

Module - III

Graph Traversal Algorithm

Traversing the graph means examining all the nodes and vertices of the graph. There are two standard methods by which we can traverse the graph.

1. Breadth first search

2. Depth first search

Breadth first search: It is a graph traversal algorithm that starts traversing the graph from root node and explores all the neighbour nodes.

- Then it selects the nearest node and explores all the unexplored nodes.
- The algorithm follows the same process for each of the nearest nodes until it finds the goal.

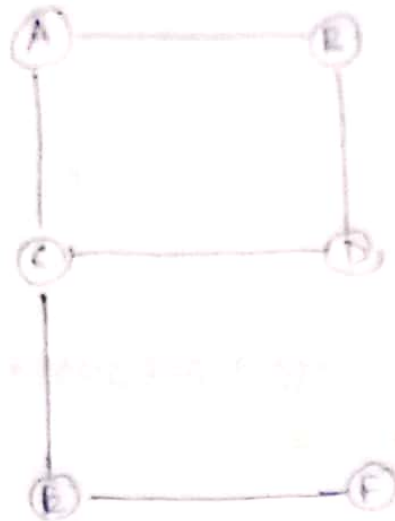
Note: (i) It is a graph searching algorithm.
(ii) visit the vertices in breadth dirn.

(iii) Find the reachable vertices from a starting vertex.

(iv) Find the minimum path between two nodes.

(v) It uses queue.

Ex



Adjacency list:

A → B, C

B → A, D

C → A, D, E

D → B, C

E → C, F

F → E

queue [A]

queue [~~A~~, B, C]

queue [~~B~~, C, D]

queue [~~C~~, D, E]

queue [~~D~~, E]

queue [~~E~~, F]

queue [~~F~~]

BFS: ABCDEF

Let starting vertex is A and A given with a value

- Then delete A and insert adjacent of A i.e B, C and put value 1 to B, C.
- Then delete B and insert adjacent of B i.e D and put value - 2 to D.
- Inserted vertex is not inserted again.
- Then delete C and insert adjacent of C (i.e E) and put value 2 to E.
- Now delete E and insert adjacent of E i.e F and put value 3 to F.

→ value of Adjacent vertex = value of deleted vertex + 1

→ we use 3 color for vertex

White - vertex is not processed

Gray - vertex is to be processed

Black - vertex has finish processing.

Steps: ① Initially all node are given with white color.

② Insert starting vertex A into the queue and change its color to queue (gray) and distance of $[A] = 0$.

③ Delete first vertex A into the queue and change its colour to black.

④ present the adjacent vertex of A having white color change their color to gray
The value of adjacent vertices $\geq$ value of deleted vertex $+1$

⑤ Repeat step ③ and ④ while the queue is empty.

Algorithm

// G is the graph

// A is the starting vertex for each vertex u of the graph.

$color[u] \leftarrow white$

$color[u] \leftarrow \infty$

$parent[u] \leftarrow NIL$

$deleted[A] = 0$

while (queue $\neq$ empty)

{

$u = deleted_queue()$

for each ~~vertex~~ adjacent vertex v of u

{

if ($color[v] \leftarrow white$)

$color[v] \leftarrow gray$

$distance[v] \leftarrow distance[u] + 1$

$parent[v] \leftarrow u$

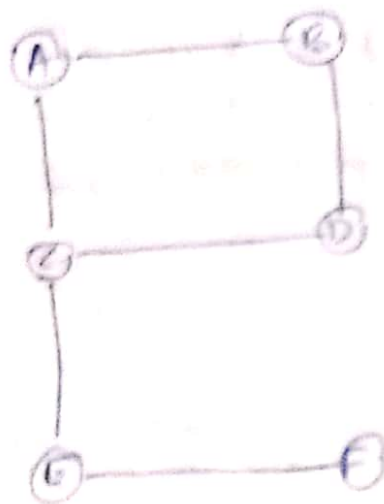
}

$color[u] \leftarrow black$

}

DEPTH FIRST SEARCH

- ① It is also a graph searching algorithm.
- ② Visit the vertex in depth direction.
- ③ Finding the reachable vertex from the starting.
- ④ Find minimum path between two nodes.
- ⑤ It uses stack.



