Data Structures & Algorithms Design- SS ZG519
Lecture - 15

BITS Pilani
Pilani Campus

Dr. Padma Murali

Lecture 15 Topics

- Graph Traversals- Depth First Search & Breadth First Search
- Minimum Spanning tree problem

innovate achieve lead

BITS Pilani
Pilani Campus

Graph Searching Algorithms

Graph Searching Algorithms

- Searching a graph means systematically following the edges of the graph so as to visit the vertices of the graph.

- A graph-searching algorithm can discover much about the structure of a graph.

- Many algorithms begin by searching their input graph to obtain this structural information.

Searching in a Graph

Graph searching : systematically follow the edges of the graph so as to visit the vertices of the graph

Two basic graph searching algorithms:

- Breadth-first search
- Depth-first search
- The difference between them is in the order in which they explore the unvisited edges of the graph

Graph algorithms are typically elaborations of the basic graph-searching algorithms

Graph Searching Algorithms

Depth First Search

- Depth First Search is a powerful technique for solving many graph theory problems

- It is a systematic way of visiting all the vertices of a graph.

Description of Depth First search Traversal

1. Assume that a given graph G with V(G) = { $v_1, v_2, \ldots\ldots v_p$ } is represented by its adjacency lists.

2. Unless indicated otherwise, we assume , in the adjacency list of a given vertex that the vertices adjacent to that vertex are listed in increasing order of their subscripts.

3. In a depth first search of G, the vertex that is currently visited is designated as the **active vertex.**

Description of Depth First search Traversal

4. **A** depth first search of G is begun by selecting a first vertex to visit namely v_1. Vertex v_1 is the first active vertex and is assigned label 1.

5. Next, select a vertex adjacent to 1 (the first vertex on the adjacency list of 1). Label it 2 and this vertex becomes the new active vertex.

6. The edge joining the vertices labelled 1 and 2 is placed in a set *S*.

Description of Depth First search Traversal

7. In general, let n denote the label of the current active vertex in the search and suppose that not all vertices in the component of G containing n have been visited.

   We proceed as follows:

- If there are unvisited vertices adjacent to n, select the first vertex on the adjacency list of n that has not been visited and label it with the next available label.
- The vertex just labelled becomes the new active vertex.
- The edge joining n and this newly labelled vertex is placed in the set S.

Description of Depth First search Traversal

- If on the other hand, all the vertices adjacent to *n* have been visited, we backtrack *(i.e)* revisit the vertex that was the active vertex before *n* was first visited, and designate this vertex as the current active vertex.

- The general step is repeated until every vertex in that component of G has been visited.

- If not all vertices of G have been visited, then a vertex not yet visited , say the first such vertex is chosen as the next active vertex and the process continues.

Description of Depth First search(DFS) Traversal

- The label assigned to a vertex v in a graph G by the depth first search is called the **depth first search index** of v and is denoted by $dfi(v)$.

- When the DFS of G is completed, the number $dfi(v)$ is the order in which v was first visited during the search.

- Since each edge of G in S joins two vertices, one of which is being visited for the first time $< S >$ is a spanning forest of G, called the **Depth First Search Forest**.

Description of Depth First search(DFS) Traversal

- If G is connected, then $< S >$ is a spanning tree, called a **depth first search tree.**

