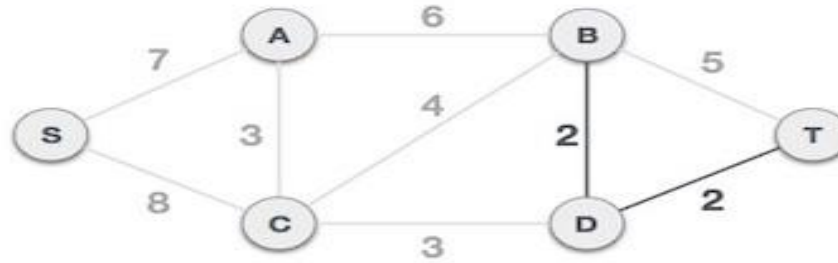
Now we start adding edges to the graph beginning from the one which has the least weight. Throughout, we shall keep

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.

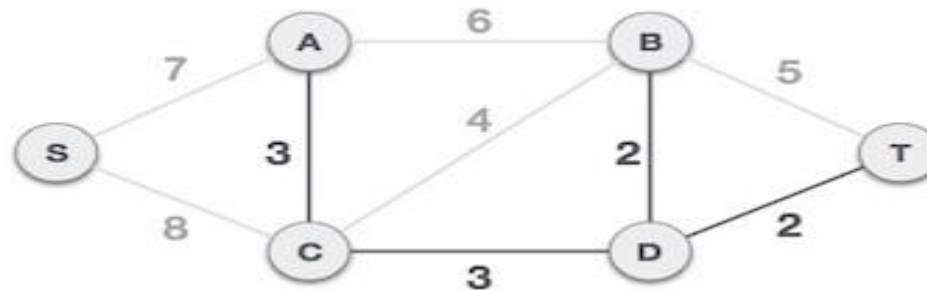
.

# Cont...



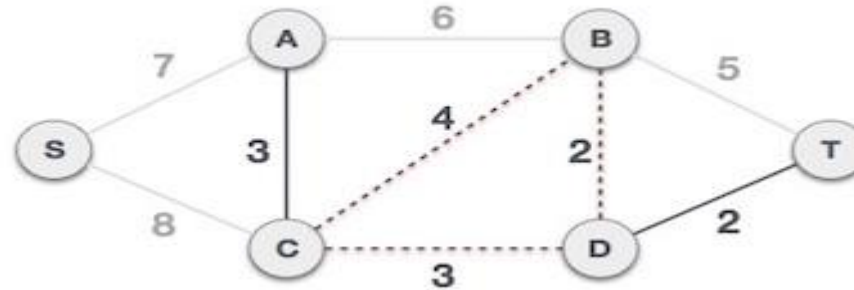
The least cost is 2 and edges involved are B,D and D,T. We add them. Adding them does not violate spanning tree properties, so we continue to our next edge selection.

Next cost is 3, and associated edges are A,C and C,D. We add them again –

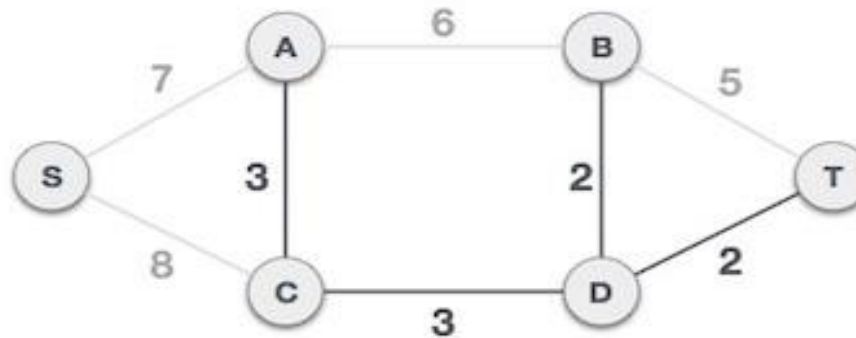


# Cont...

Next cost in the table is 4, and we observe that adding it will create a circuit in the graph. –

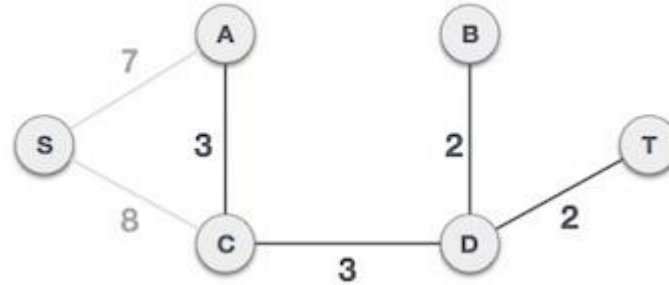


We ignore it. In the process we shall ignore/avoid all edges that create a circuit.