ADJACENT LIST

A → B, C

B → A, D

C → A, D, E

D → B, C

E → C, F

F → E



DFS = A C B F D B

Let starting vertex is A visit time of A is 0

→ Then delete A and insert adjacent of A (i.e. B, C)

→ Then delete top element C and insert adjacent of C (i.e. A, D, E)

- But A is already inserted. So we insert only D, E.
- Then delete top element E and insert adjacent of E (i.e. C, F)
- C is already inserted so we insert only F.
- Delete F (adjacent of F is E) which is already inserted.
- Now the top element is D, so delete D adjacent of D is B, C which is already inserted.
- Delete B.

Steps:

- ① Initially all nodes are given white color.
- ② Insert the starting vertex A into queue and change its color to queue.
- ③ Delete first vertex 'N' of queue change its color to black.
- ④ Insert the adjacent vertex of N having white color and change their color to gray.

$$\text{distance}[\text{adjacent of N}] = \text{distance}[N] + 1$$
- ⑤ Repeat step (3) and (4) until queue is empty.

Algorithm

Algorithm DFS(G, A)

{

where G is a graph.

$A \rightarrow$ starting vertex for each vertex u of the graph.

color[u] $\leftarrow$ white

parent[u] $\leftarrow$ NIL

time $\leftarrow$ 0

for each vertex u of the graph

if (color[u] $\leftarrow$ white)

DFS-visit(u)

{

color[u] $\leftarrow$ gray

time $\leftarrow$ time + 1

~~visit~~ visit time[u] $\leftarrow$ true

for each adjacent vertex v of u

{

if color[v] $\leftarrow$ white

DFS-visit(v)

parent[v] $\leftarrow$ u .

{

color[v] $\leftarrow$ black.

time $\leftarrow$ time + 1

}

Q: Difference betn BFS and DFS

BFS	DFS
→ visit the vertex in breadth dirn	→ visit the vertex in depth direction.
→ It use a queue	→ It use a stack.
→ find the reachable vertex from the starting vertex	→ find the reachable vertex from the starting vertex
→ It is a graph searching algorithm	→ It is also a graph searching algorithm
→ Used for finding minimum path betn two nodes	→ Used for finding minimum path betn two nodes.

Single source shortest path

It find shortest path from one vertex to all other vertices

→ there are two Algorithms

i) Dijkstra Algorithm

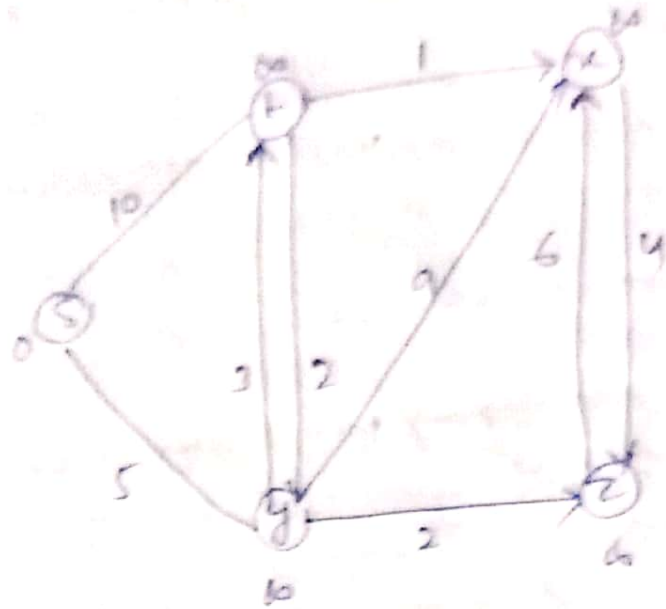
ii) Bellman Ford Algorithm

Dijkstra Algorithm

① Source vertex or starting vertex is given

② It determine the shortest path from the source vertex to any other vertex

Ex:

