Greedy Algorithm

★★★

P

walk, cycle, bike, Bus, car, train, flight

$$X \longrightarrow y$$

city city

constraints : 15 hrs.

min cost

max profit

min cost

optimization

Feasible solⁿ :

train

Greedy Algorithm

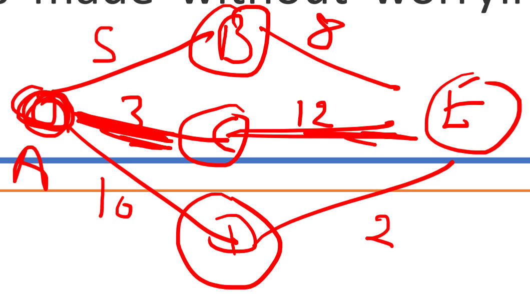
i. **The greedy method is one of the strategies like Divide and conquer used to solve the problems**.

ii. **This method is used for solving optimization problems.** An optimization problem is a problem that demands either maximum or minimum results.

iii. **The Greedy method is the simplest and straightforward approach.**

iv. The main function of this approach is that the decision is taken on the basis of the currently available information. Whatever the current information is present, the decision is made without worrying about the effect of the current decision in future.

v. **This technique is basically used to determine the feasible solution**



Components of Greedy Algorithm

The components that can be used in the greedy algorithm are:

- **Candidate set:** A solution that is created from the set is known as a candidate set.
- **Selection function:** This function is used to choose the candidate or subset which can be added in the solution.
- **Feasibility function:** A function that is used to determine whether the candidate or subset can be used to contribute to the solution or not.
- **Objective function:** A function is used to assign the value to the solution or the partial solution.
- **Solution function:** This function is used to intimate whether the complete function has been reached or not.

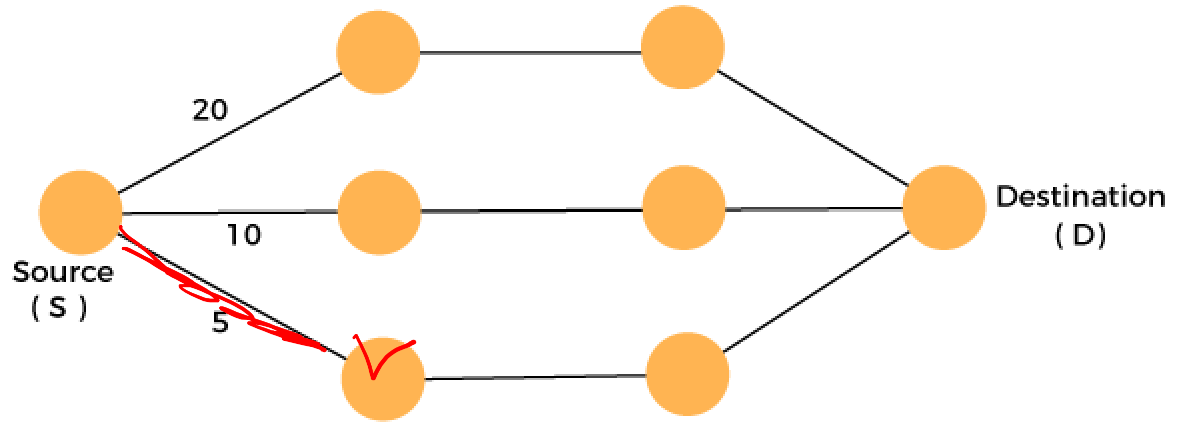
Greedy Algorithm

Applications of Greedy Algorithm

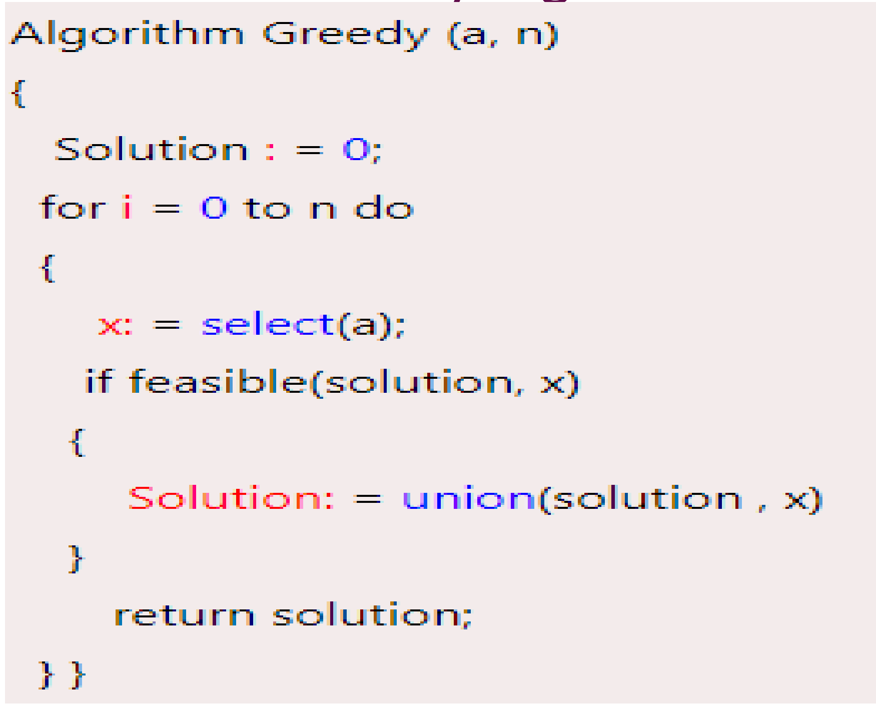
- It is used in finding the shortest path. ✓ ★★★
- It is used to find the minimum spanning tree using the prim's algorithm or the Kruskal's algorithm. ★★
- It is used in a job sequencing with a deadline. ★★
- This algorithm is also used to solve the fractional knapsack problem. ★★

* Huffman coding
* optimal merge pattern

It follows the local optimum choice at each stage with a intend of finding the global optimum. Let's understand through an example.



Pseudo code of Greedy Algorithm



```
Algorithm Greedy (a, n)
{
 Solution : = 0;
 for i = 0 to n do
 {
    x: = select(a);
   if feasible(solution, x)
   {
     Solution: = union(solution , x)
   }
    return solution;
} }
```

Knapsack Problem

- A knapsack (kind of shoulder bag) with limited weight capacity.
- Few items each having some weight and value.

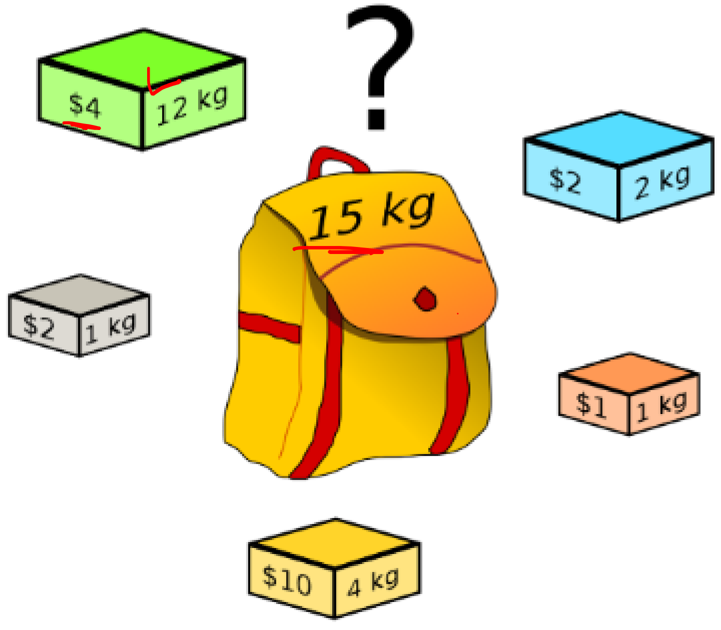
The problem states-

Which items should be placed into the knapsack such that-

- The value or profit obtained by putting the items into the knapsack is maximum.
- And the weight limit of the knapsack does not exceed.

Knapsack problem has the following two variants-

1. Fractional Knapsack Problem (Greedy ~~[illegible]~~ method)
2. 0/1 Knapsack Problem (Dynamic Prog.)



Knapsack Problem

Fractional Knapsack Problem-

In Fractional Knapsack Problem,

- As the name suggests, items are divisible here.
- We can even put the fraction of any item into the knapsack if taking the complete item is not possible.
- It is solved using Greedy Method.

Fractional Knapsack Problem Using Greedy Method-

Fractional knapsack problem is solved using greedy method in the following steps-

Step-01:

For each item, compute its value / weight ratio. P/W

Step-02:

Arrange all the items in decreasing order of their value / weight ratio.

Step-03:

Start putting the items into the knapsack beginning from the item with the highest ratio.

Put as many items as you can into the knapsack.

Item	profit	weight
I_1	P_1	w_1
I_2	P_2	w_2
I_3	P_3	w_3

capacity = m

(I) P_i / w_i

For the given set of items and knapsack capacity = 60 kg, find the optimal solution for the fractional knapsack problem making use of greedy approach.

Item	**Weight**	**Value**
1	5	30
2	10	40
3	15	45
4	22	77
5	25	90

Huffman Coding- ★★★

$$\begin{tikzcd} \boxed{S} \arrow[r] & \boxed{D} \end{tikzcd}$$

A B C A B B C C A D E E D ⇒ 13

⇒ 13 × 8 = 104 bits

ASCII

A = 65 = 01000001

B = 66 = 01000010

C = 67

D = 68

E = 69

compression

① Fixed length

② Variable length (Huffman)

Fixed

000 →	A
001 →	B
010 →	C
011 →	D
100 →	E

⇒ 13 × 3 = 39 bits + 5×3 + 5×8

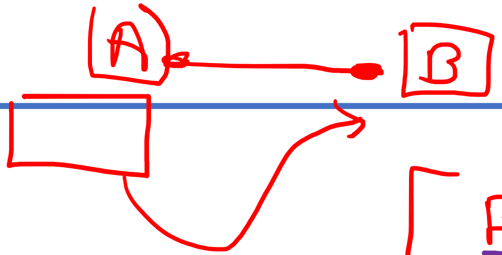
table

= 39 + 15 + 40

= 94 bit

Huffman Coding-

- Huffman Coding is a famous Greedy Algorithm.
- It is used for the lossless compression of data.
- It uses variable length encoding.
- It assigns variable length code to all the characters.
- The code length of a character depends on how frequently it occurs in the given text.
- The character which occurs most frequently gets the smallest code.
- The character which occurs least frequently gets the largest code.
- It is also known as **Huffman Encoding**.



Fixed length
variable length

A → 0
B → 01
C → 11

0 0 1 1 1
A A C

Prefix Rule

- Huffman Coding implements a rule known as a prefix rule.
- This is to prevent the ambiguities while decoding.
- It ensures that the code assigned to any character is not a prefix of the code assigned to any other character.

Huffman Tree-

The steps involved in the construction of Huffman Tree are as follows-

Step-01:

- Create a leaf node for each character of the text.
- Leaf node of a character contains the occurring frequency of that character.

Step-02:

- Arrange all the nodes in increasing order of their frequency value.

Step-03:

Considering the first two nodes having minimum frequency,

- Create a new internal node.
- The frequency of this new node is the sum of frequency of those two nodes.
- Make the first node as a left child and the other node as a right child of the newly created node.

Step-04:

- Keep repeating Step-02 and Step-03 until all the nodes form a single tree.
- The tree finally obtained is the desired Huffman Tree.

Overall time complexity of Huffman Coding **O(nlogn)**.

$$\text{Average code length per character} = \frac{\Sigma\,(\text{frequency}_i \times \text{code length}_i)}{\Sigma\,\text{frequency}_i}$$

or

$$= \Sigma\,(\text{probability}_i \times \text{code length}_i)$$

Total number of bits in Huffman encoded message

= Total number of characters in the message x Average code length per character

= ∑ (frequencyi x Code lengthi)

Algorithm of Huffman Code

```
Huffman (C)
1. n=|C|
2. Q ← C
3. for i=1 to n-1
4. do
5. z= allocate-Node ()
6. x=  left[z]=Extract-Min(Q)
7. y= right[z] =Extract-Min(Q)
8. f [z]=f[x]+f[y]
9. Insert (Q, z)
10. return Extract-Min (Q)
```

A AABAAAAABBB

Prefix code

Major Steps in Huffman Coding-

There are two major steps in Huffman Coding-

1. Building a Huffman Tree from the input characters
2. Assigning code to the characters by traversing the Huffman Tree

Huffman tree

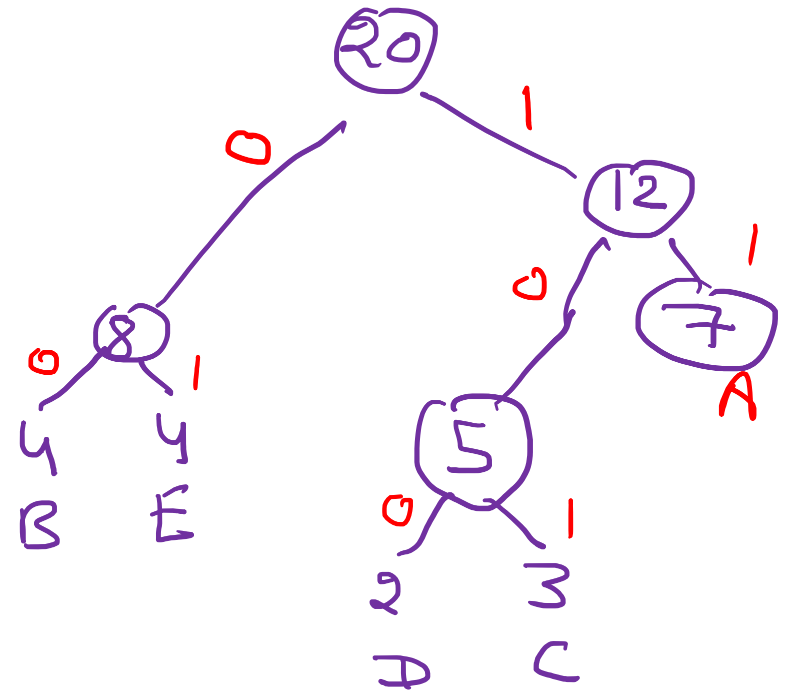
Ascending order

D	C	B	E	A
2	3	4	4	7

Characters	Frequencies
A	7
B	4
C	3
D	2
E	4

160

A →	11 →	2
B →	00 →	2
C →	101 →	3
D →	100 →	3
E →	01 →	2



Huffmans

total length = 7×2 + 4×2 + 3×3 + 2×3 + 4×2
= 14 + 8 + 9 + 6 + 8 = 45 + 12 + 8×5 =