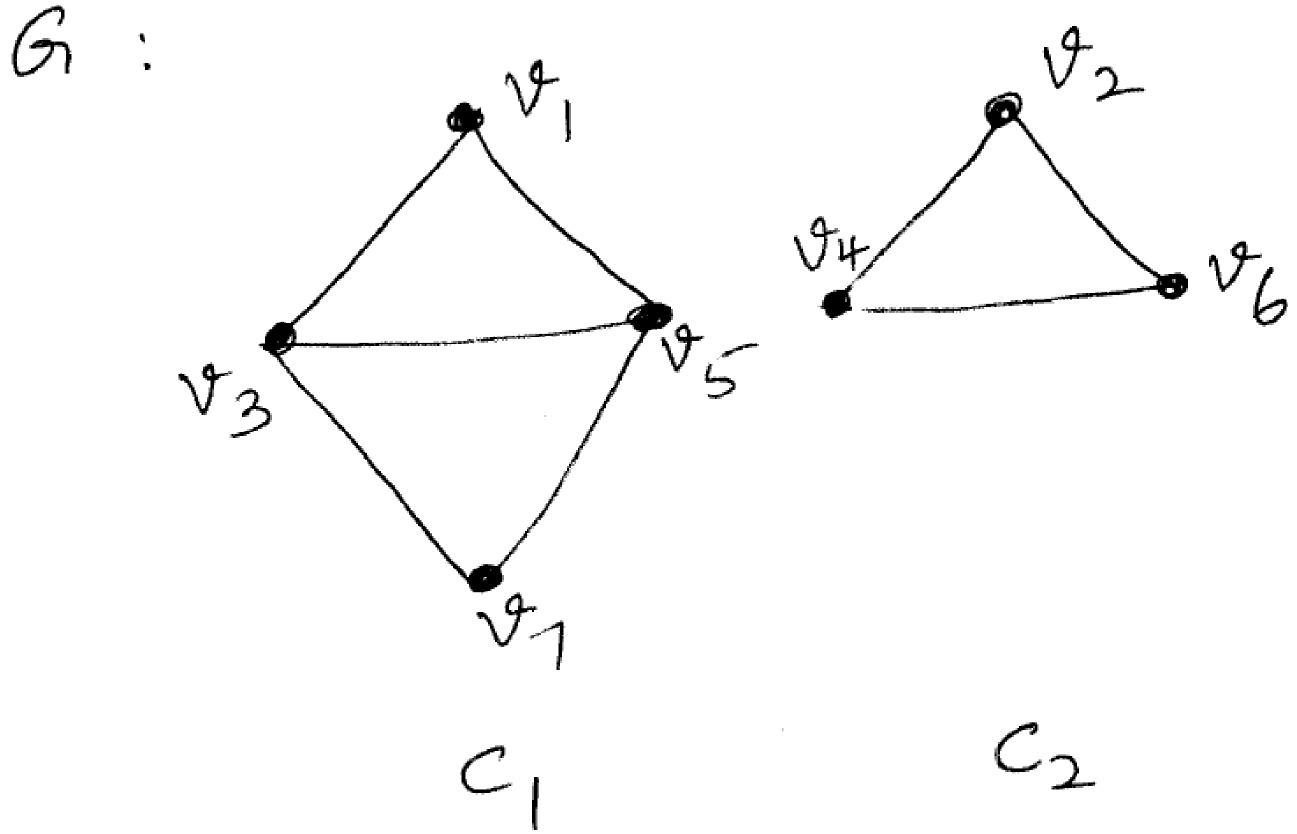
- If F is a depth first search forest of a graph G, then each component of F is a rooted tree in which the root is the first vertex visited in the component.

- Each edge of G that is not an edge of F is called a **back edge.**

- Necessarily, each back edge joins two vertices in a component of G and thus in a component of F.

Example

1. Find the depth first search tree or forest in the graph G.



Solution

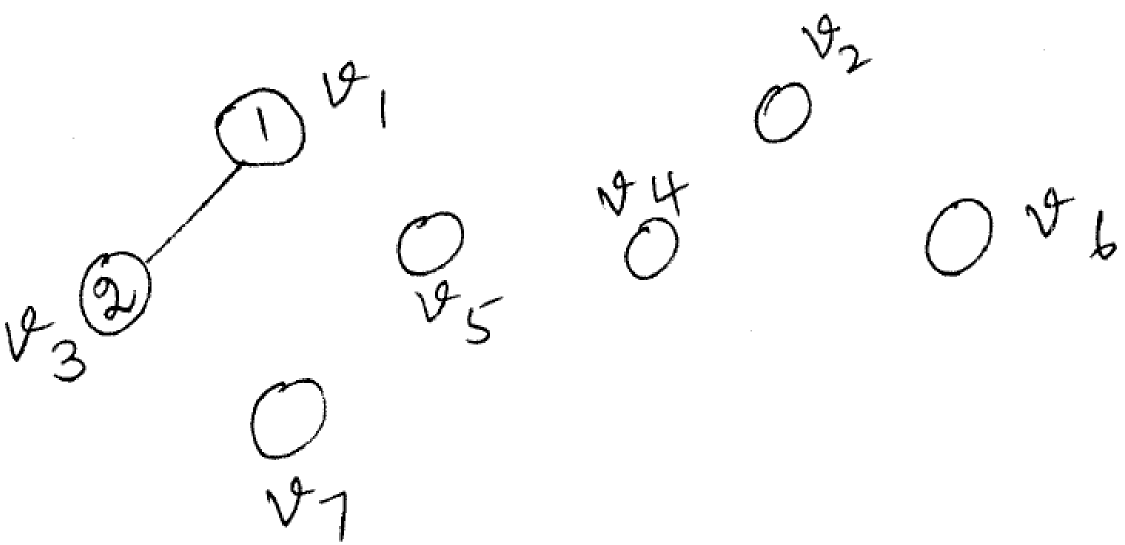
G is a disconnected graph with 2 components C_1 and C_2.