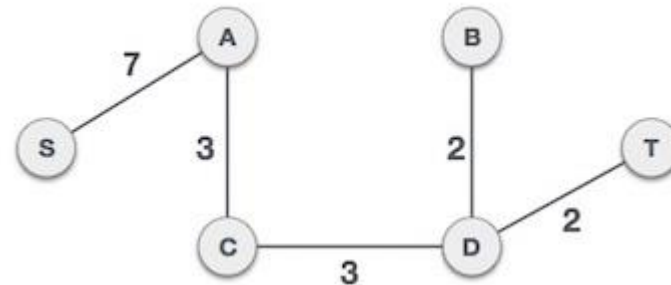


# Cont...

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



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..

# Kruskal's Spanning Tree Algorithm

Algo: MST\_KRUSKAL( $G$ )

```
1  A:= $\{ \}$ 
2  sort the edges of E by non-decreasing weight w
3  for each edge (u,v) in E, in order by non-decreasing weight
4      if FIND_SET(u)  $\neq$  FIND_SET(v)
5          then  A:=A $\cup$ {(u,v)}
6              UNION(u,v)
7  return A
```

## Kruskal's Algorithm Complexity

The time complexity Of Kruskal's Algorithm is:  $O(E \log E)$ .

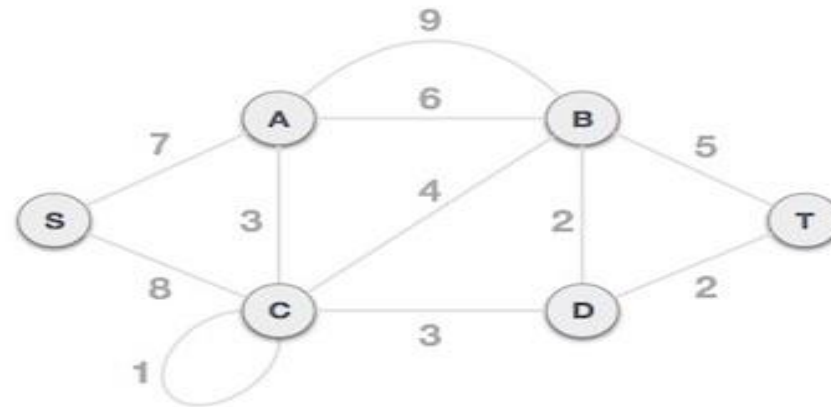
# Prim's Spanning Tree Algorithm

Prim's algorithm to find minimum cost spanning tree (as Kruskal's algorithm) uses the greedy approach. Prim's algorithm

shares a similarity with the shortest path first algorithms.

Prim's algorithm, in contrast with Kruskal's algorithm, treats the nodes as a single tree and keeps on adding new nodes to the spanning tree from the given graph.

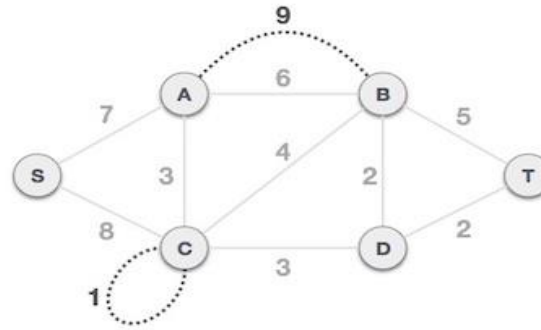
To contrast with Kruskal's algorithm and to understand Prim's algorithm better, we shall use the same example –



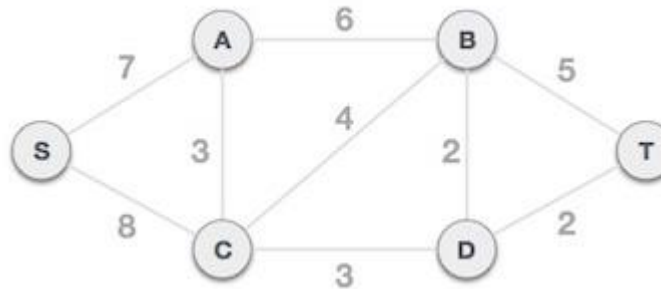


# Prim's Spanning Tree Algorithm

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.



# Cont...

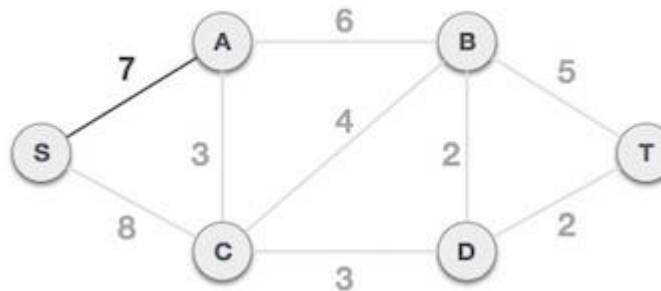
## Step 2 - **Choose any arbitrary node as root node**

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.