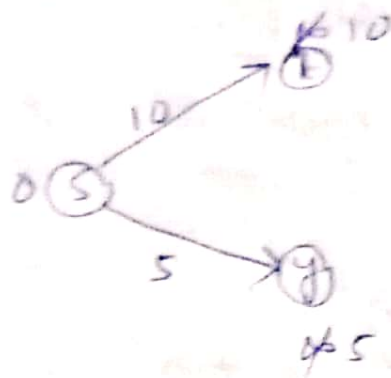


Initially starting vertex is S

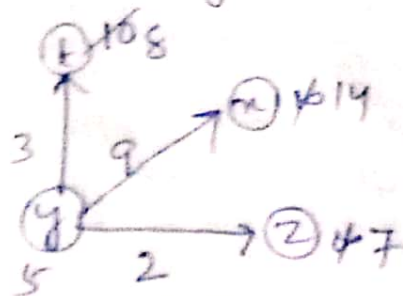
distance from starting vertex is $d[S] = 0$

distance of all other vertices $d[V] = \infty$

Step-1: process S



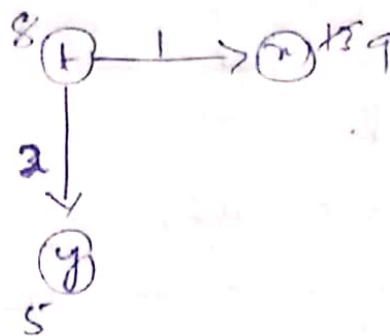
Step-2: process y [because minimum value at y]



Step-3: process z (minimum value at z)



Step-4: consider value at x
minimum value is at t.
process t.

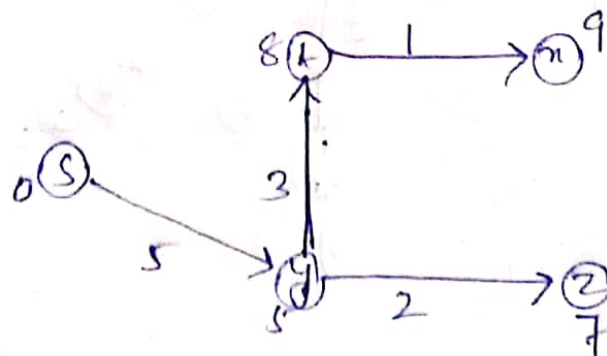


Step-5: Remaining Remaining vertex is x
process x.



Now all vertex are processed. Finally
draw path by following value at vertices.

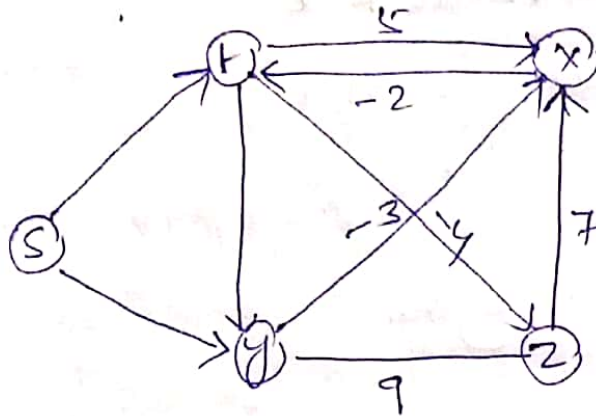
Output:



Bellman - Ford Algorithm

- This algorithm also solves the single source shortest path problem.
- The weight of an edge may be negative.

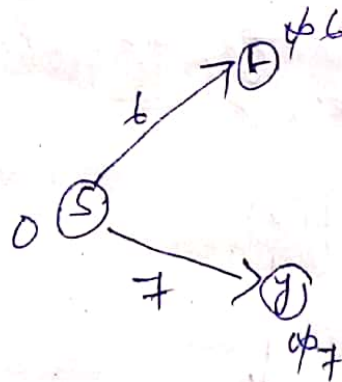
Ex:



Initially starting vertex $d[s] = 0$.

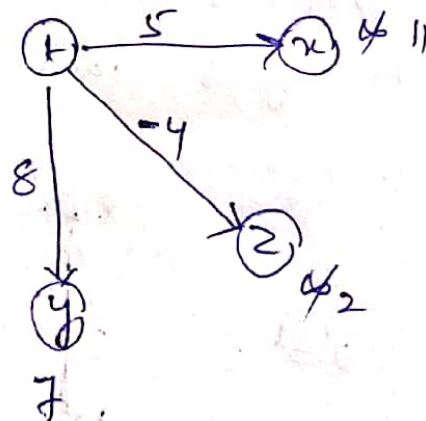
Distance of all other vertices $d[v] = \infty$.