45 + 12 + 40
= 97 bits

D 181, 184, 185, 192, 194, ~~195~~, 197, ~~200~~, 201, ~~211~~, ~~220~~
222, ~~226~~, 2[illegible], 232, 233, ~~234~~, 235, 239, ~~240~~, Harsh, Swami, ~~Mayank~~

I 484, 490, ~~493~~, 496, 504, 513, 515, 521,
L03, L26, L29,

A file contains the following characters with the frequencies as shown. If Huffman Coding is used for data compression, determine-

1. Huffman Code for each character ✓
2. Average code length
3. Length of Huffman encoded message (in bits)

✓

Characters	Frequencies
a	10
e	15
i	12
o	3
u	4
s	13
t	1

A text is made up of the characters a, b, c, d, e each occurring with the probability 0.11, 0.40, 0.16, 0.09 and 0.24 respectively. The optimal Huffman coding technique will have the average length of:

Job Sequencing with Deadlines

Job Sequencing with Deadlines problem uses the greedy approach.
The sequencing of jobs on a single processor with deadline constraints is called as Job Sequencing with Deadlines.

Points to Remember for Job Sequencing with Deadlines

- Each job has deadline d_i & it can process the job within its deadline; only one job can be processed at a time.
- Only one CPU is available for processing all jobs.
- CPU can take only one unit at a time for processing any job.
- All jobs arrived at the same time. ✓

P_i	1	2	3	4
d_i	3	4	5	1
p_i	80	30	40	90

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.

Jobs	J1	J2	J3	J4	J5	J6
Deadlines	5	3	3	2	4	2
Profits	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?

Step-01:

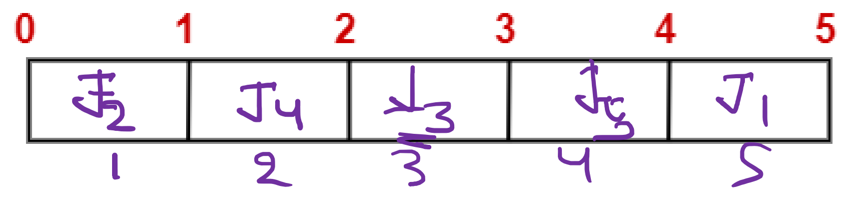
Sort all the given jobs in decreasing order of their profit-

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 maximum time on Gantt chart = 5



Gantt Chart

Now,

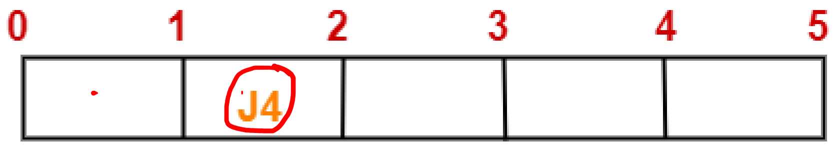
•We take each job one by one in the order they appear in Step-01.

•We place the job on Gantt chart as far as possible from 0.

J2 J4 J3 J5 J1

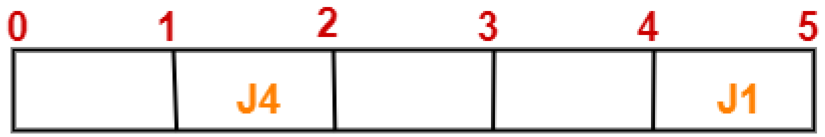
180 + 300 + 190 + 120 + 200

Step-03:



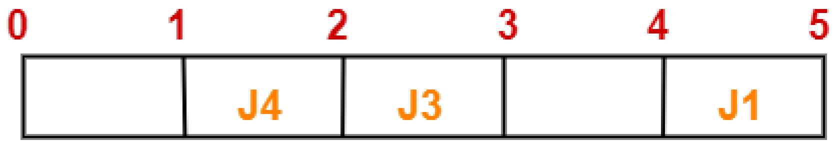
- We take job J4.
- Since its deadline is 2, so we place it in the first empty cell before deadline 2

Step-04:



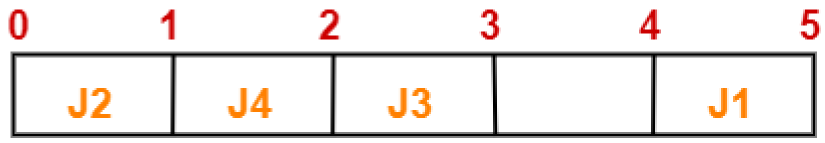
- We take job J1.
- Since its deadline is 5, so we place it in the first empty cell before deadline 5

Step-05:



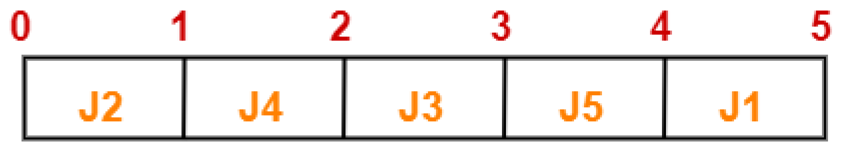
- We take job J3.
- Since its deadline is 3, so we place it in the first empty cell before deadline 3

Step-06:



- We take job J2.
- Since its deadline is 3, so we place it in the first empty cell before deadline 3

Step-07:



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

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

- 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
= 180 + 300 + 190 + 120 + 200

Job	**J_1**	**J_2**	**J_3**	**J_4**	**J_5**
Deadline	2	1	1	2	3
Profit	40	100	20	60	20

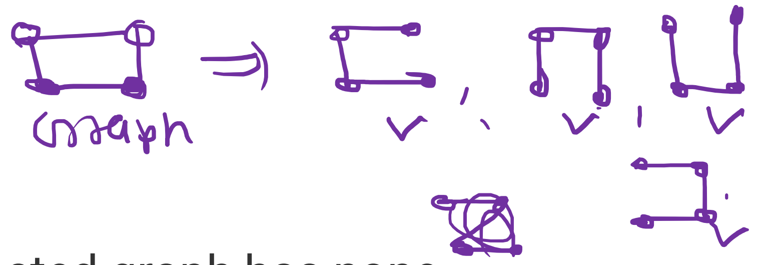
JobID	*Deadline*	*Profit*
a	*2*	*100*
b	*1*	*19*
c	*2*	*27*
d	*1*	*25*
e	*3*	*15*

	1	2	3	4	5	6	7
d_i	4	2	4	3	1	4	6
w_i	70	60	50	40	30	20	10

Spanning Tree

A spanning tree is defined as a subset of Graph G that has all the vertices covered with the minimum possible number of edges. Let us suppose a Graph G which has n number of vertices and e number of edges, then:

- Spanning tree T of Graph G contains n vertices and (n-1) edges.
- Spanning tree T has no cycles.



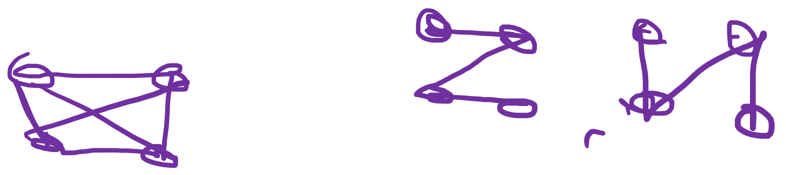
Spanning Tree Properties

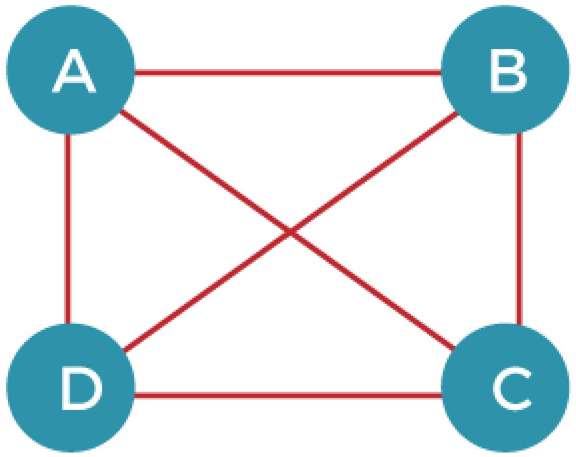
A connected, undirected graph has at least one spanning tree, while a disconnected graph has none.

properties:

- For a graph, all the possible spanning trees have the same number of vertices and edges.
- A spanning tree never contains a loop or a cycle, which is the property of a tree.
- A graph can have more than one spanning tree.
- A complete graph has maximum n^{n-2} spanning trees.