Step 1

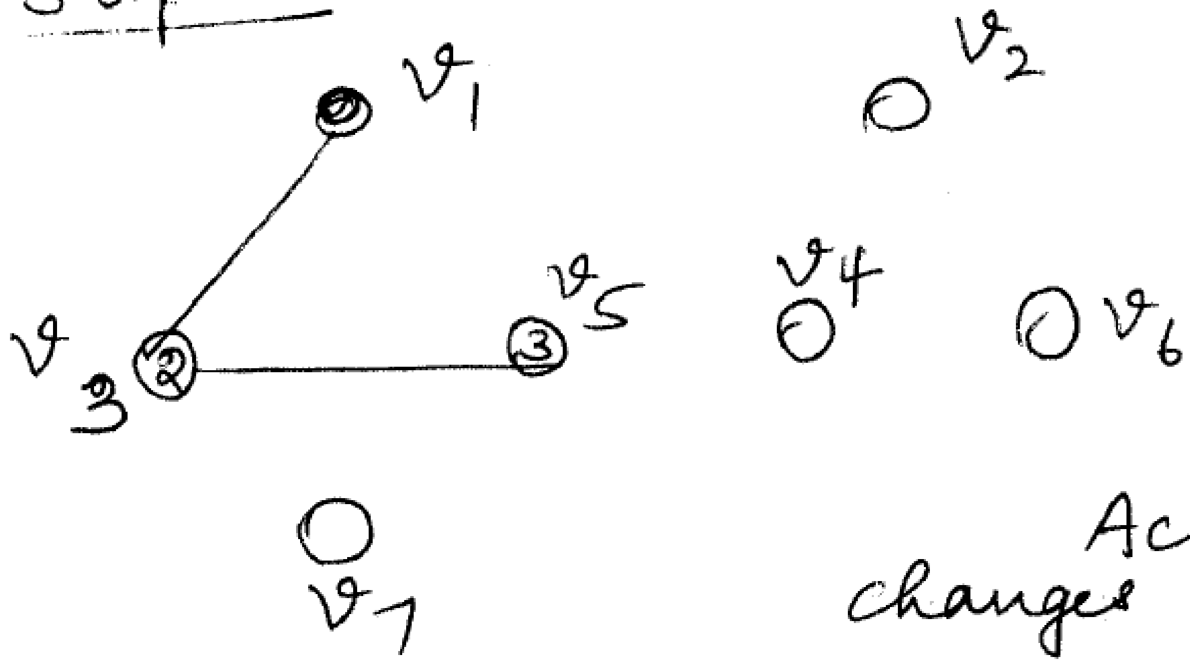
Edges in S and labels of vertices

Active vertex changes

From	To
v_1	v_3

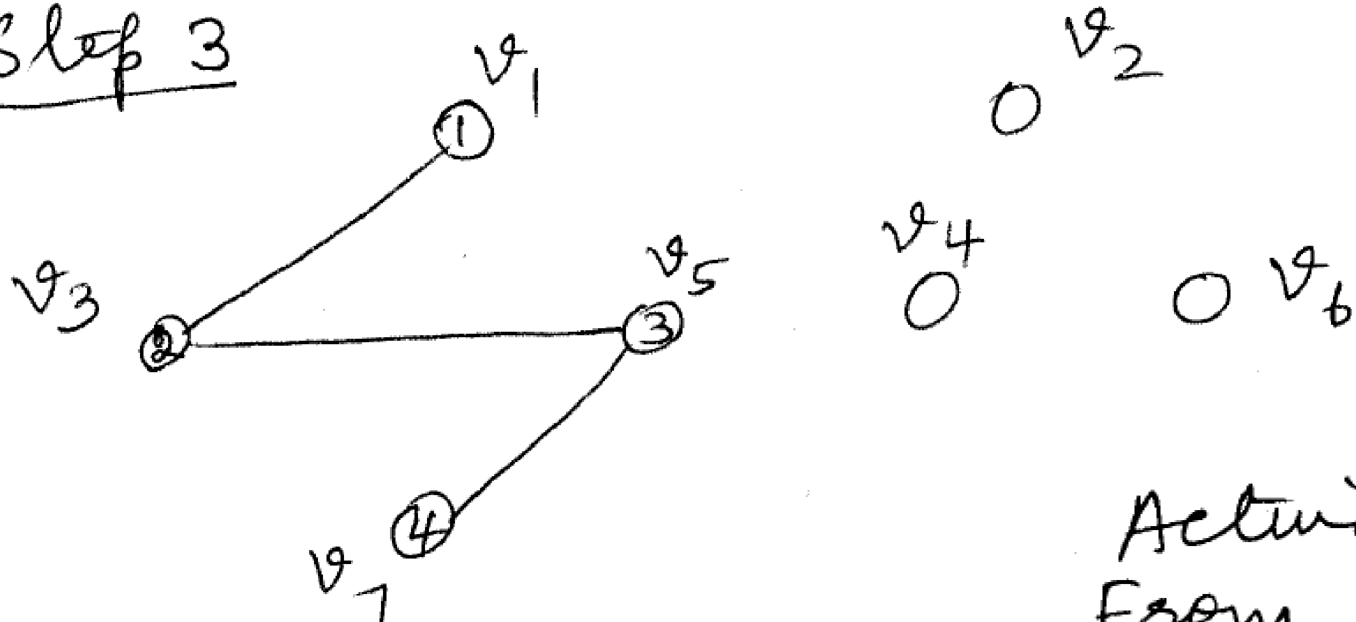


Step 2



Active vertex changes From v_3 To v_5

Step 3

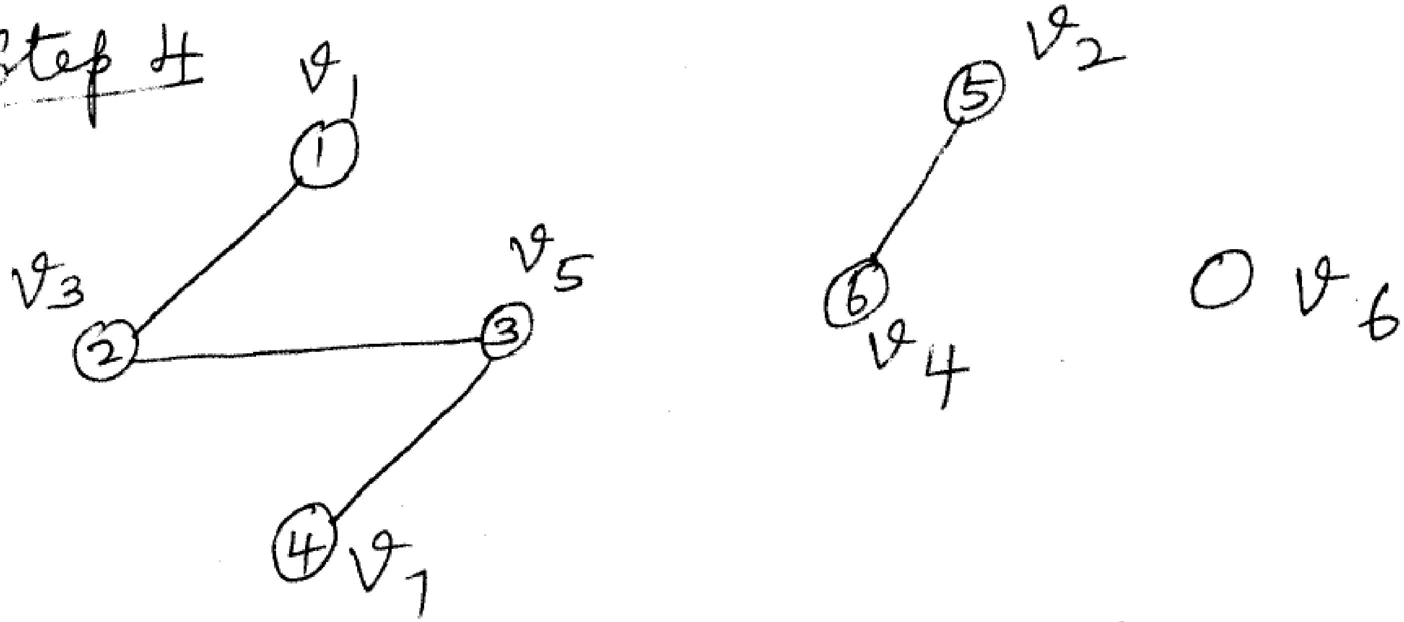


Active vertex

From	To