Step-1: process s

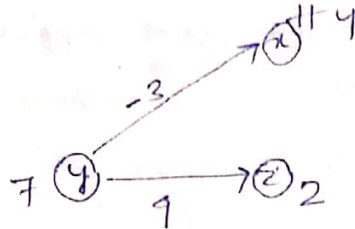


Step-2: process t y

process t

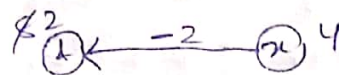


process y

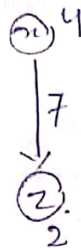


Step-3: Process x, z

process x

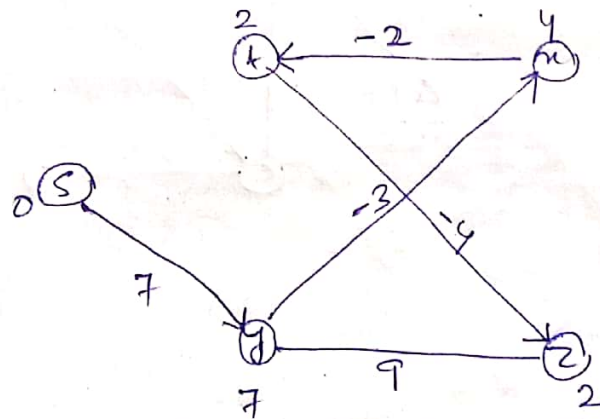


process z



Now all vertices are processed

O/p:



Minimum Spanning tree:

Spanning tree is a subset of graph in which all the vertices are present but all the edges may not present

→ Spanning tree with minimum distance is called minimum spanning tree

- to find min. spanning tree there are two algorithm

1) Kruskal

2) Prim's

- Both the algorithm are applied on a weighted graph. (A weighted graph is a graph where each edge has a weight)

Kruskal Algorithm

Select edge one after another by increasing order of weight

$(a, b) \rightarrow 2$

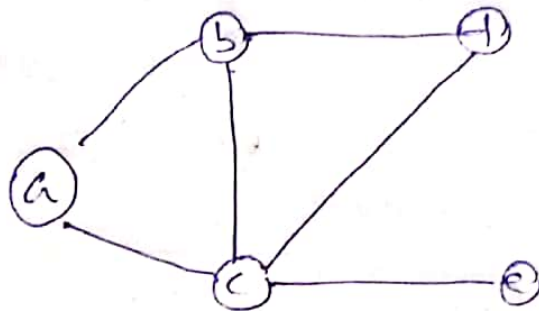
$(b, c) \rightarrow 4$

$(a, c) \rightarrow 5$

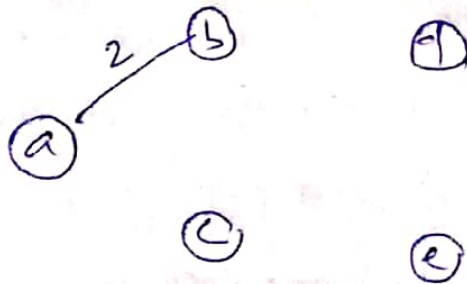
$(c, d) \rightarrow 7$

$(b, d) \rightarrow 9$

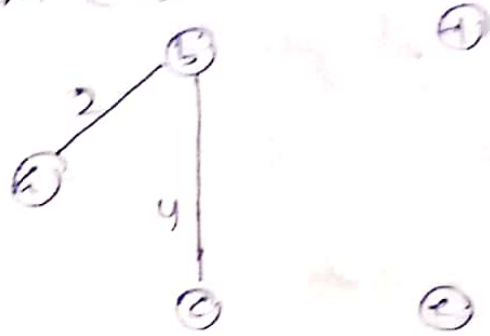
$(c, e) \rightarrow 12$



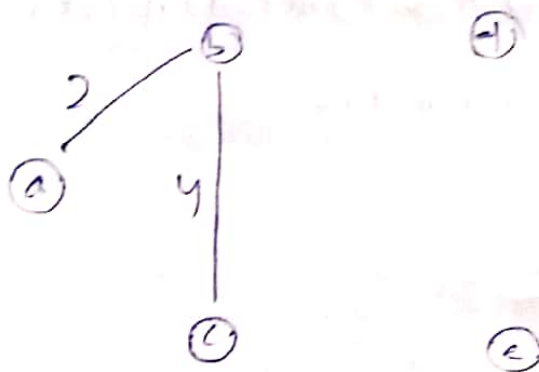
Step-1: The min weighted is 2. so we have to cover path (a, b)