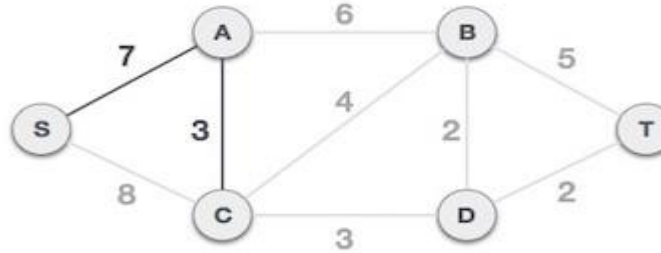
## Step 3 - **Check outgoing edges and select the one with less cost**

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.

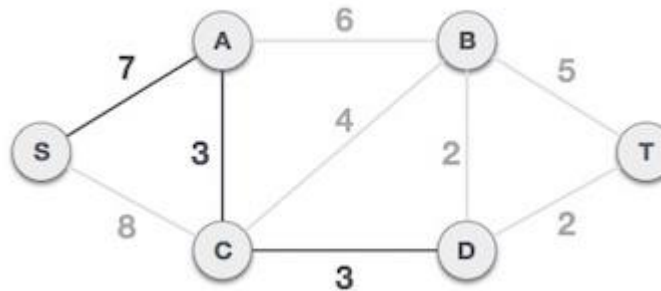


# Cont...

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.

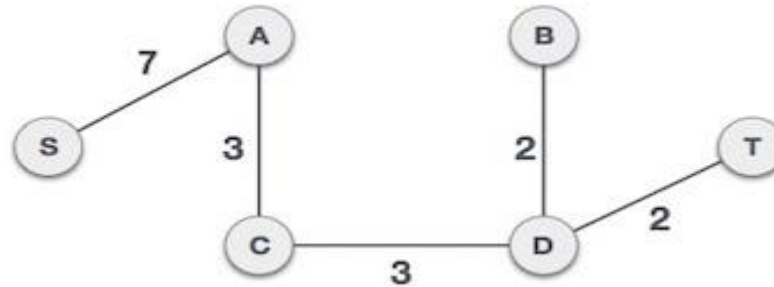


After this step, S-7-A-3-C tree is formed. Now we'll again treat it as a node and will check all the edges again. However, we will choose only the least cost edge. In this case, C-3-D is the new edge, which is less than other edges' cost 8, 6, 4, etc.



# Cont...

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.



We may find that the output spanning tree of the same graph using two different algorithms is same.

# Prim's Spanning Tree Algorithm

Algo: MST\_PRIM( $G$ )

1.  $A = \{\}$
2.  $S := \{r\}$  ( $r$  is an arbitrary node in  $V$ )
3.  $Q = V - \{r\}$ ;
4. while  $Q$  is not empty do {
- 5     take an edge  $(u, v)$  such that  $u \in S$  and  $v \in Q$  ( $v \notin S$ ) and  
       $(u, v)$  is the shortest edge
- 6     add  $(u, v)$  to  $A$ , add  $v$  to  $S$  and delete  $v$  from  $Q$
- }

**Prim's Algorithm Complexity:** The time complexity of Prim's algorithm is  $O(E \log V)$ .

# Important Points regarding to MST

1. For the any given if it have more than one Minimum Cost spanning tree , then in the given graph same adjacent edge repeated(Edge Repetition)
2. If the given graph contain edge repetition then we may get more than one MST or may be not.
3. For every connected graph at least one more MST should be possible.
4. Every MST must contain edge with minimum cost always.
5. If edge with maximum cost is on MST then it removal must disconnect the graph.
6. Any undirected graph  $G$  have  $n$ -nodes and all edges are distinct weight then  $G$  has unique MST.
7. Both kruskal's and prism algorithm of calculating MST having the same efficiency.

# 4. Shortest Path Algorithm

Path :  $V_1-V_2-V_3-V_4-V_5-V_6\ldots\ldots V_n$

$(V_1-V_2) (V_2-V_3) (V_3-V_4)(V_4-V_5)\ldots\ldots(V_{n-1} -V_n)$

Let  $d[u, v]$ = shortest path from  $u$  to  $v$

$d[u, v]$ = Min $\Rightarrow$  when all possible path from  $u$  to  $v$

$d[u, v]$ = Infinite $\Rightarrow$  when there is no path from  $u$  to  $v$

Note-1: Shortest path from  $u$  to  $v$  is not possible , if there is no path between  $u$  to  $v$

Note-2: We do not get shortest path from  $u$  to  $v$  ,if the given graph contain negative weight cycle.

# Single Source Shortest Path Algorithm

Find shortest path from given source to destination or all other remaining vertices.

There are two Algorithms are used:

- 1. Dijkstra's Algorithm [ it works only for nonnegative edge weights]**
- 2. Bellman Ford Algorithm[ it works for nonnegative edge weights and negative edge weights]**



# Dijkstra's shortest path algorithm

It is a greedy algorithm that solves the single-source shortest path problem for a directed graph  $G = (V, E)$  with nonnegative edge weights, i.e.,  $w(u, v) \geq 0$  for each edge  $(u, v) \in E$ .