v_5	v_7

- All vertices in component C_1 have been visited;
- v_2 is the new active vertex.

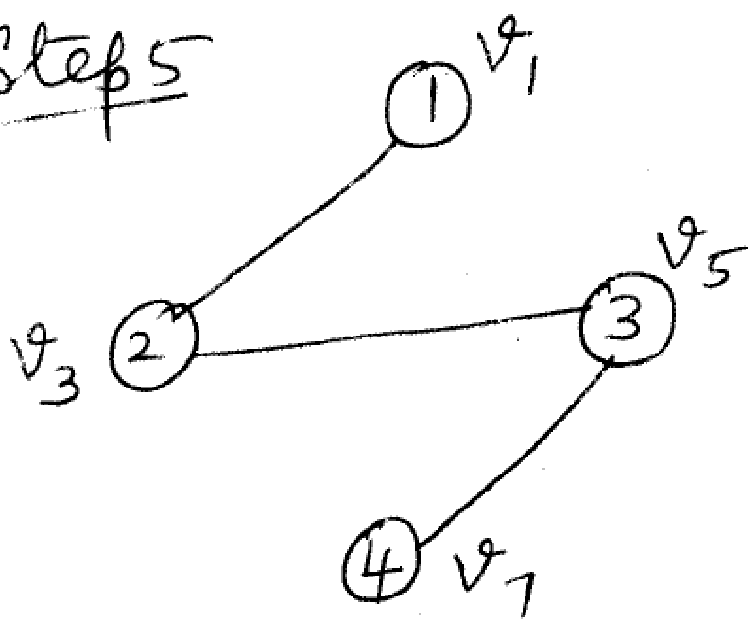
Step 4

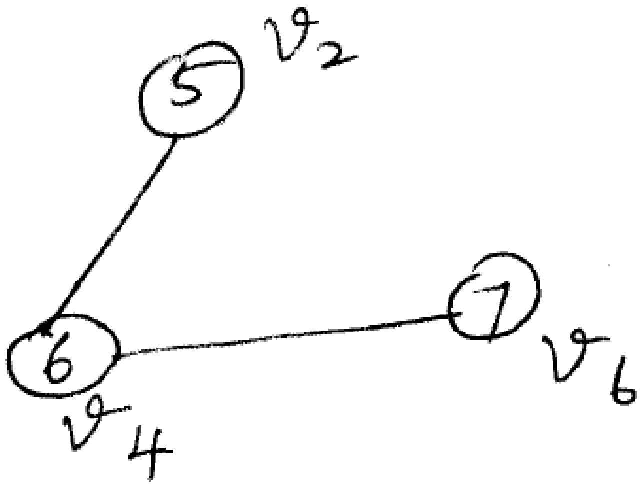


Active vertex

From	To
v_2	v_4

Step 5





Active vertex

From	To
v_4	v_6

- This is the DFS forest *F*.