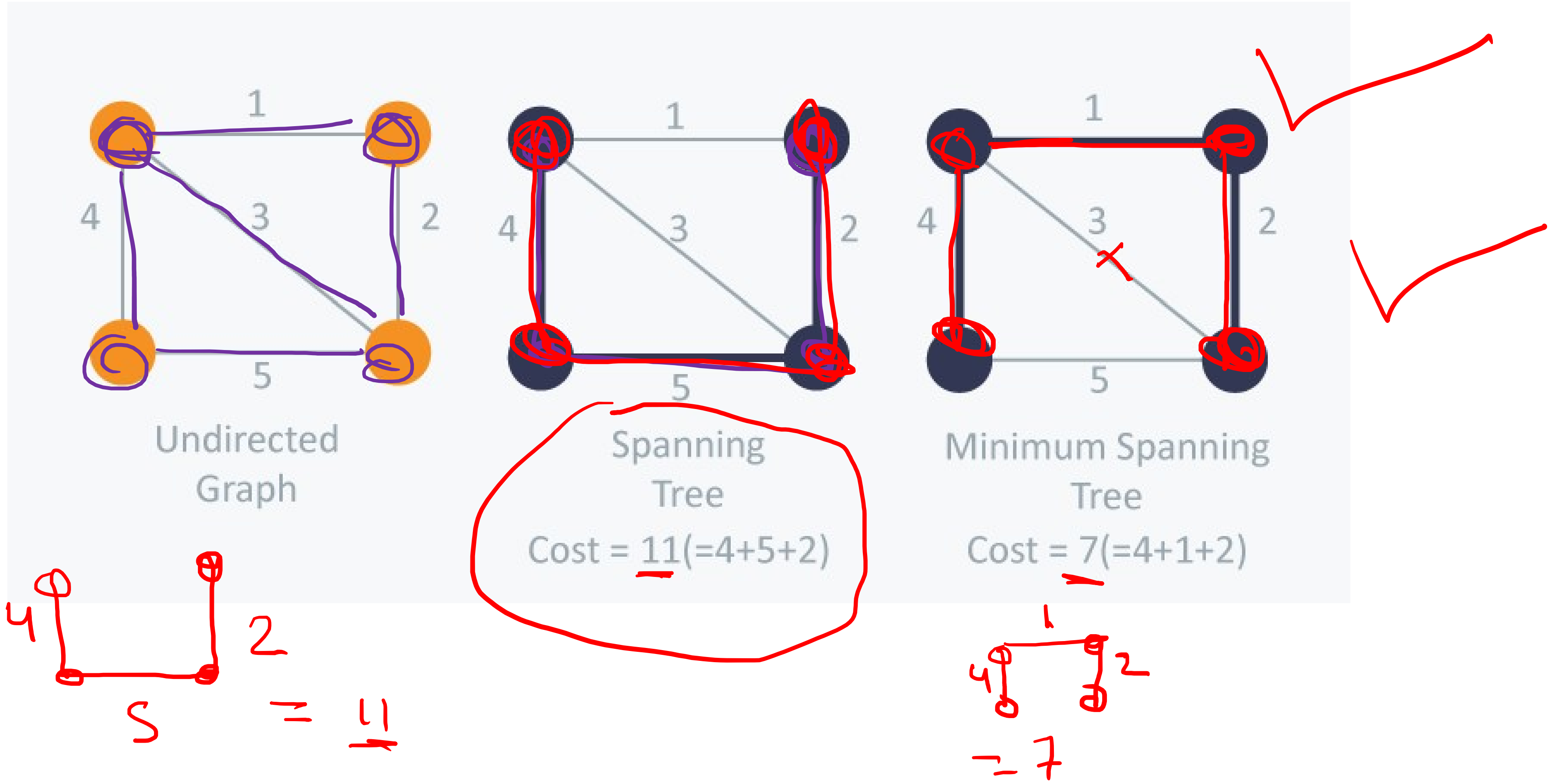
$4^{4-2} =$



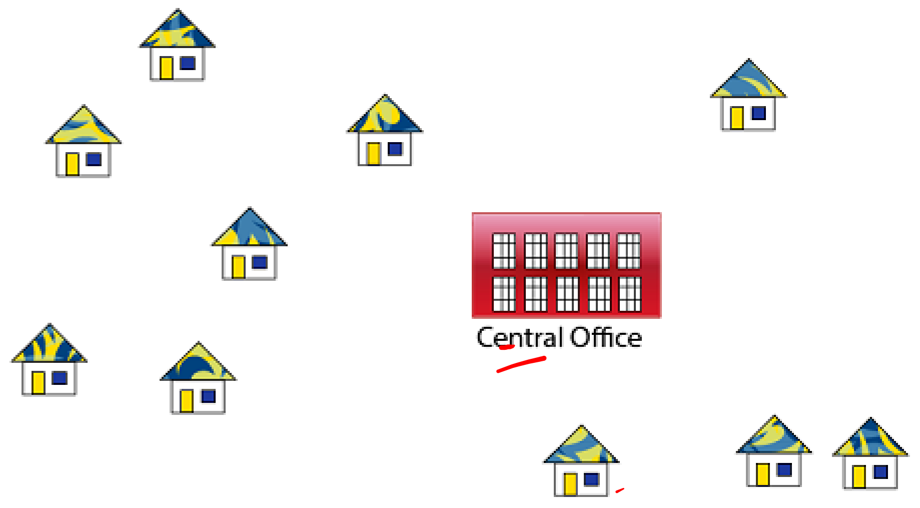

A
B
D
C

minimum spanning tree

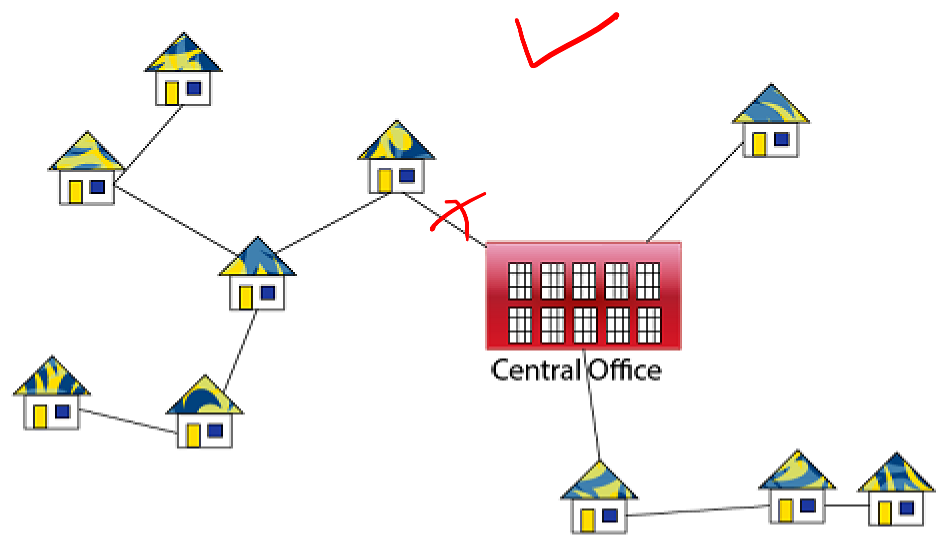
1. The minimum spanning tree is a spanning tree whose sum of the edges is minimum.
2. A minimum cost spanning tree abbreviated as MST is a subset of a connected, weighted, and an undirected graph that connects all the vertices of the graph with the minimum possible weight of the edges.



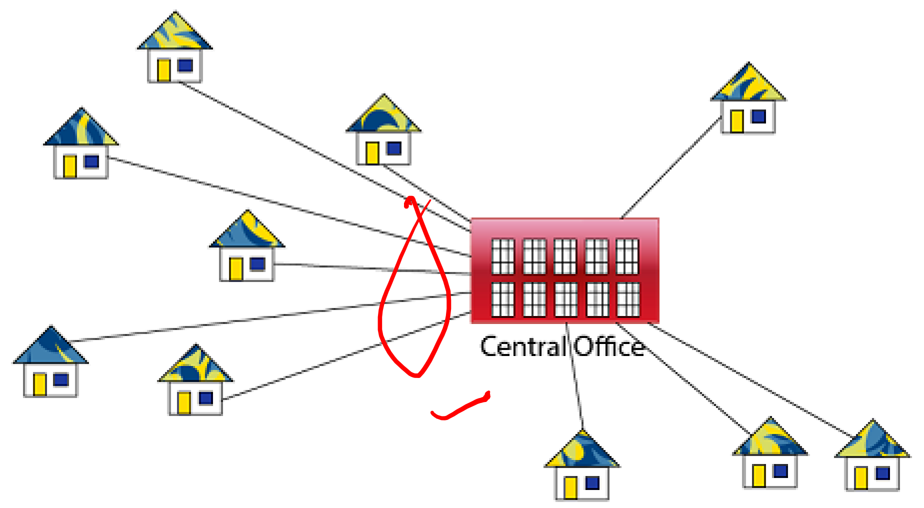
For Example, Problem laying Telephone Wire.



Wiring : Better Approach



Wiring : Naive Approach



Minimize the total length of wire connecting the customers

Application of Minimum Spanning Tree

1.Consider n stations are to be linked using a communication network & laying of communication links between any two stations involves a cost. The ideal solution would be to extract a subgraph termed as minimum cost spanning tree.

2.Suppose you want to construct highways or railroads spanning several cities then we can use the concept of minimum spanning trees.

3.Designing Local Area Networks.

4.Laying pipelines connecting offshore drilling sites, refineries and consumer markets.

5.Suppose you want to apply a set of houses with

1. Electric Power ✓
2. ✓ Water
3. ✓ Telephone lines
4. ✓ Sewage lines

To find the minimum cost spanning tree for a graph, two algorithms are used:

- Prims Minimum Spanning Tree Algorithm ✓
- Kruskal's Minimum Spanning Tree Algorithm ✓

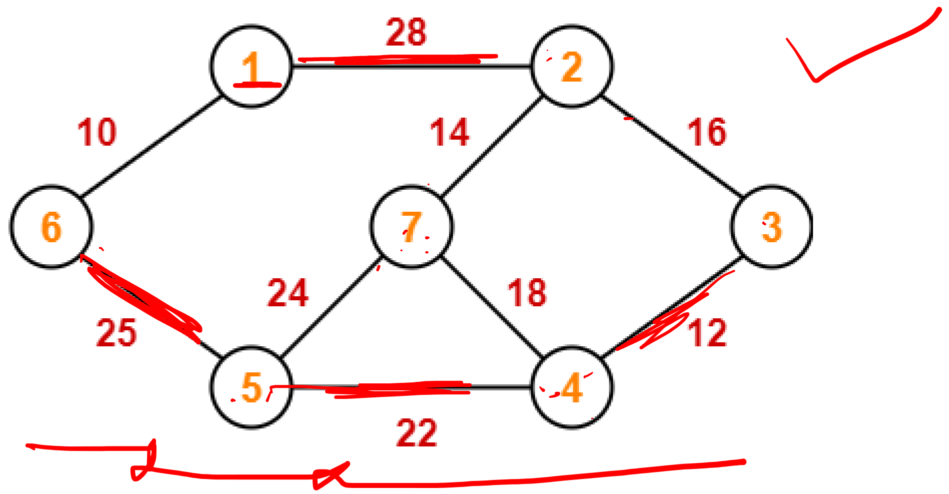
Prim’s Algorithm-

- Prim’s Algorithm is a famous greedy algorithm.
- It is used for finding the Minimum Spanning Tree (MST) of a given graph.
- To apply Prim’s algorithm, the given graph must be weighted, connected and undirected.

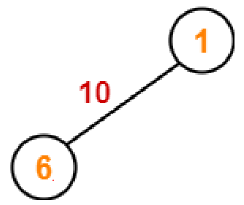
Prim’s Algorithm Time Complexity-

Worst case time complexity of Prim’s Algorithm is-

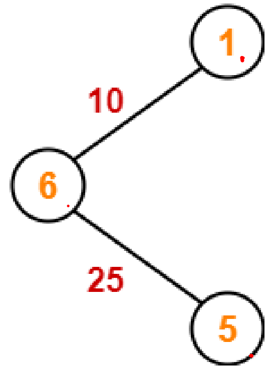
- O(ElogV) using binary heap
- O(E + VlogV) using Fibonacci heap



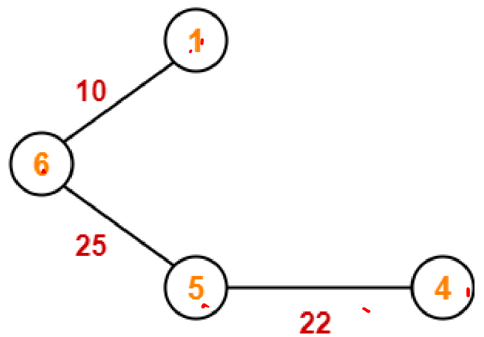
Step-01:



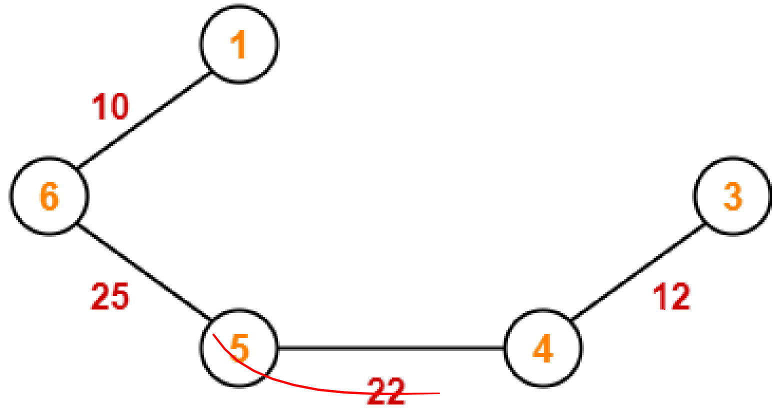
Step-02:



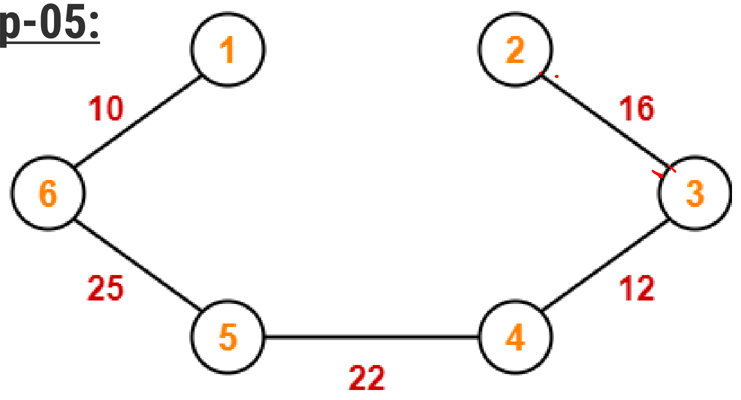
Step-03:



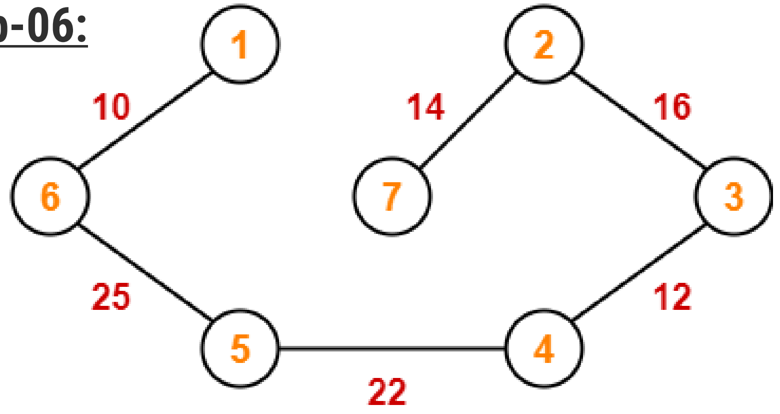
Step-04:



Step-05:

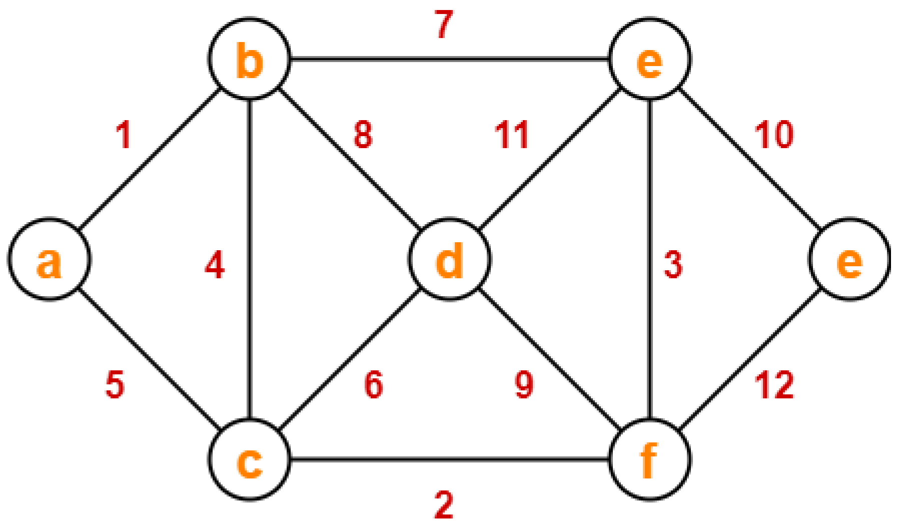


Step-06:



Cost of Minimum Spanning Tree
= Sum of all edge weights
= 10 + 25 + 22 + 12 + 16 + 14
= 99 units

Using Prim’s Algorithm, find the cost of minimum spanning tree (MST) of the given graph



Kruskal’s Algorithm-

•Kruskal’s Algorithm is a famous greedy algorithm.
•It is used for finding the Minimum Spanning Tree (MST) of a given graph.
•To apply Kruskal’s algorithm, the given graph must be weighted, connected and undirected.