Dijkstra's Algorithm maintains a set  $S$  of vertices whose final shortest - path weights from the source  $s$  have already been determined. That's for all vertices  $v \in S$ ; we have  $d[v] = \delta(s, v)$ . The algorithm repeatedly selects the vertex  $u \in V - S$  with the minimum shortest - path estimate, insert  $u$  into  $S$  and relaxes all edges leaving  $u$ .

Because it always chooses the "lightest" or "closest" vertex in  $V - S$  to insert into set  $S$ , it is called as the greedy strategy.

# Dijkstra's algorithm

Dijkstra'S Algorithm

$D = \emptyset$

for each vertex  $U \in V$

$d[S, U] = \infty$  // S is Source

$d[S, S] = 0$

$Q = V$

While(  $Q \neq \emptyset$  )

{

$U = \text{EXTRACT-MIN}(Q)$

$D = D \cup \{U\}$

For (each  $x \in \text{adj}(V)$ )

{

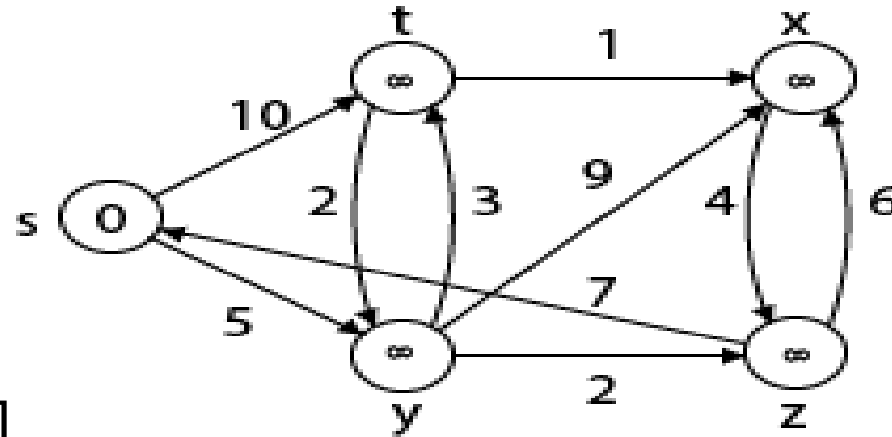
If ( $d[S, x] > d(S, u) + w(u, x)$ )

$d[S, x] = d(S, u) + w(u, x);$

}

}

# Example of Dijkstra's algorithm



Solution:

Step1:  $Q = [s, t, x, y, z]$

We scanned vertices one by one and find out its adjacent. Calculate the distance of each adjacent to the source vertices.

We make a stack, which contains those vertices which are selected after computation of shortest distance.

Firstly we take 's' in stack M (which is a source)

$M = [S]$        $Q = [t, x, y, z]$

# Example of Dijkstra's algorithm

Step 2: Now find the adjacent of s that are t and y.

Adj [s]  $\rightarrow$  t, y [Here s is u and t and y are v]

Case - (i) s  $\rightarrow$  t

$$\begin{aligned} d[v] &> d[u] + w[u, v] \\ d[t] &> d[s] + w[s, t] \\ \infty &> 0 + 10 \end{aligned} \quad \text{[false condition]}$$

$$\begin{aligned} \text{Then } d[t] &\leftarrow 10 \\ \pi[t] &\leftarrow s \end{aligned}$$

$$\text{Adj}[s] \leftarrow t, y$$

Case - (ii) s  $\rightarrow$  y

$$\begin{aligned} d[v] &> d[u] + w[u, v] \\ d[y] &> d[s] + w[s, y] \\ \infty &> 0 + 5 \end{aligned} \quad \text{[false condition]}$$

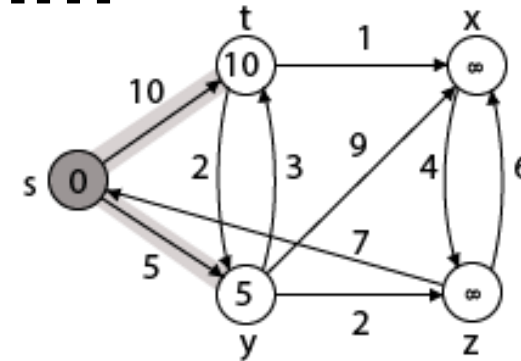
$$\begin{aligned} \text{Then } d[y] &\leftarrow 5 \\ \pi[y] &\leftarrow s \end{aligned}$$

By comparing case (i) and case (ii)

$$\text{Adj}[s] \rightarrow t = 10, y = 5$$

y is shortest

$$y \text{ is assigned in } 5 = [s, y]$$



# Example of Dijkstra's algorithm

Step 3: Now find the adjacent of y that is t, x, z.