Depth First search(DFS) Algorithm

To conduct a depth first search of a graph G represented by its adjacency lists.

1. *dfi(v)* <- 0 for each *v* ε V(G)
   [Initially, all vertices are given a depth first search index of 0]

2. *i* <- 1
   [The parameter *i* is initialised and will be assigned to the i^{th} vertex visited during the search.

3. *S* <- ϕ
   [The set *S* is initialised and will be the arc set of the DFS forest.]

Depth First search(DFS) Algorithm

4. If $dfi(r) \neq 0$ for all $r \in V(G)$, then output S and stop;
   [If not all vertices of G have been visited, a new root is selected from which a depth first search of the component of G containing that vertex is conducted.]

   Otherwise, let r be the first vertex such that $dfi(r) = 0$ and let w <- r.

5. $dfi(w)$ <- i

6. i <- $i + 1$

Depth First search(DFS) Algorithm

7. [A search is conducted for a vertex not yet visited.]

   7.1 If $dfi(v) = 0$ for some vertex v in the adjacency list of w, then continue; Otherwise go to step *7.4*.

   7.2 $S \leftarrow S \cup \{(w,v)\}$ and assign $Parent(v) \leftarrow w$

   7.3 $w \leftarrow v$ and return to step 5.

   7.4 If $w \neq r$, then $w \leftarrow Parent(w)$ and return to step 7.1; Otherwise, return to step 4.

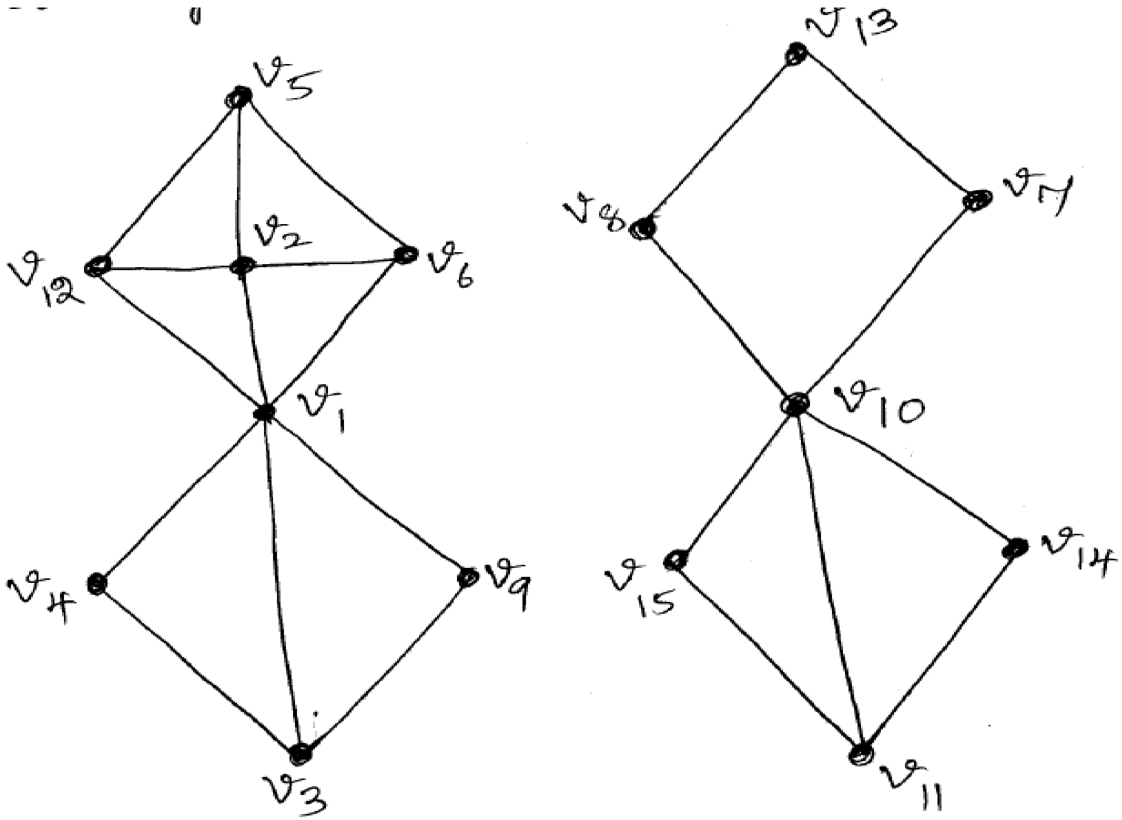
Depth First search(DFS) Algorithm

Time complexity:

If G is a graph with |V| vertices and |E| edges, then the complexity of a depth first search of G is $\Theta(|V| + |E|)$.