Step-01:

•Sort all the edges from low weight to high weight.

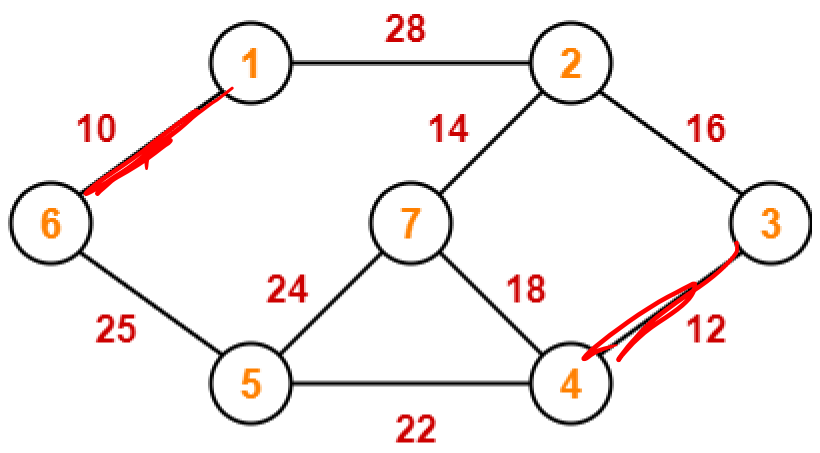
Step-02:

•Take the edge with the lowest weight and use it to connect the vertices of graph.
•If adding an edge creates a cycle, then reject that edge and go for the next least weight edge.

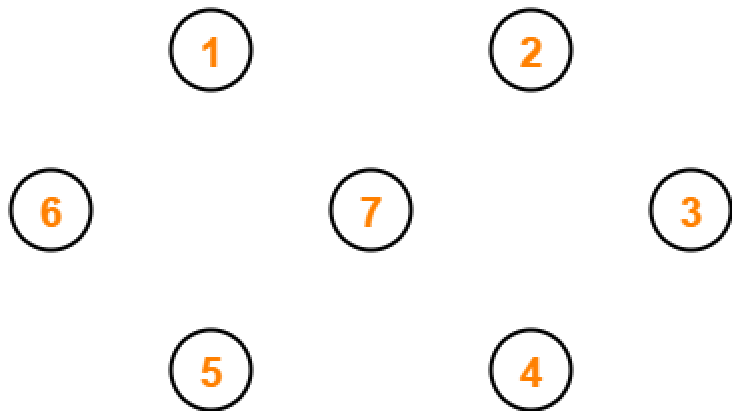
Step-03:

•Keep adding edges until all the vertices are connected and a Minimum Spanning Tree (MST) is obtained.

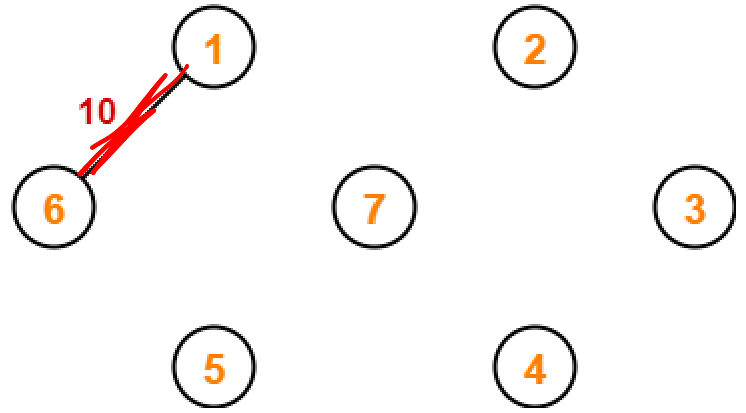
Worst case time complexity of Kruskal's Algorithm = O(ElogV) or O(ElogE)



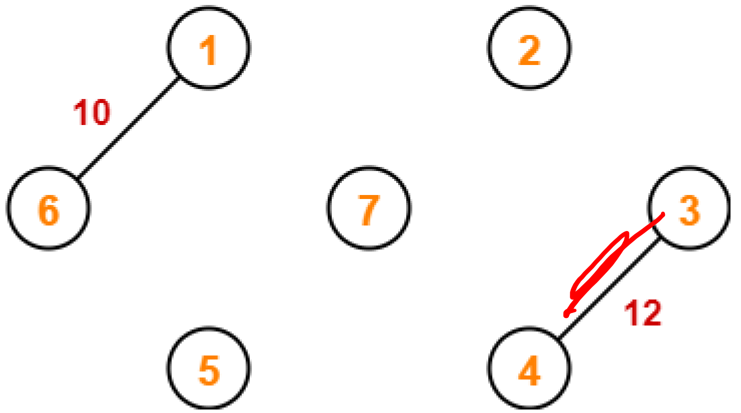
Step-01:



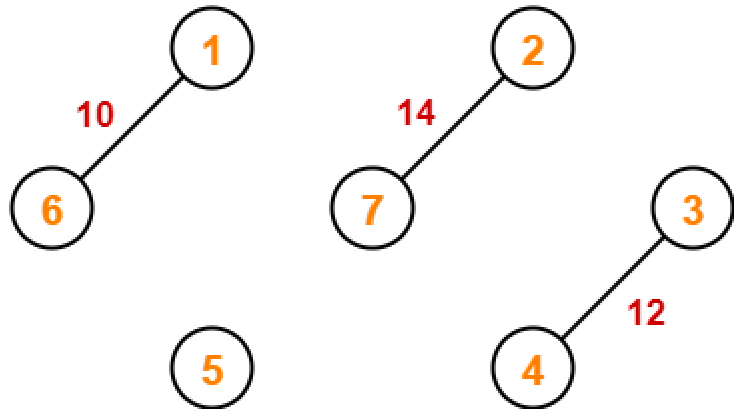
Step-02:



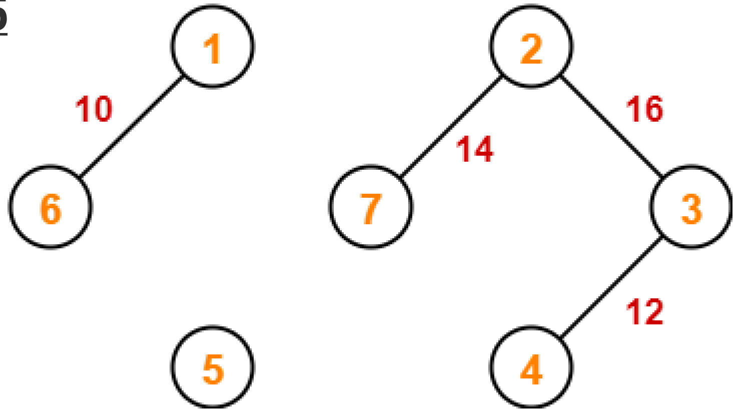
Step-03:



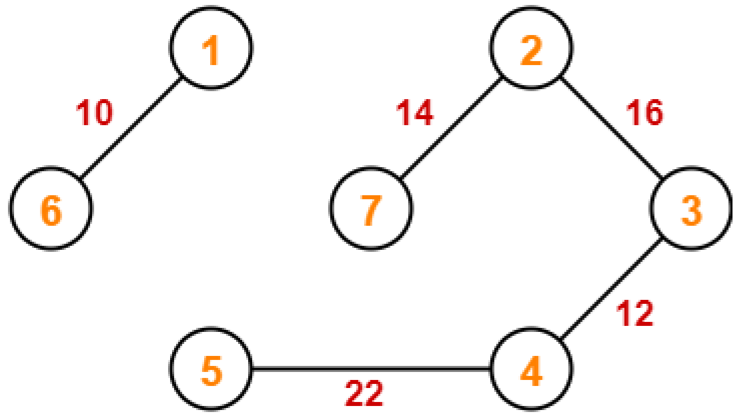
Step-04:



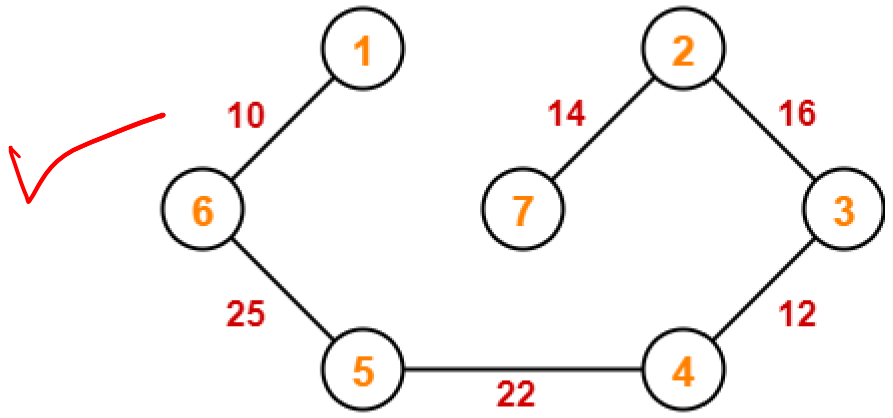
Step-05



Step-06:



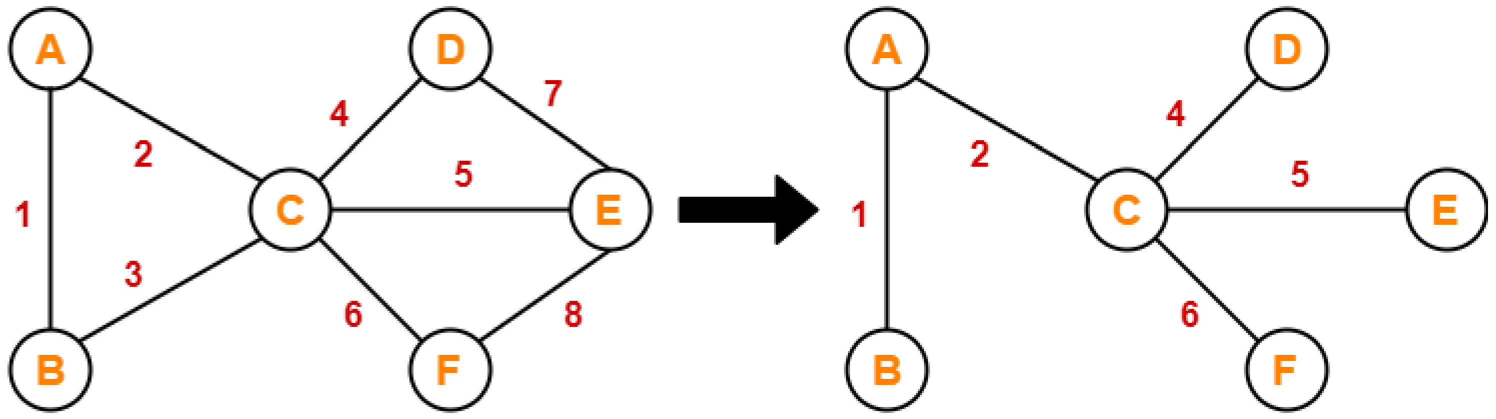
Step-07:



Weight of the MST
= Sum of all edge weights
= 10 + 25 + 22 + 12 + 16 + 14
= 99 units

Concept-01:

If all the edge weights are distinct, then both the algorithms are guaranteed to find the same MST.



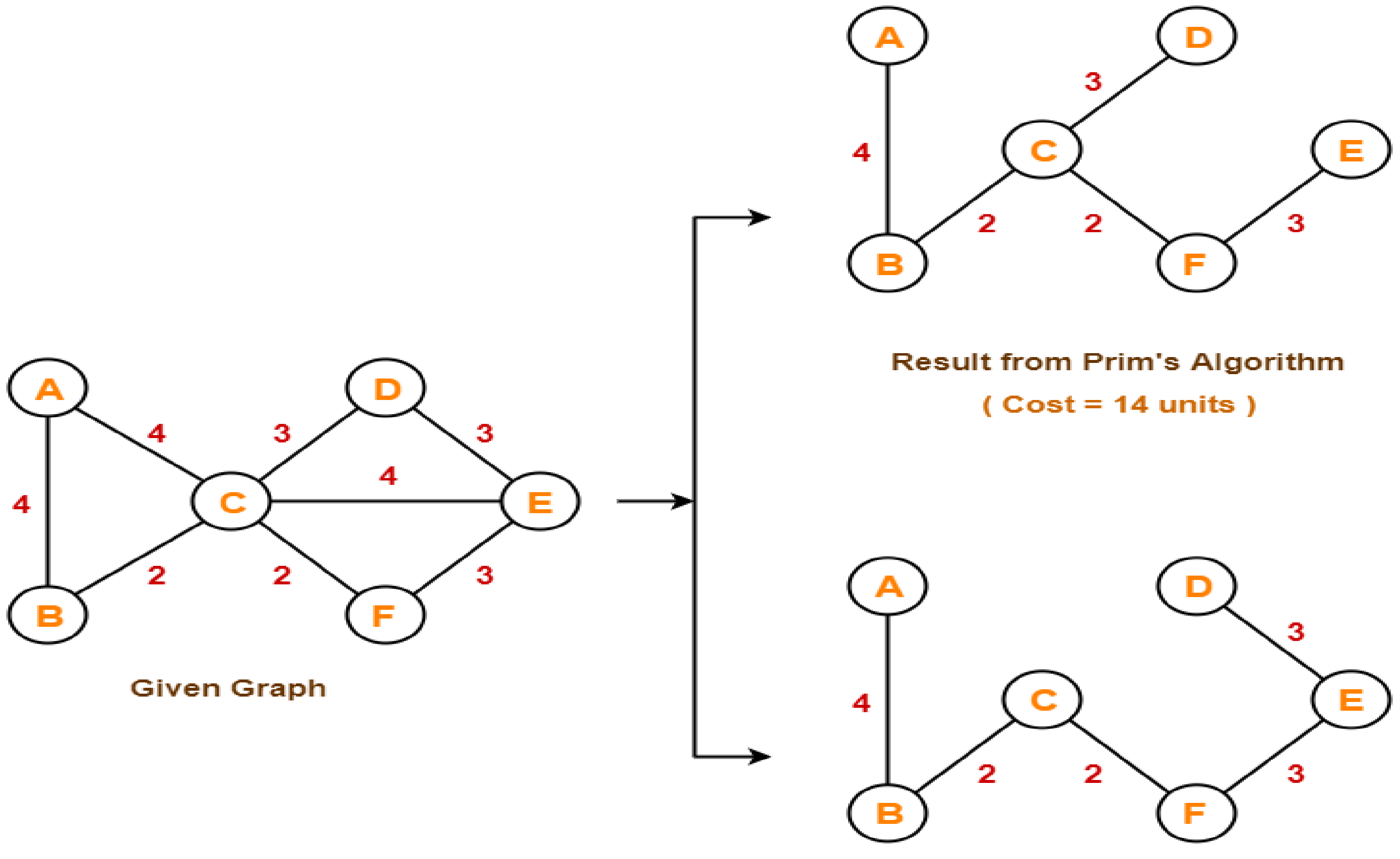
Given Graph

Minimum Spanning Tree (MST)

(Cost = 18 units)

Concept-02:

- If all the edge weights are not distinct, then both the algorithms may not always produce the same MST.
- However, cost of both the MST$_s$ would always be same in both the cases.