Adj [y]  $\rightarrow$  t, x, z [Here y is u and t, x, z are v]

Case - (i) y  $\rightarrow$  t

$$\begin{aligned} d[v] &> d[u] + w[u, v] \\ d[t] &> d[y] + w[y, t] \\ 10 &> 5 + 3 \end{aligned}$$

$$\begin{aligned} \text{Then } d[t] &\leftarrow 8 \\ \pi[t] &\leftarrow y \end{aligned}$$

Case - (ii) y  $\rightarrow$  x

$$\begin{aligned} d[v] &> d[u] + w[u, v] \\ d[x] &> d[y] + w[y, x] \\ \infty &> 5 + 9 \\ \infty &> 14 \end{aligned}$$

$$\begin{aligned} \text{Then } d[x] &\leftarrow 14 \\ \pi[x] &\leftarrow y \end{aligned}$$

Case - (iii) y  $\rightarrow$  z

$$\begin{aligned} d[v] &> d[u] + w[u, v] \\ d[z] &> d[y] + w[y, z] \\ \infty &> 5 + 2 \\ \infty &> 7 \end{aligned}$$

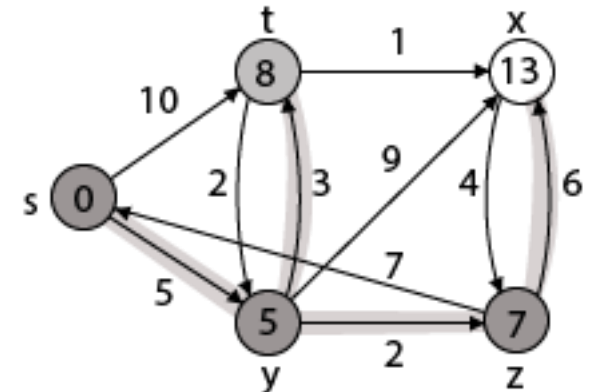
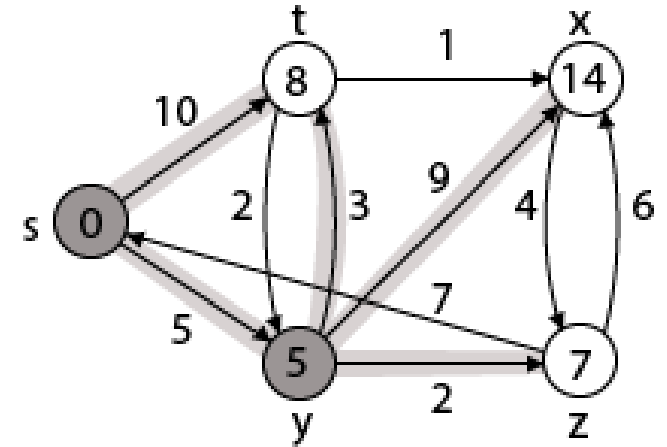
$$\begin{aligned} \text{Then } d[z] &\leftarrow 7 \\ \pi[z] &\leftarrow y \end{aligned}$$

By comparing case (i), case (ii) and case (iii)

Adj [y]  $\rightarrow$  x = 14, t = 8, z = 7

z is shortest

z is assigned in 7 = [s, z]



# Example of Dijkstra's algorithm

Step - 4 Now we will find adj [z] that are s, x

Adj [z]  $\rightarrow$  [x, s] [Here z is u and s and x are v]

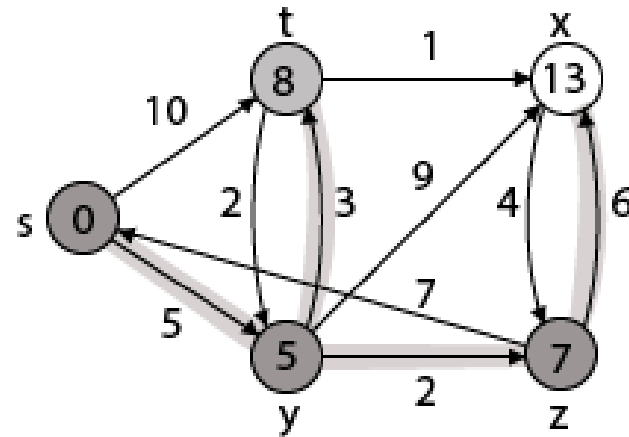
Case - (i) z  $\rightarrow$  x

$$\begin{aligned} d[v] &> d[u] + w[u, v] \\ d[x] &> d[z] + w[z, x] \\ 14 &> 7 + 6 \\ 14 &> 13 \end{aligned}$$

Then  $d[x] \leftarrow 13$   
 $\pi[x] \leftarrow z$

Case - (ii) z  $\rightarrow$  s

$$\begin{aligned} d[v] &> d[u] + w[u, v] \\ d[s] &> d[z] + w[z, s] \\ 0 &> 7 + 7 \\ 0 &> 14 \end{aligned}$$



$\therefore$  This condition does not satisfy so it will be discarded.  
 Now we have x = 13.