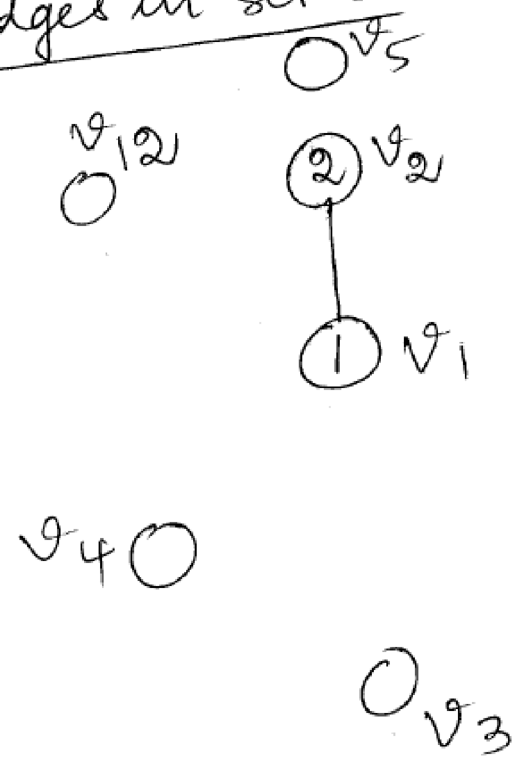
- Example

- Apply a DFS to a graph G and make a table of the DFS index and the corresponding stack.



Step 1

Edges in set S



Active vertex

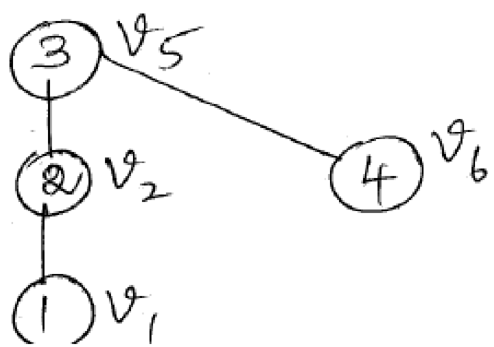
v_1

Step 2

Active vertex changes from

v_1 v_2

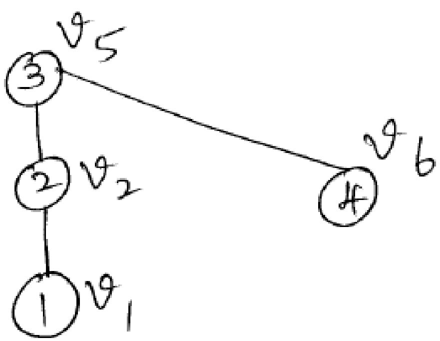
Step 3



Active vertex changes from

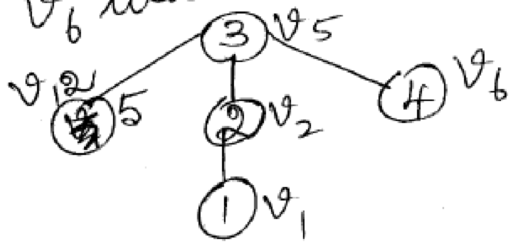
$v_2 \rightarrow v_5$

Step 4



Active vertex changes from $v_5 \rightarrow v_6$

Step 5 Since all vertices adjacent to v_6 have been visited. Active vertex is the previous vertex that was the active vertex before v_6 was visited.



Active vertex $v_6 \rightarrow v_5$

Step 6 since all vertices adjacent to v_{12} have been visited, we back track & go to the previous active vertex.

Active vertex changes from

v_{12} v_5

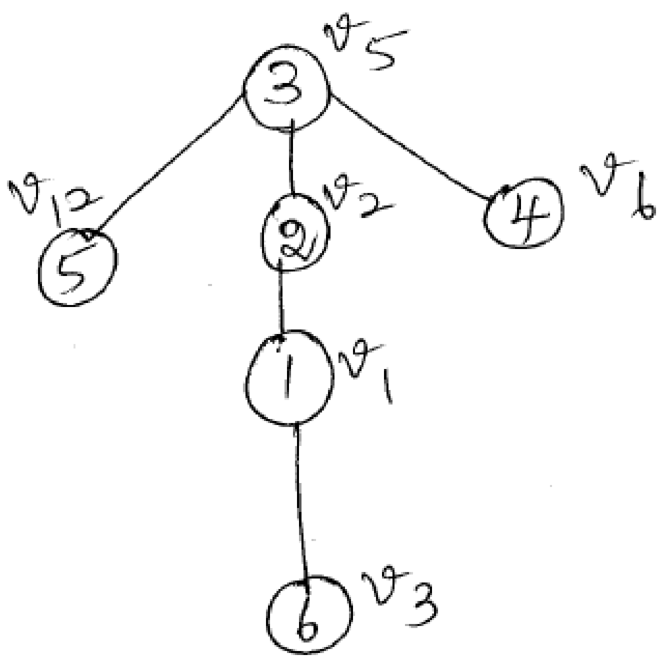
step 7 same reason as step 6.