Given Graph

Result from Prim's Algorithm
(Cost = 14 units)

Result from Kruskal's Algorithm
(Cost = 14 units)

Shortest Path Problem

S D

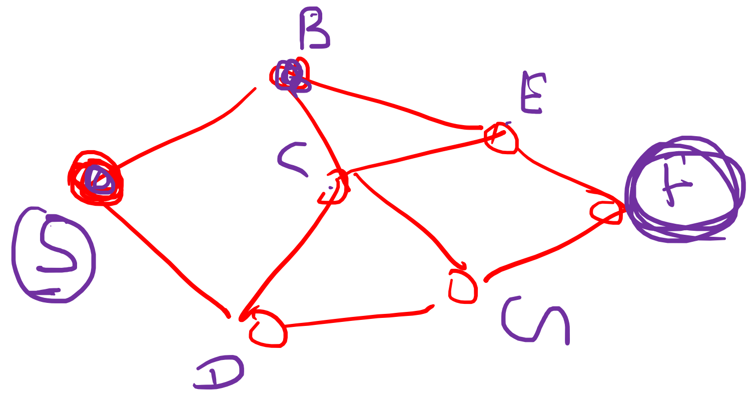
•Shortest path problem is a problem of finding the shortest path(s) between vertices of a given graph.
•Shortest path between two vertices is a path that has the least cost as compared to all other existing paths.

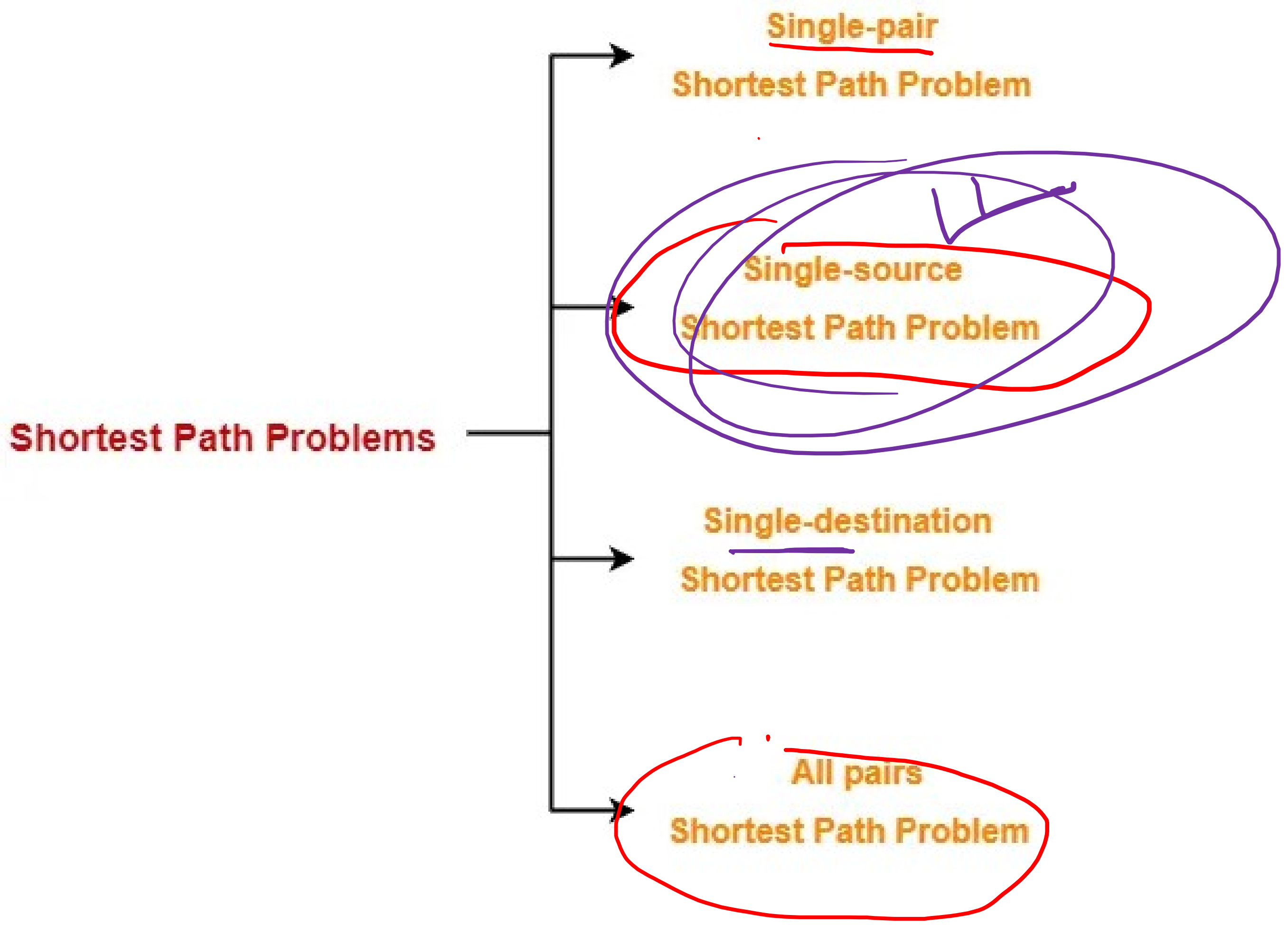
Applications-

Shortest path algorithms have a wide range o

1) Google Maps
2) Road Networks
3) Logistics Research

Types of Shortest Path Problem




Shortest Path Problems
Single-pair Shortest Path Problem
Single-source Shortest Path Problem
Single-destination Shortest Path Problem
All pairs Shortest Path Problem

Single-Pair Shortest Path Problem-

•It is a shortest path problem where the shortest path between a given pair of vertices is computed.
•A* Search Algorithm is a famous algorithm used for solving single-pair shortest path problem.

Single-Source Shortest Path Problem-

•It is a shortest path problem where the shortest path from a given source vertex to all other remaining vertices is computed.
•Dijkstra's Algorithm and Bellman Ford Algorithm are the famous algorithms used for solving single-source shortest path problem.

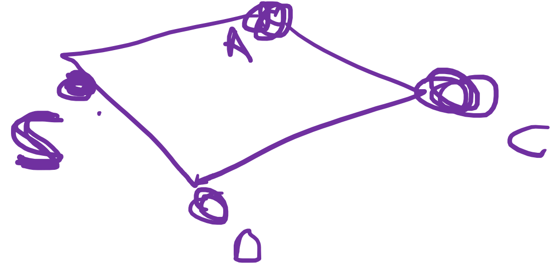
Single-Destination Shortest Path Problem-

•It is a shortest path problem where the shortest path from all the vertices to a single destination vertex is computed.
•By reversing the direction of each edge in the graph, this problem reduces to single-source shortest path problem.
•Dijkstra's Algorithm is a famous algorithm adapted for solving single-destination shortest path problem.

All Pairs Shortest Path Problem-

- It is a shortest path problem where the shortest path between every pair of vertices is computed.
- Floyd-Warshall Algorithm and Johnson’s Algorithm are the famous algorithms used for solving All pairs shortest path problem.

Dijkstra's Algorithm – Single Source Shortest Path Algorithm



•Dijkstra Algorithm is a very famous greedy algorithm.
•It is used for solving the single source shortest path problem.
•It computes the shortest path from one particular source node to all other remaining nodes of the graph.

Conditions-

Dijkstra algorithm works only for connected graphs.
•Dijkstra algorithm works only for those graphs that do not contain any negative weight edge.
•Dijkstra algorithm works for directed as well as undirected graphs.

Step-01:

In the first step. two sets are defined-

- One set contains all those vertices which have been included in the shortest path tree.
- In the beginning, this set is empty.
- Other set contains all those vertices which are still left to be included in the shortest path tree.
- In the beginning, this set contains all the vertices of the given graph.

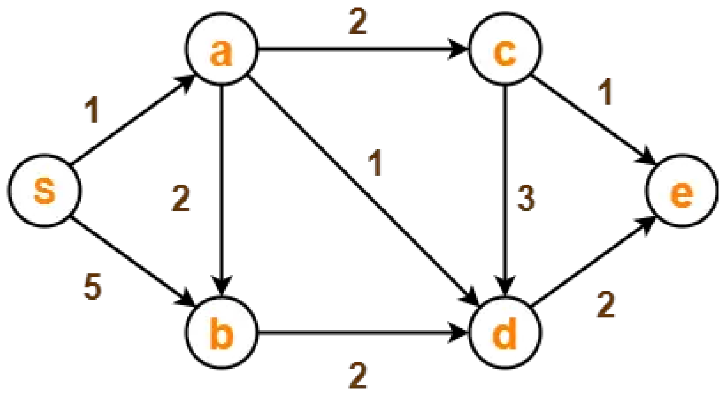
Step-02:

For each vertex of the given graph, two variables are defined as-

- Π[v] which denotes the predecessor of vertex ‘v’
- d[v] which denotes the shortest path estimate of vertex ‘v’ from the source vertex.

Initially, the value of these variables is set as-

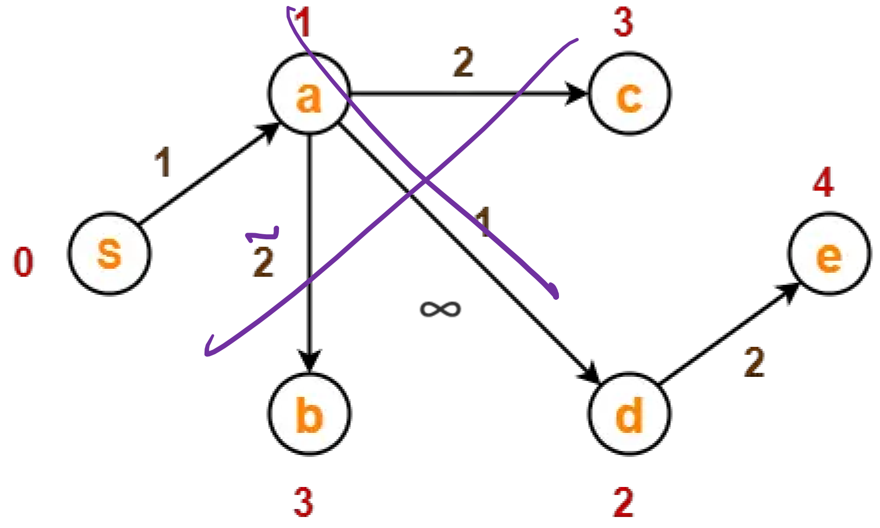
- The value of variable ‘Π’ for each vertex is set to NIL i.e. Π[v] = NIL
- The value of variable ‘d’ for source vertex is set to 0 i.e. d[S] = 0
- The value of variable ‘d’ for remaining vertices is set to ∞ i.e. d[v] = ∞



if (d(u) + c(u,v) < d(v))
{
d[v] = d[u] + c[u,v)
}

Step 1

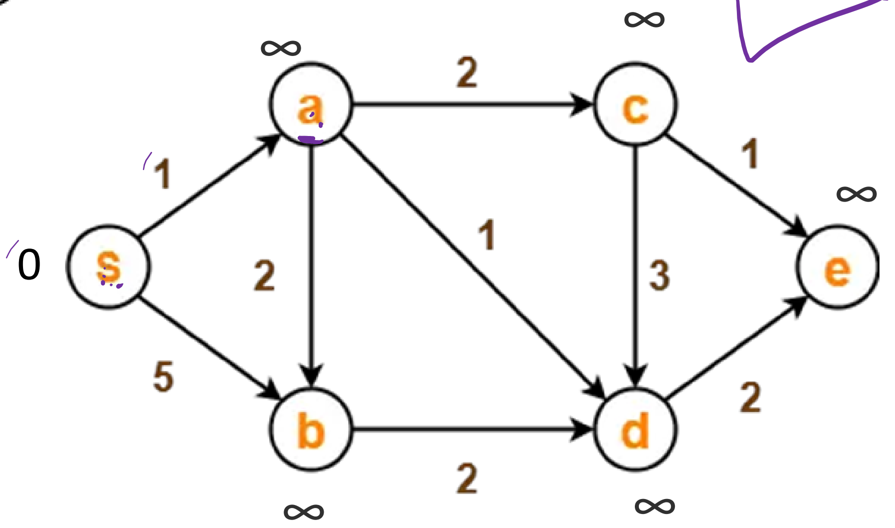
The following two sets are created-
- Unvisited set : {S , a , b , c , d , e}
- Visited set : { }