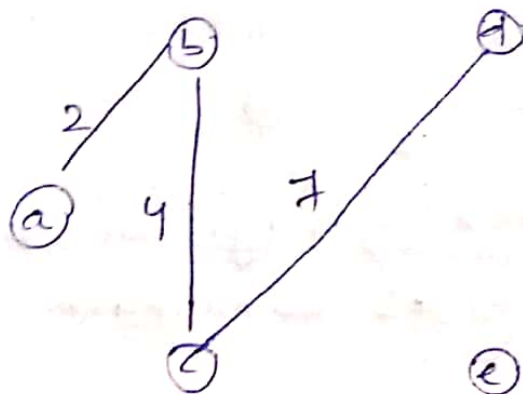
Step-2: Next minimum is 4 i.e. (b,c)
so path (b,c) is covered



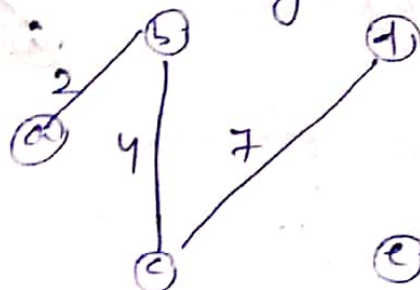
Step-3: select (c,d) but we cannot select (c,d)
becoz it will create a cycle.



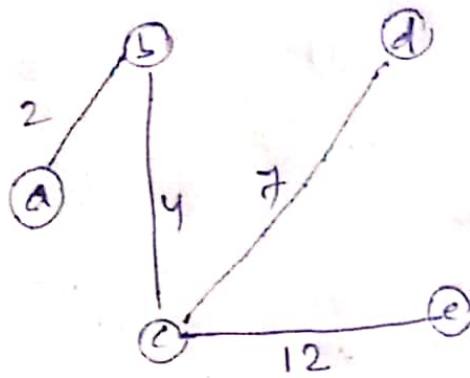
Step-4: select (c,d), so cover path from
(c,d) = 7



Step-5: select (b,d) but we can't take (b,d) becoz
it will create a cycle



Step-6: Select (c,e) so minimum distance = 12



Total edge = 4

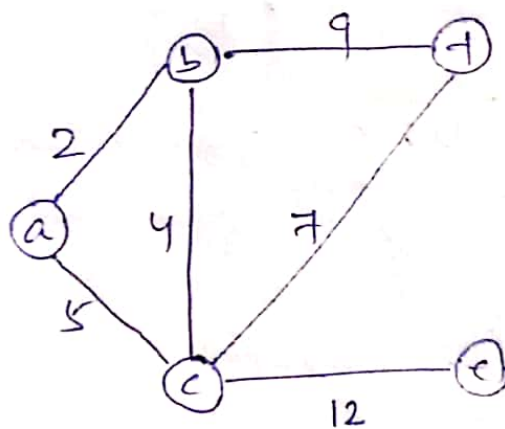
total step are = 6

Total cost = $(a,b) + (b,c) + (c,d) + (c,e)$

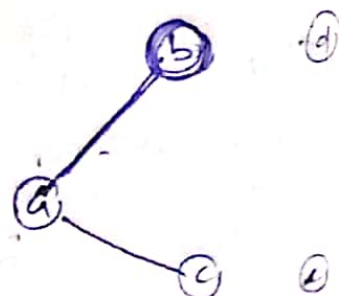
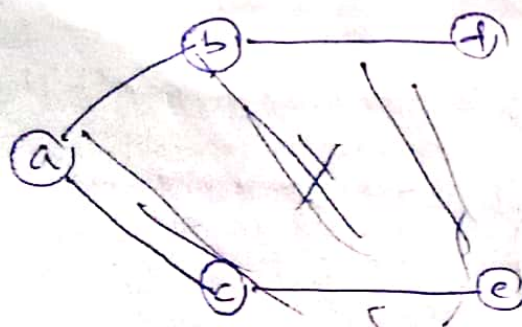
$$= 2 + 4 + 7 + 12$$