Active vertex changes from

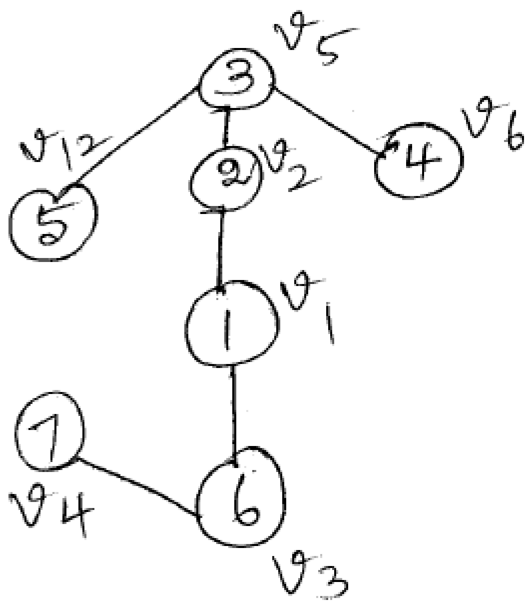
$v_5 \rightarrow v_2$

step 7 Active vertex changes from v_2 to v_1.

Active vertex
$v_2 \rightarrow v_1$



Step 8



Active vertex changes from $v_1 \longrightarrow v_3$

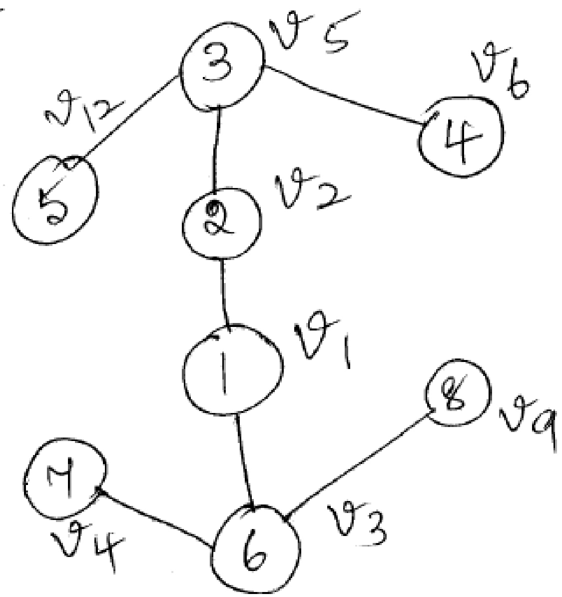
step 4

since all vertices adjacent to v_4 have been visited we back track & go to the vertex which was the active vertex before v_4.

Active vertex changes from $v_3 \to v_4$

Step 10

F_1 :



Active node changes

$v_4 \rightarrow v_3$

Step 11 since, all vertices adjacent to v_9 have been visited all all vertices in this component of G have nonzero labels, we move to the next component.

Step 12.

F_1

(9) v_7

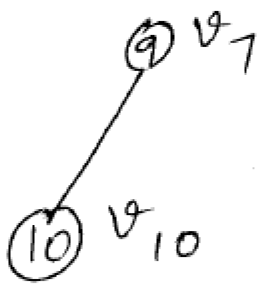
Active vertex changes from

$v_3 \rightarrow v_9$

Active vertex changes from

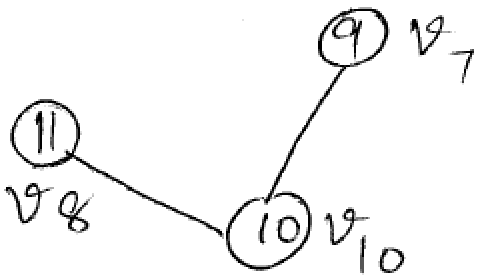
$v_9 \rightarrow v_7$.

Step 13



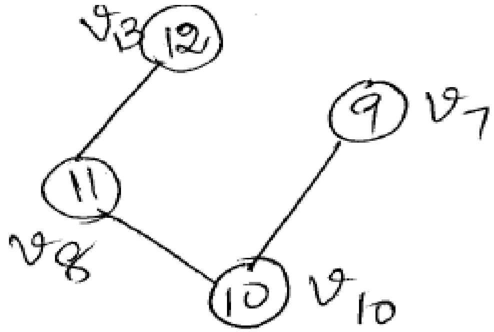
Active vertex changes from $v_7 \longrightarrow v_{10}$

Step 14



Active vertex changes from $v_{10} \longrightarrow v_8$

Step 15

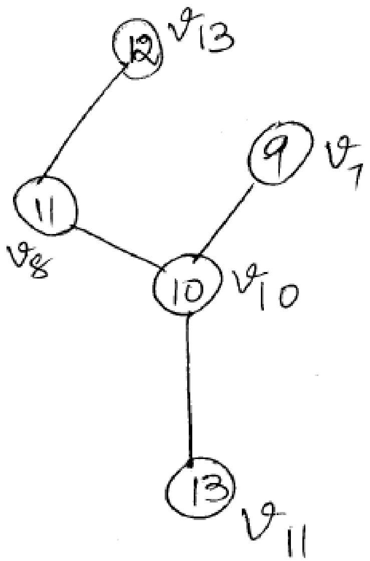


Active vertex changes from $v_8 \to v_{13}$.