Shortest Path Tree

Step-02:

The two variables Π and d are created for each vertex and initialized as-
- Π[S] = Π[a] = Π[b] = Π[c] = Π[d] = Π[e] = NIL
- d[S] = 0
- d[a] = d[b] = d[c] = d[d] = d[e] = ∞

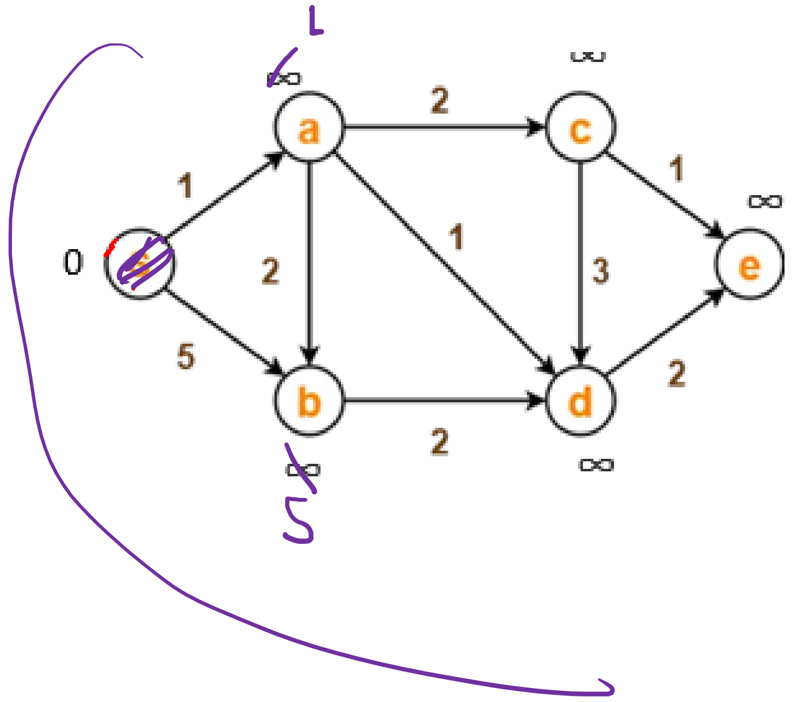


Step-03:

Vertex ‘S’ is chosen.

This is because shortest path estimate for vertex ‘S’ is least.

The outgoing edges of vertex ‘S’ are relaxed.



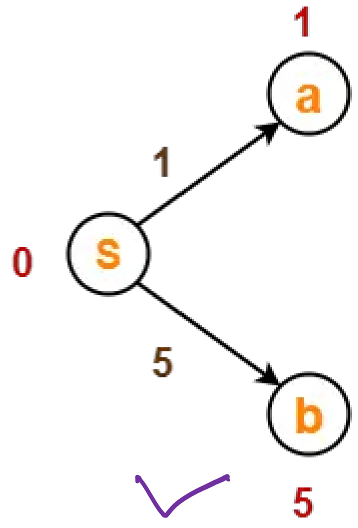
Now,

- $d[S] + 1 = 0 + 1 = 1 < \infty$

∴ $d[a] = 1$ and $\Pi[a] = S$

- $d[S] + 5 = 0 + 5 = 5 < \infty$

∴ $d[b] = 5$ and $\Pi[b] = S$



Now, the sets are updated as-

- Unvisited set : {a , b , c , d , e}
- Visited set : {S}

Step-04:

- Vertex ‘a’ is chosen.
- This is because shortest path estimate for vertex ‘a’ is least.
- The outgoing edges of vertex ‘a’ are relaxed.

$S \to a \to 1$

Now,

- $d[a] + 2 = 1 + 2 = 3 < \infty$

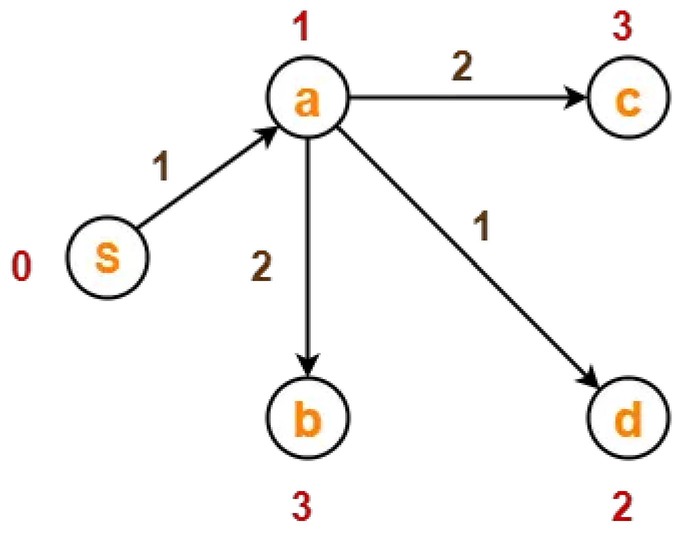
$\therefore$ $d[c] = 3$ and $\Pi[c] = a$

- $d[a] + 1 = 1 + 1 = 2 < \infty$

$\therefore$ $d[d] = 2$ and $\Pi[d] = a$

- $d[b] + 2 = 1 + 2 = 3 < 5$

$\therefore$ $d[b] = 3$ and $\Pi[b] = a$

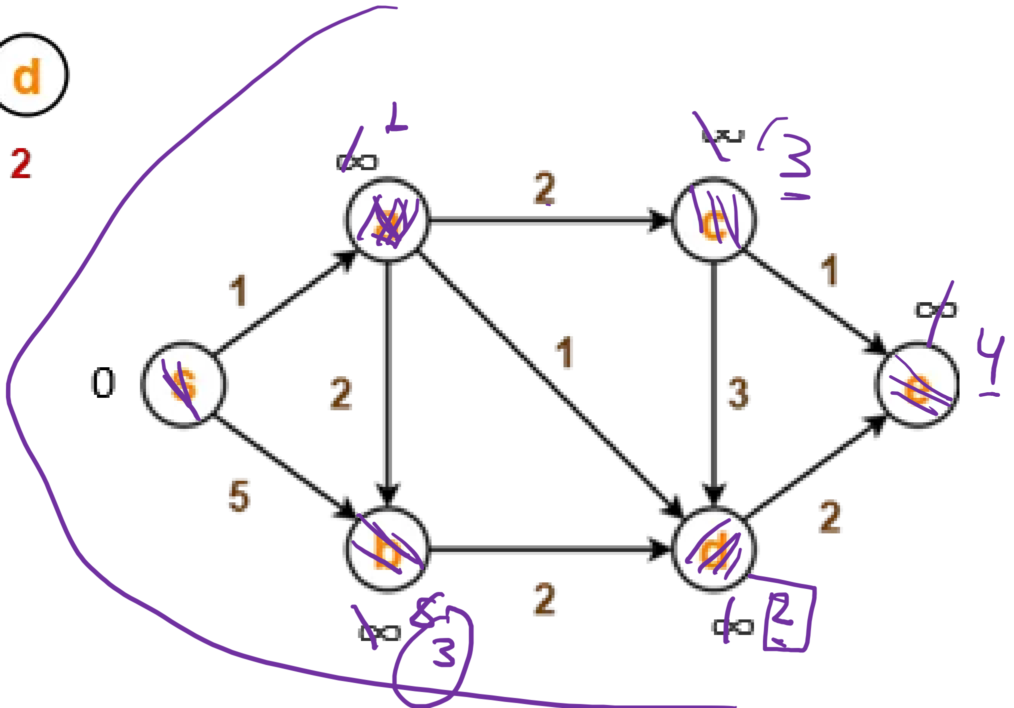


Now, the sets are updated as-

Unvisited set : {b , c , d , e}

Visited set : {S , a}

Visited {S, a, e}



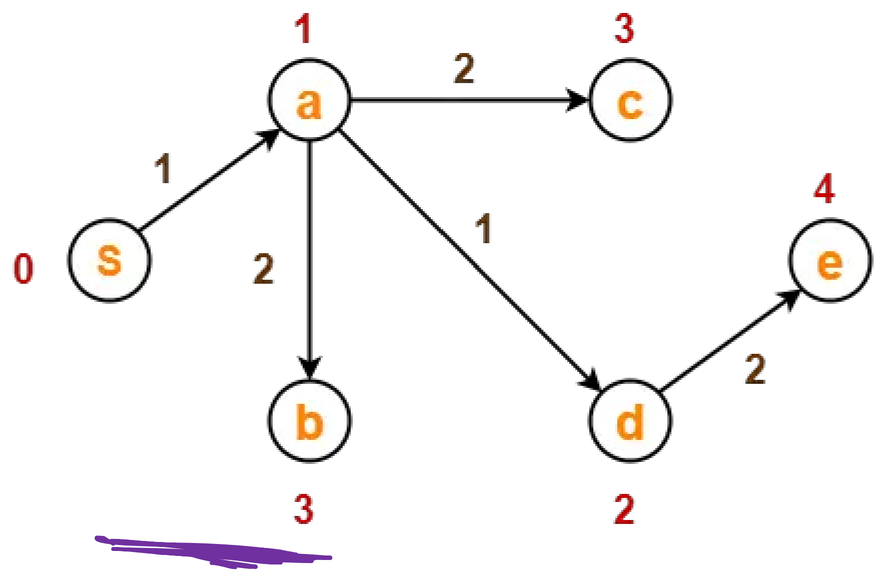
Step-05:

- Vertex ‘d’ is chosen.
- This is because shortest path estimate for vertex ‘d’ is least.
- The outgoing edges of vertex ‘d’ are relaxed.

Now,

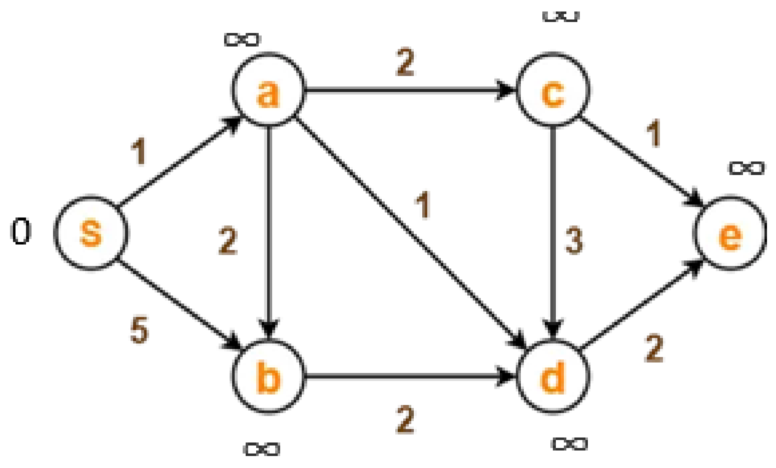
- d[d] + 2 = 2 + 2 = 4 < ∞

∴ d[e] = 4 and Π[e] = d



Now, the sets are updated as-

- Unvisited set : {b , c , e}
- Visited set : {S , a , d}



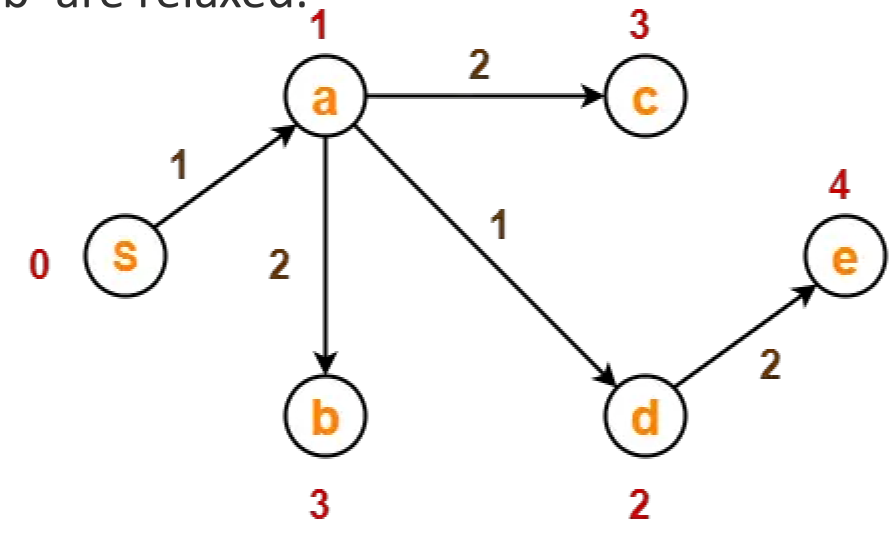
Step-06:

- Vertex ‘b’ is chosen.
- This is because shortest path estimate for vertex ‘b’ is least.
- Vertex ‘c’ may also be chosen since for both the vertices, shortest path estimate is least.
- The outgoing edges of vertex ‘b’ are relaxed.