$$= 25$$

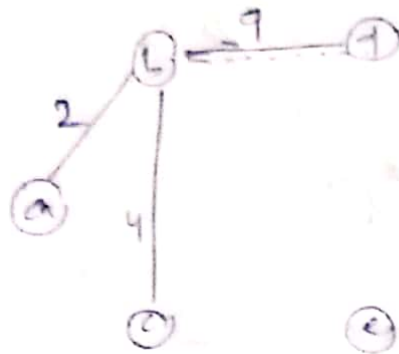
PRIM'S Algorithm



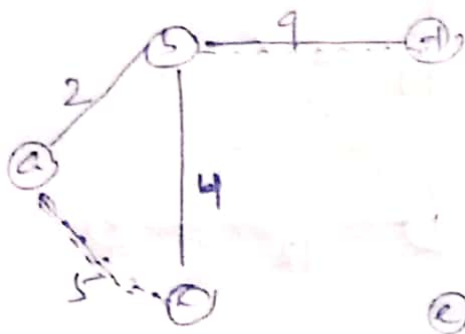
Step-1: From vertex "a" edge are (a,b) and (a,c)
out of this (a,b) is minimum



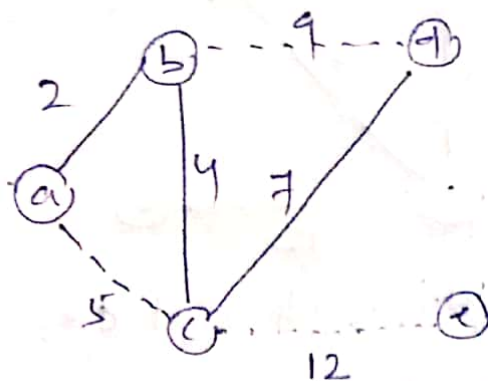
Step-2: From vertex b new edge are (b, d) and (b, c) . minimum edge from the above edge (b, c)



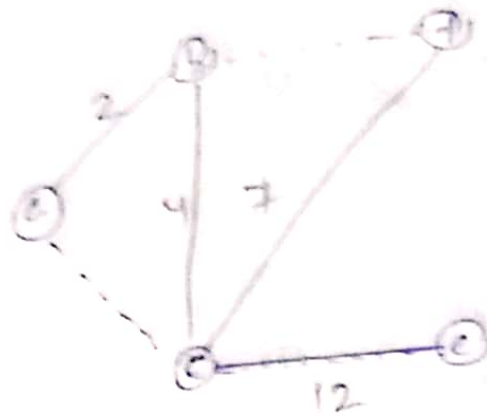
Step-3: From vertex c new edge are (c, d) , (c, e) and (c, a) . Out of this three minimum is (c, a) but (c, a) cannot taken, becoz it create a cycle.



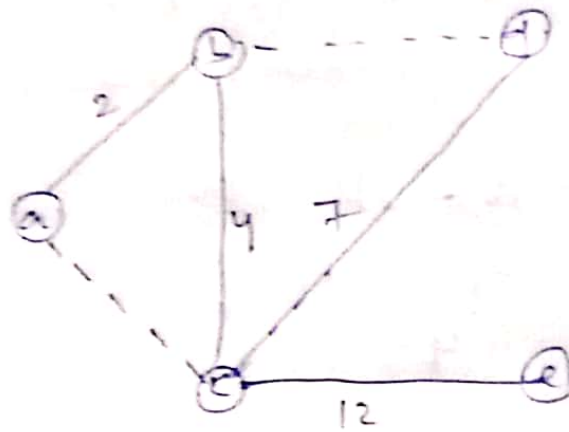
Step-4: minimum edge from the remaining edge is (c, d) so we reach at d.



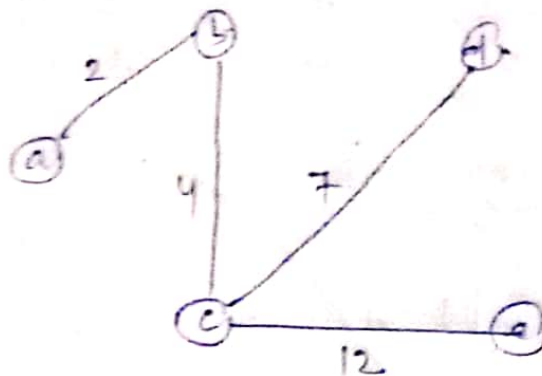
Step-5 minimum edge (b,d) out of remaining edge (b,d) and (c,e) but (b,d) is not taken because it will create a cycle.