# Example of Dijkstra's algorithm

Step 5: Now we will find Adj [t]

Adj [t]  $\rightarrow$  [x, y] [Here t is u and x and y are v]

Case - (i)  $t \rightarrow x$

$$d[v] > d[u] + w[u, v]$$

$$d[x] > d[t] + w[t, x]$$

$$13 > 8 + 1$$

$$13 > 9$$

Then  $d[x] \leftarrow 9$

$$\pi[x] \leftarrow t$$

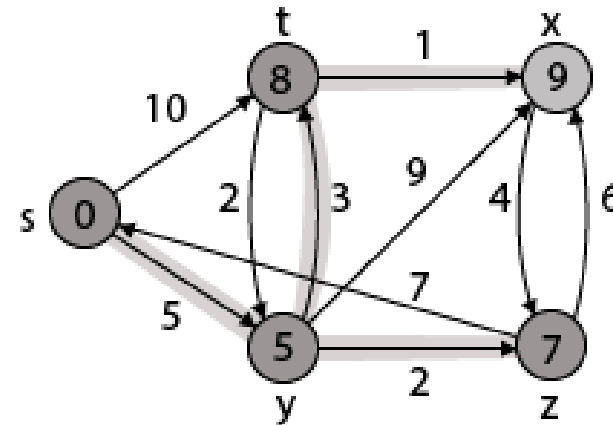
Case - (ii)  $t \rightarrow y$

$$d[v] > d[u] + w[u, v]$$

$$d[y] > d[t] + w[t, y]$$

$$5 > 10$$

$\therefore$  This condition does not satisfy so it will be discarded.



# Example of Dijkstra's algorithm

Thus we get all shortest path vertex as

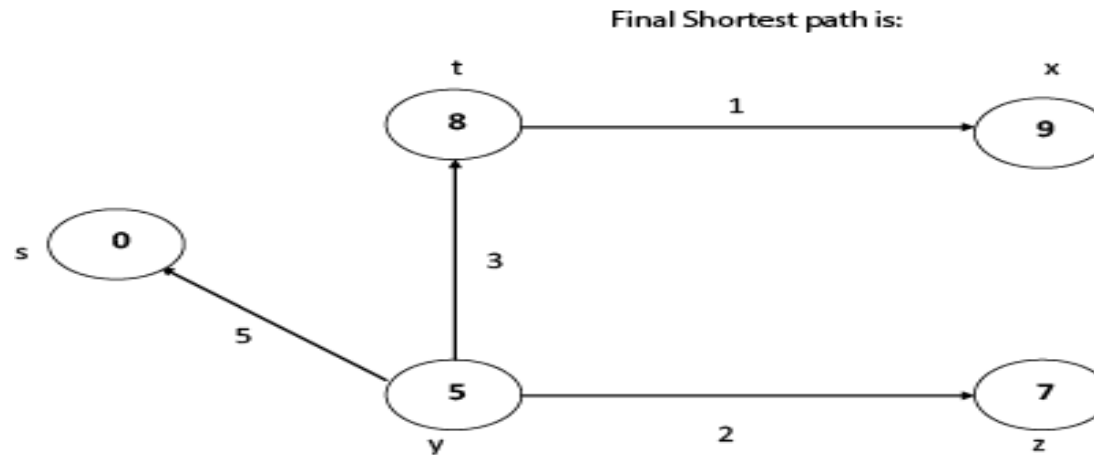
Weight from s to y is 5

Weight from s to z is 7

Weight from s to t is 8

Weight from s to x is 9

These are the shortest distance from the source's' in the given graph.