Now,

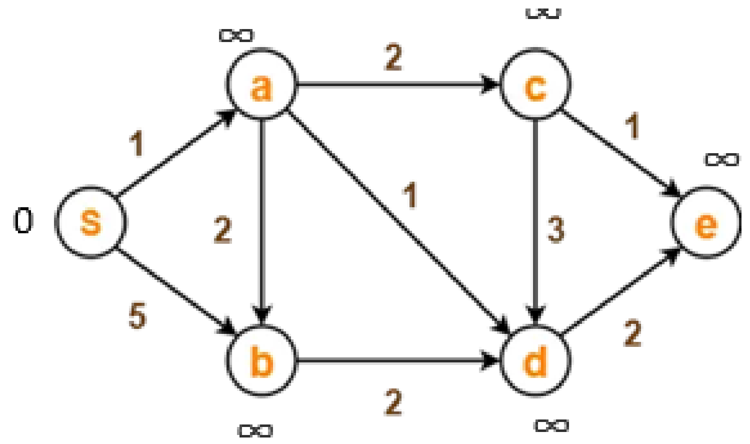
- d[b] + 2 = 3 + 2 = 5 > 2

∴ No change



Now, the sets are updated as-

- Unvisited set : {c , e}
- Visited set : {S , a , d , b}



Step-07:

- Vertex ‘c’ is chosen.
- This is because shortest path estimate for vertex ‘c’ is least.
- The outgoing edges of vertex ‘c’ are relaxed.

Now,

- d[c] + 1 = 3 + 1 = 4 = 4

∴ No change

Now, the sets are updated as-

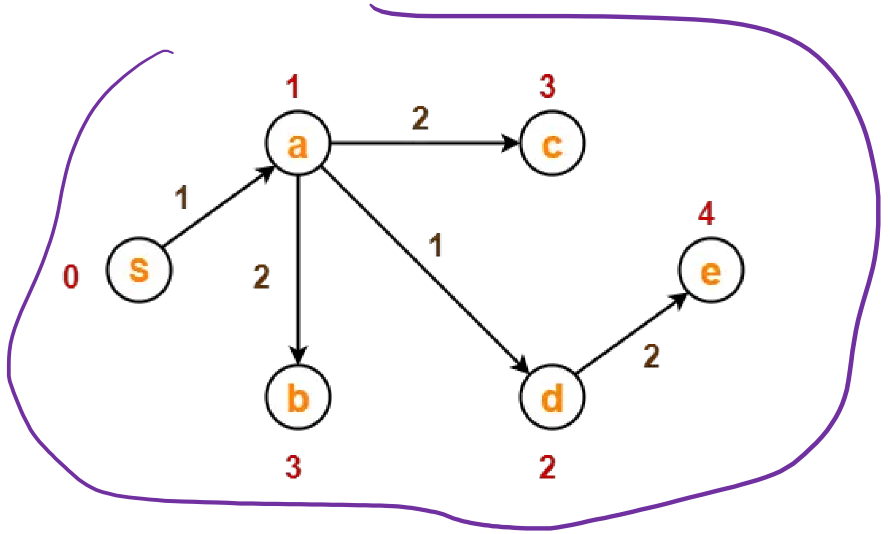
Unvisited set : {e}

Visited set : {S , a , d , b , c}

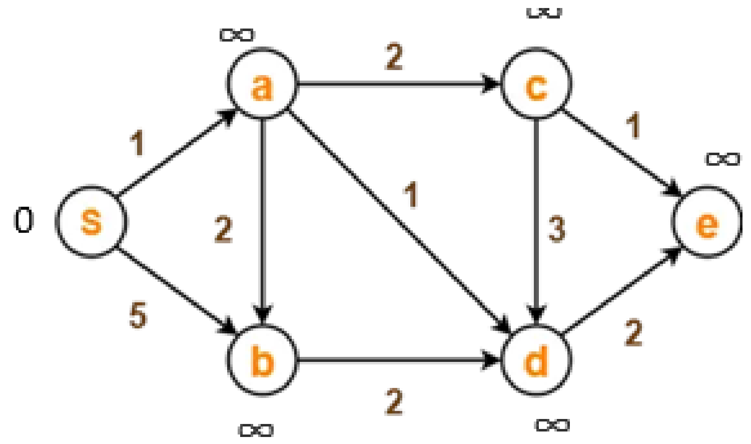
Step-08:

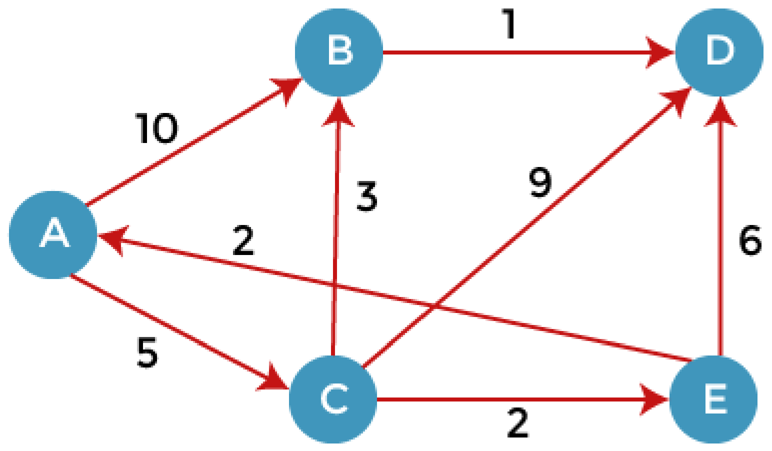
•Vertex ‘e’ is chosen.
•This is because shortest path estimate for vertex ‘e’ is least.
•The outgoing edges of vertex ‘e’ are relaxed.
•There are no outgoing edges for vertex ‘e’.
•So, our shortest path tree remains the same as in Step-05.

Now, the sets are updated as-
•Unvisited set : { }
•Visited set : {S , a , d , b , c , e}



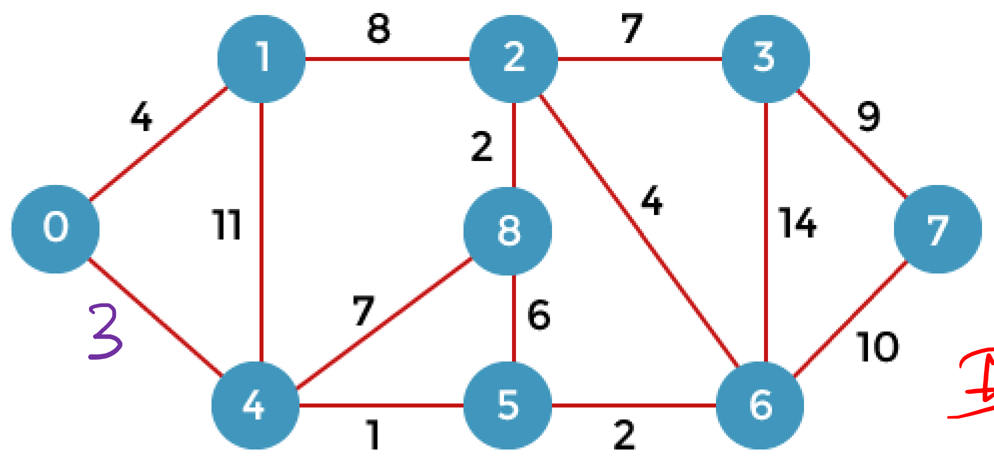
Shortest Path Tree

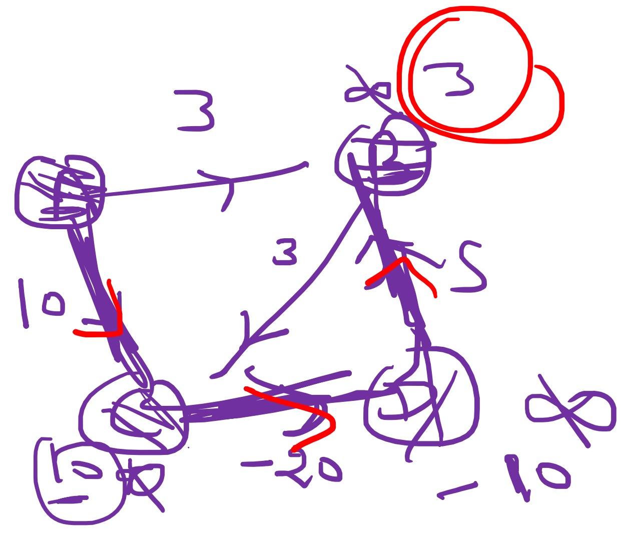




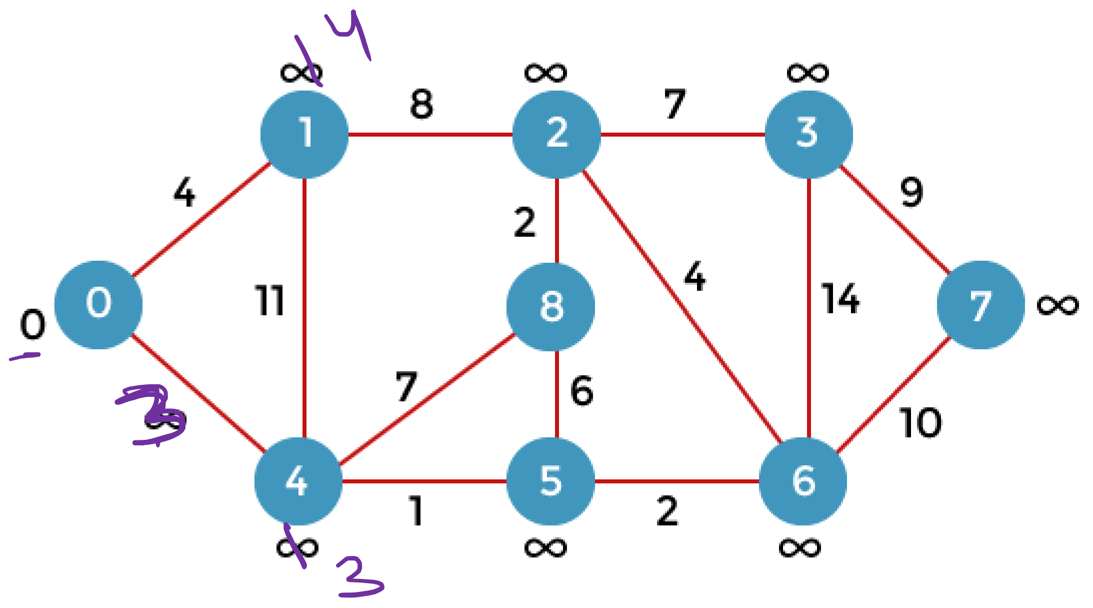
	A	B	C	D	E
A	0	∞	∞	∞	∞

	A	B	C	D	E
A	0	∞	∞	∞	∞
		10	5	∞	∞





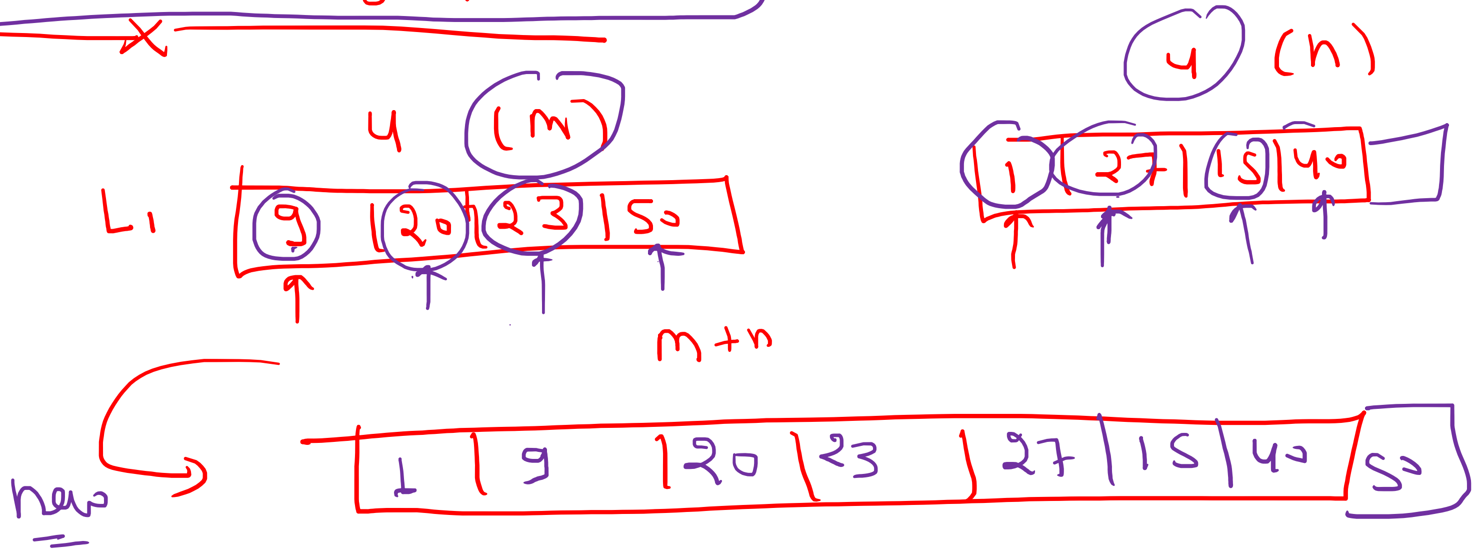
I 484, 487, 504,
~~5X~~, 515, 518, 520,
524, L07, ~~L12~~, L29,
L40,



Bellman ford

D ~~181~~ 185, 188, 192, 197,
198, ~~188~~, 203, 208, ~~215~~ 216
218, 221, 222, 232, 233, 234, 235
L21, L50,

Optimal merge pattern



Optimal File Merge Patterns

When two or more sorted files are to be merged altogether to form a single file, the minimum computations are done to reach this file are known as **Optimal Merge Pattern.**

If the number of sorted files are given, there are many ways to merge them into a single sorted file. This merge can be performed pair wise. Hence, this type of merging is called as **2-way merge patterns**

We use the Greedy strategy by merging the two smallest size files among all the files present.