Step-6 (c,e) is taken because it is the last.



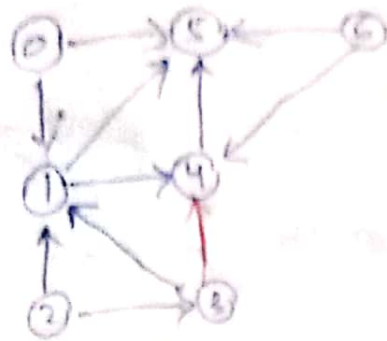
TOTAL COST:



$$\begin{aligned}
 \text{Total cost} &= (a,b) + (b,c) + (c,d) + (c,e) \\
 &= 2 + 4 + 7 + 12 \\
 &= 25
 \end{aligned}$$

Topological Sorting

Let us have a graph and apply topological sorting



Indegree of vertices

$$\text{In}(0) = 0 \quad \text{In}(4) = 3$$

$$\text{In}(1) = 3 \quad \text{In}(5) = 4$$

$$\text{In}(2) = 0 \quad \text{In}(6) = 0$$

$$\text{In}(3) = 1$$

The vertices with zero indegree are 0, 2 and 6.

They can be inserted in the queue in any order

Queue: 0, 2 and 6.

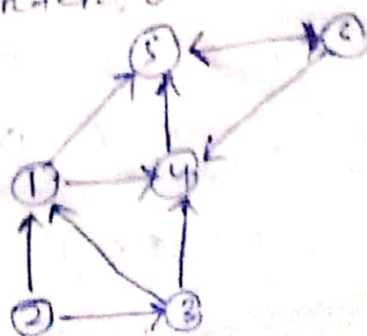
Step-1: Delete the vertex 0 and edge going from vertex 0.

Queue: 2 and 6 top-order: 0

Update indegree

$$\text{In}(1) = 2 \quad \text{In}(4) = 3$$

$$\text{In}(3) = 1 \quad \text{In}(5) = 3$$



Step-2: Delete the vertex 2 and edge going from vertex 2

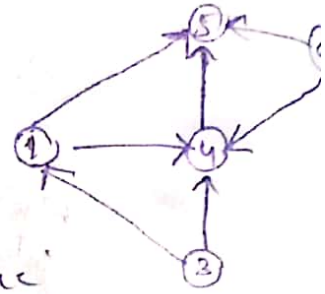
Queue: 6 top-order: 0, 2
update in-degree of vertices

$In(1) = 1$ $In(3) = 0$

$In(4) = 3$ $In(5) = 2$

Insert vertex 3 into the Queue

Queue: 6, 3



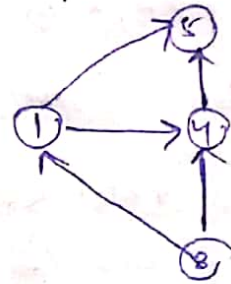
Step-3: Delete the vertex 6 and edge going from vertex 6.

Queue: 3 top-order: 0, 2, 6.

update in-degree of vertices

$In(1) = 1$ $In(5) = 2$

$In(4) = 2$



Step-4: Delete vertex 3 and edge going from vertex 3

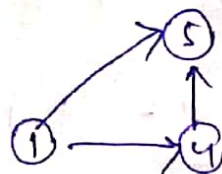
Queue: Empty top-order: 0, 2, 6, 3

update in-degree of vertices

$In(1) = 0$

$In(4) = 1$

$In(5) = 2$



Step-5: Delete the vertex 1 and edge going from vertex 1

Queue: Empty top-order: 0, 2, 6, 3, 1

Update in-degree of vertices

$$\text{In}(4) = 0$$

$$\text{In}(5) = 1$$



Insert vertex 4 into the queue

Queue: 4.

Step-6: Delete the vertex 4 and edge going from vertex 4.

Queue: Empty top-order: 0, 2, 6, 3, 1, 4.

Update in-degree of vertices $\text{In}(5) = 0$.

Insert vertex 5 into the queue.

Step-7: Delete the vertex 5 and edge going from vertex 5.

Queue: Empty top-order: 0, 2, 6, 3, 1, 4, 5.