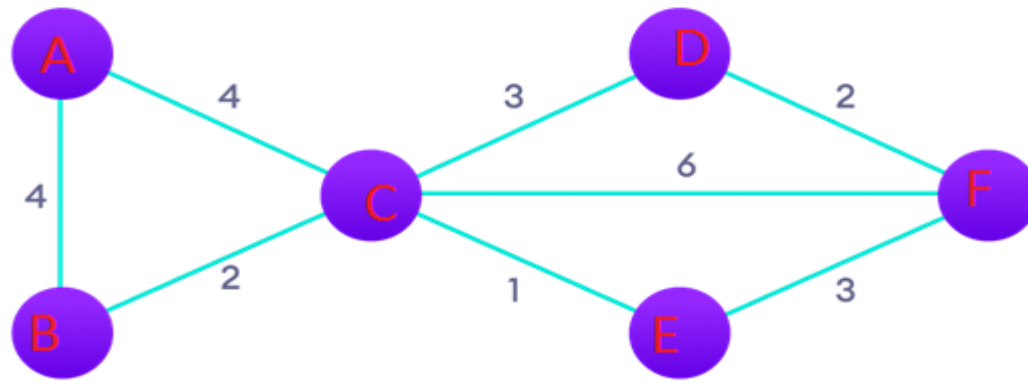




# Disadvantage of Dijkstra's Algorithm:

1. It does a blind search, so wastes a lot of time while processing.
2. It can't handle negative edges.

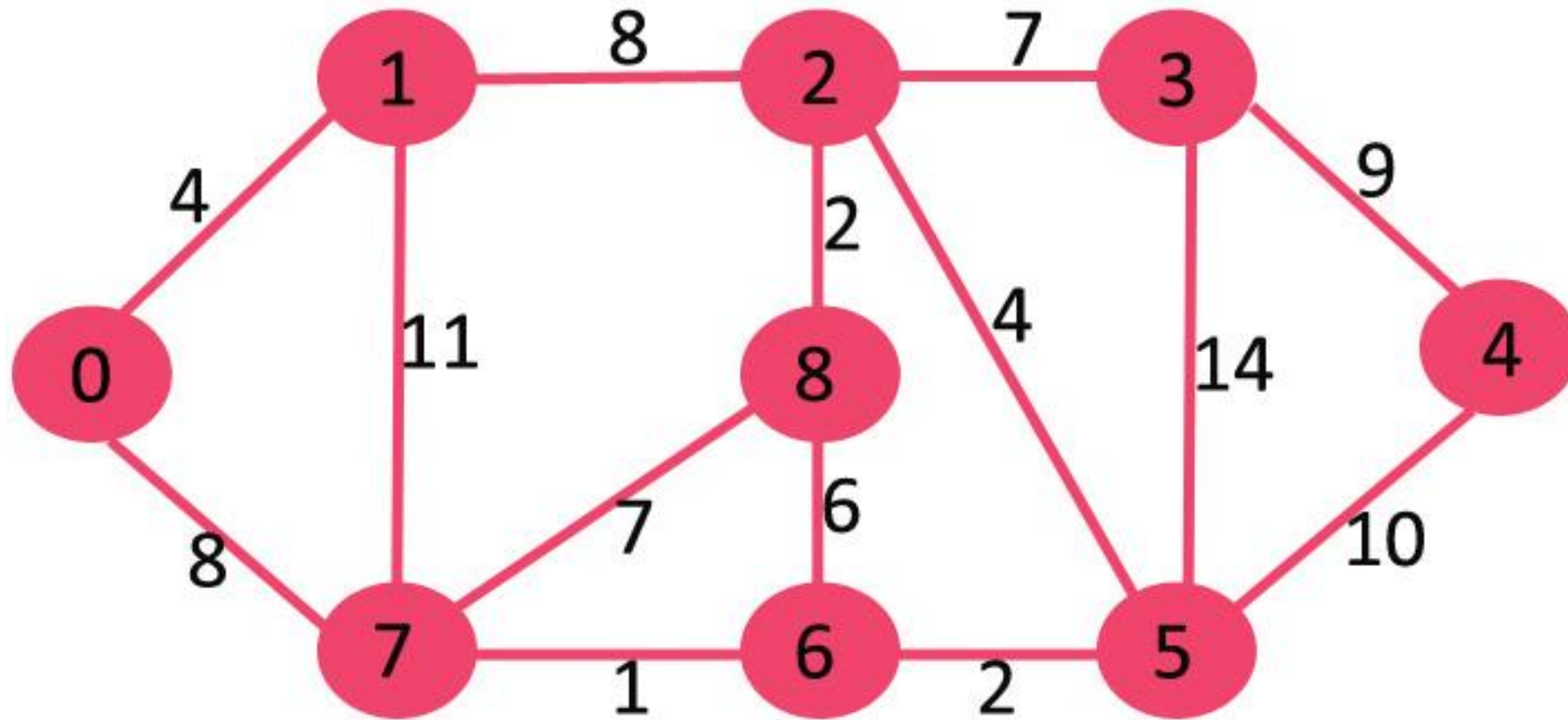
Starting node



	A	B	C	D	E	F
initially	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$
	0	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$
A	0	4	4	$\infty$	$\infty$	$\infty$
B		4	4	$\infty$	$\infty$	$\infty$
C			4	7	5	
E				7	5	8
D				7		
F						8
SHORTEST PATH BY USING DIJKSTRA'S ALGO : A-B-C-E-D-F						

# Example of Dijkstra's shortest path algorithm

Let us understand with the following example:



# Solution of Dijkstra's shortest path algorithm

	0	1	2	3	4	5	6	7	8
initially	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$
	0	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$
0	0	4	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$	8	$\infty$
1		4	12	$\infty$	$\infty$	$\infty$	$\infty$	8	$\infty$
7			12	$\infty$	$\infty$	$\infty$	9	8	15
6			12	$\infty$	$\infty$	11	9		15
5			12	25	21	11			15
2			12	25	21				15
8				19	21				15
3				19	21				
4					21				
SHORTEST PATH BY USING DIJKSTRA'S ALGO : 0-1-7-6-5-2-8-3-4									

## 5. Job Sequencing With Deadline

The sequencing of jobs on a single processor with deadline constraints is called as Job Sequencing with Deadlines.

Here-

1. You are given a set of jobs.
2. Each job has a defined deadline and some profit associated with it.
3. The profit of a job is given only when that job is completed within its deadline.
4. Only one processor is available for processing all the jobs.
5. Processor takes one unit of time to complete a job

**The problem states-**

“How can the total profit be maximized if only one job can be completed at a time?”