If more than 2 files need to be merged then it can be done in pairs. For example, if need to merge 4 files A, B, C, D. First Merge A with B to get X1, merge X1 with C to get X2, merge X2 with D to get X3 as the output file.

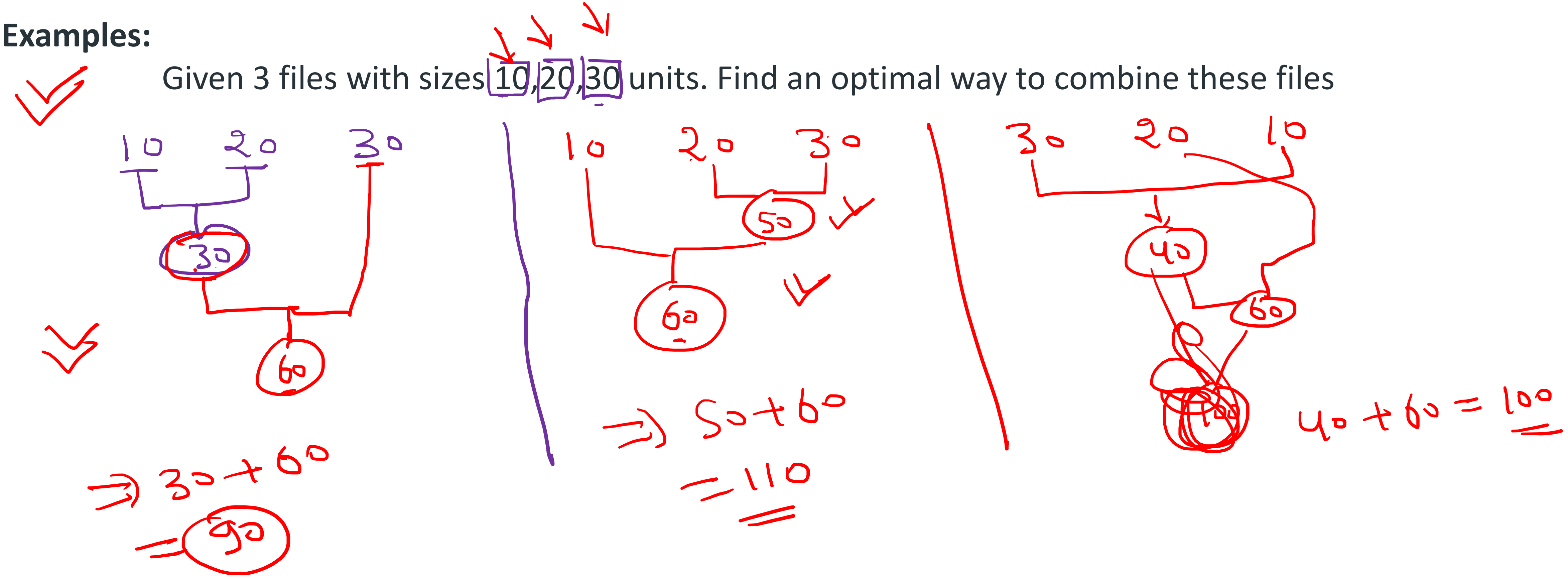
Two files of sizes m and n, the total computation time will be m+n.

The formula of external merging cost is: $$\sum_{i=1}^{n} f(i)d(i)$$

Where, `f (i)` represents the number of records in each file and `d (i)` represents the depth.

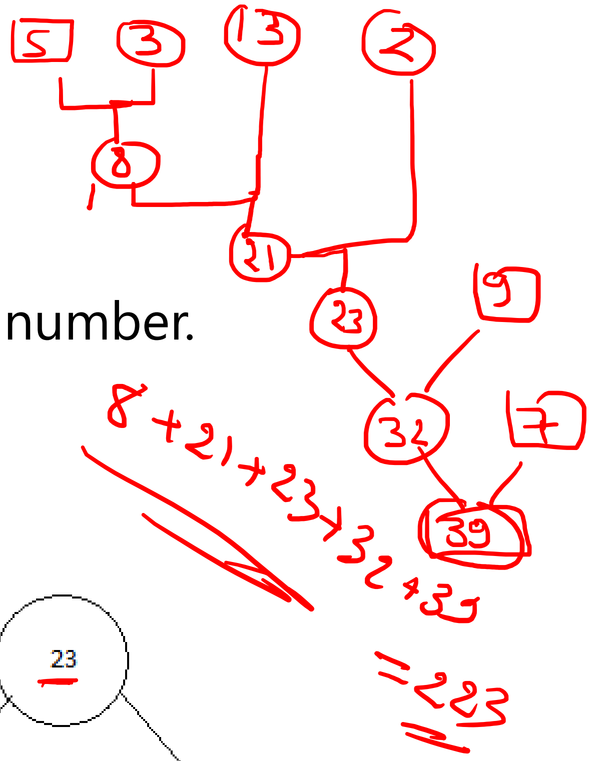
Examples:

Given 3 files with sizes $\boxed{10}$,$\boxed{20}$,$\boxed{30}$ units. Find an optimal way to combine these files

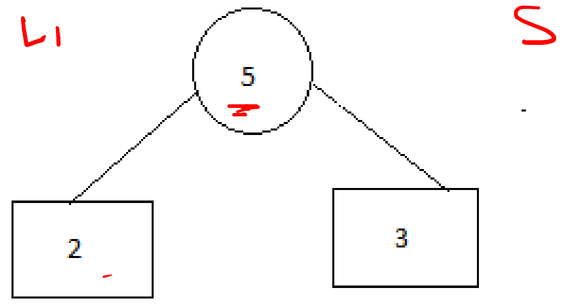


Example: Given a set of unsorted files: 5, 3, 2, 7, 9, 13

arrange these elements in ascending order: 2, 3, 5, 7, 9, 13
After this, pick two smallest numbers and repeat this until we left with only one number.

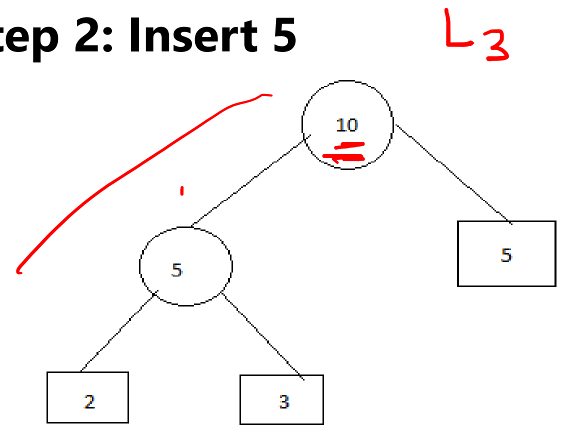


Now follow following steps:
Step 1: Insert 2, 3

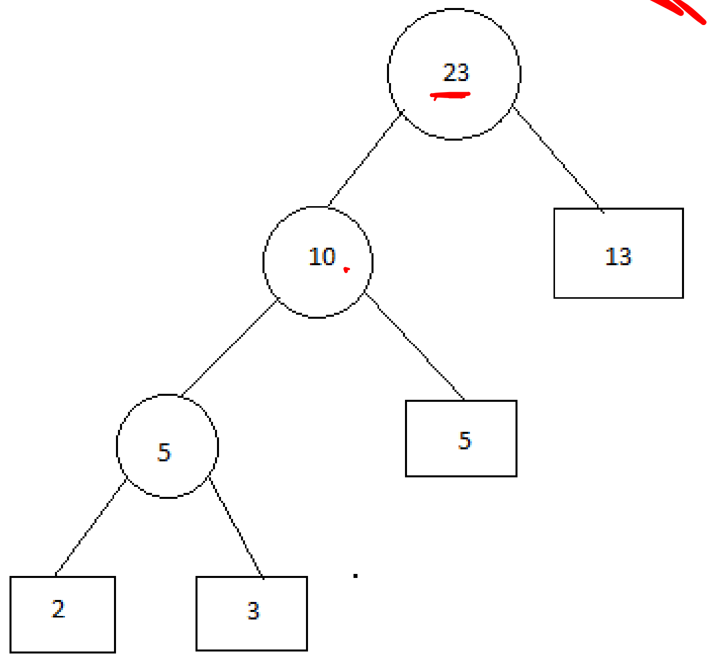


L2
Step 3
16
7
9

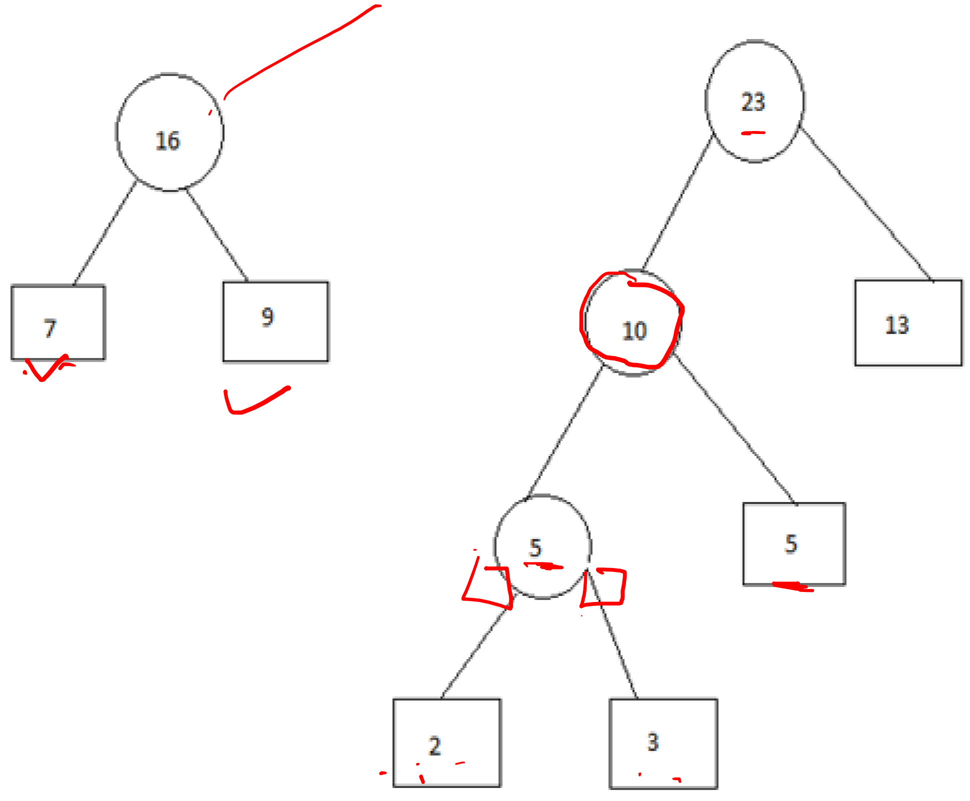
Step 2: Insert 5



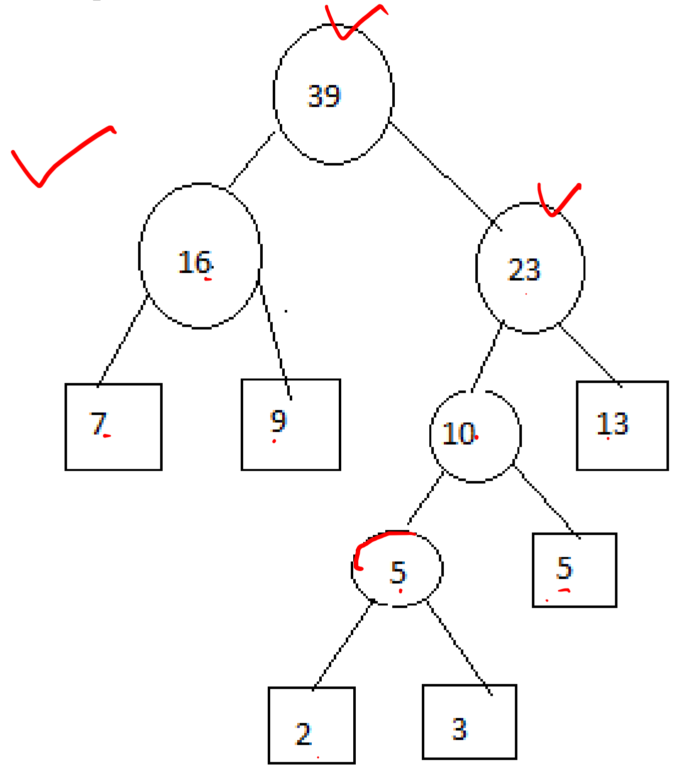
Step 3: Insert 13



Step 4: Insert 7 and 9



Step 5:



So, The merging cost = 5 + 10 + 16 + 23 + 39 = $\boxed{93}$

I 484, 485, 497, 504, ~~512~~, 520, 524, L07, L15, L29, L40

D 183, 184, 186, ~~[illegible]~~ 188, 193, 194, 196, 198, 208, 212, 222, 226, 227, 228, ~~23~~ L21, L50, 5 10 20 30 30

```
Algorithm: TREE (n)
for i := 1 to n – 1 do
    declare new node
    node.leftchild := least (list)
    node.rightchild := least (list)
    node.weight) := ((node.leftchild).weight) + ((node.rightchild).weight)
    insert (list, node);
return least (list);
```

```
Struct node
{
  *
  treenode * leftchild,
  treenode * right child
  int weight;
};
```