# Job Sequencing With Deadline

Greedy Algorithm- Greedy Algorithm is adopted to determine how the next job is selected for an optimal solution. The greedy algorithm described below always gives an optimal solution to the job sequencing problem-

## Step-01:

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

## Step-02:

- Check the value of maximum deadline.
- Draw a Gantt chart where maximum time on Gantt chart is the value of maximum deadline.

## Step-03:

- Pick up the jobs one by one.
- Put the job on Gantt chart as far as possible from 0 ensuring that the job gets completed before its deadline.

# PROBLEM BASED ON JOB SEQUENCING WITH DEADLINES-

Problem-

Given the jobs, their deadlines and associated profits as shown-

JOB	J1	J2	J3	J4	J5	J6
DEADLINE	5	3	3	2	4	2
PROFIT	200	180	190	300	120	100

Answer the following questions-

1. Write the optimal schedule that gives maximum profit.
2. Are all the jobs completed in the optimal schedule?
3. What is the maximum earned profit?

# PROBLEM BASED ON JOB SEQUENCING WITH DEADLINES-

Solution-

**Step-01:** Sort all the given jobs in decreasing order of their profit-

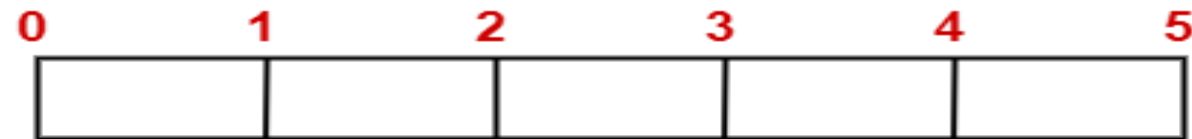
JOB	J4	J1	J3	J2	J5	J6
DEADLINE	2	5	3	3	4	2
PROFIT	300	200	190	180	120	100



# PROBLEM BASED ON JOB SEQUENCING WITH DEADLINES-

**Step-02:** Value of maximum deadline = 5.

So, draw a Gantt chart with maximum time on Gantt chart = 5 units as shown



**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.

# PROBLEM BASED ON JOB SEQUENCING WITH DEADLINES-

**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-



# PROBLEM BASED ON JOB SEQUENCING WITH DEADLINES-

**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-



# PROBLEM BASED ON JOB SEQUENCING WITH DEADLINES-

**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-

0	1	2	3	4	5
J2	J4	J3	J5	J1	

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.

# PROBLEM BASED ON JOB SEQUENCING WITH DEADLINES-

**Part-01:** 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.

**Part-02:** All the jobs are not completed in optimal schedule.

This is because job J6 could not be completed within its deadline.

**Part-03:** 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

# PROBLEM BASED ON JOB SEQUENCING WITH DEADLINES-

## Problem-1

Given the jobs, their deadlines and associated profits as shown-

JOB	$J_1$	$J_2$	$J_3$	$J_4$	$J_5$
DEADLINE	3	3	3	4	4
PROFIT	10	20	15	5	80

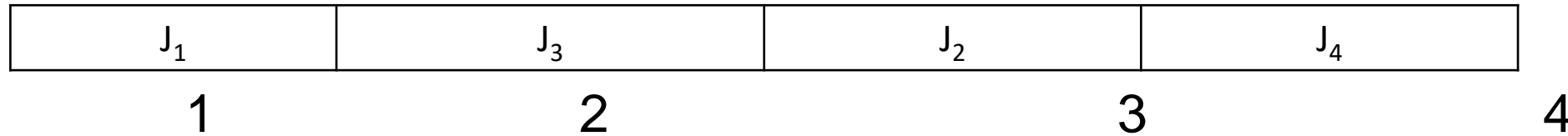
## Problem-2

Given the jobs, their deadlines and associated profits as shown-

JOB	$J_1$	$J_2$	$J_3$	$J_4$	$J_5$	$J_6$
DEADLINE	5	3	3	2	4	2
PROFIT	24	18	22	30	12	10

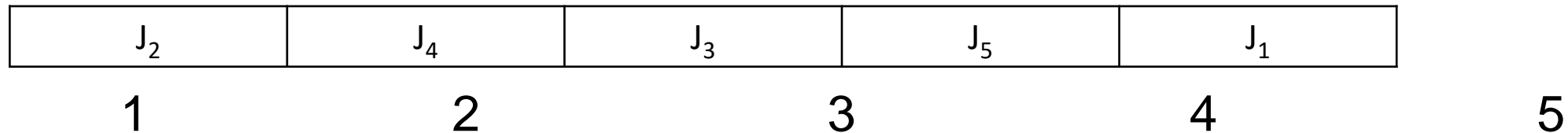
# PROBLEM BASED ON JOB SEQUENCING WITH DEADLINES-

## Solution-1



Maximum Profit =  $10+15+20+80 = 125$

## Solution-2



MAXIMUM PROFIT =  $18+30+22+12+24= 106$

# Thank You