Step 16. since all vertices adjacent to v_{13} have been visited, we go to the previon active vertex.
(ie) active vertex changes from
$v_{13} \to v_8$

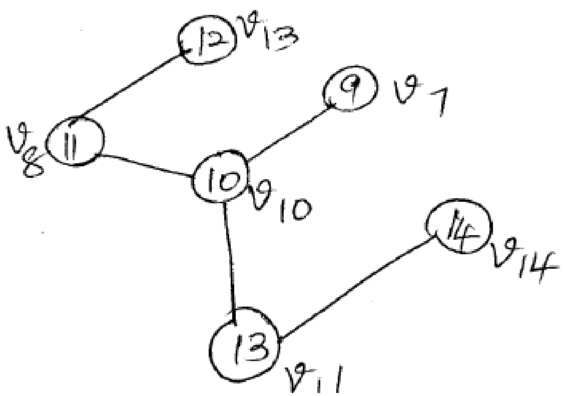
Step 17 Same as step 16. we backtrack active vertex changes from
$v_8 \to v_{10}$.

Step 18.



Active vertex changes from $v_{10} \rightarrow v_{11}$

Step 19



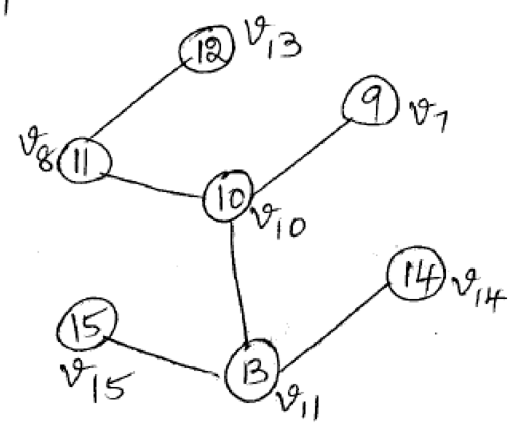
Active vertex changes from $v_{11} \rightarrow v_{14}$

Step 20 Since all vertices adjacent to v_{14} have been visited, we back track to the previous active vertex.

∴ Active vertex changes from

$v_{14} \to v_{11}$

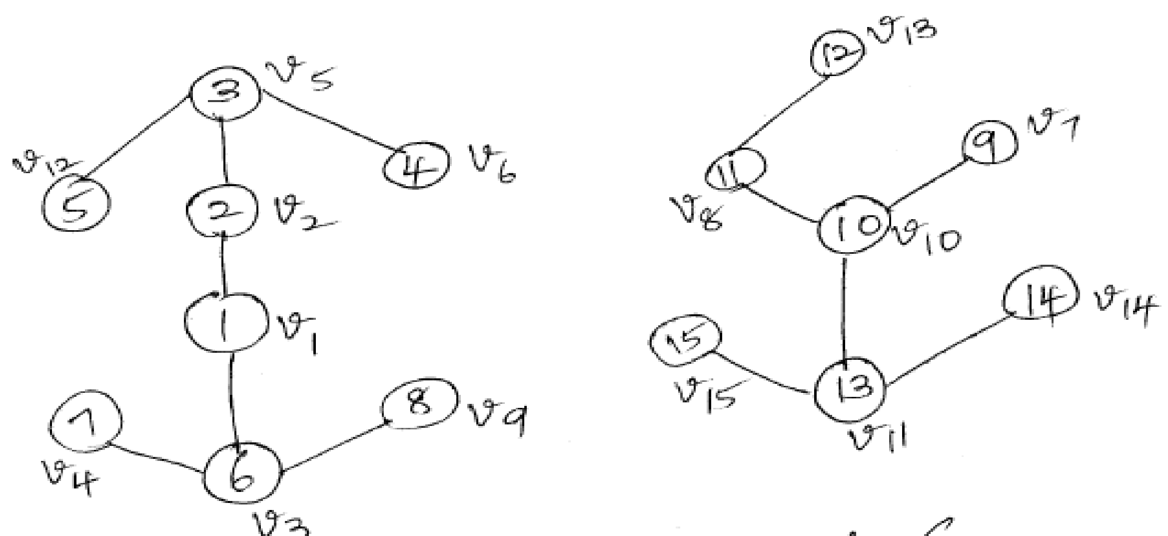
Step 21



Active vertex changes from

$v_{11} \to v_{15}$

Step 22 Since all vertices adjacent to v_{15} have been visited & by backtracking we find that all vertices have been visited (have nonzero labels), then DFS is complete & the DFS forest is output.

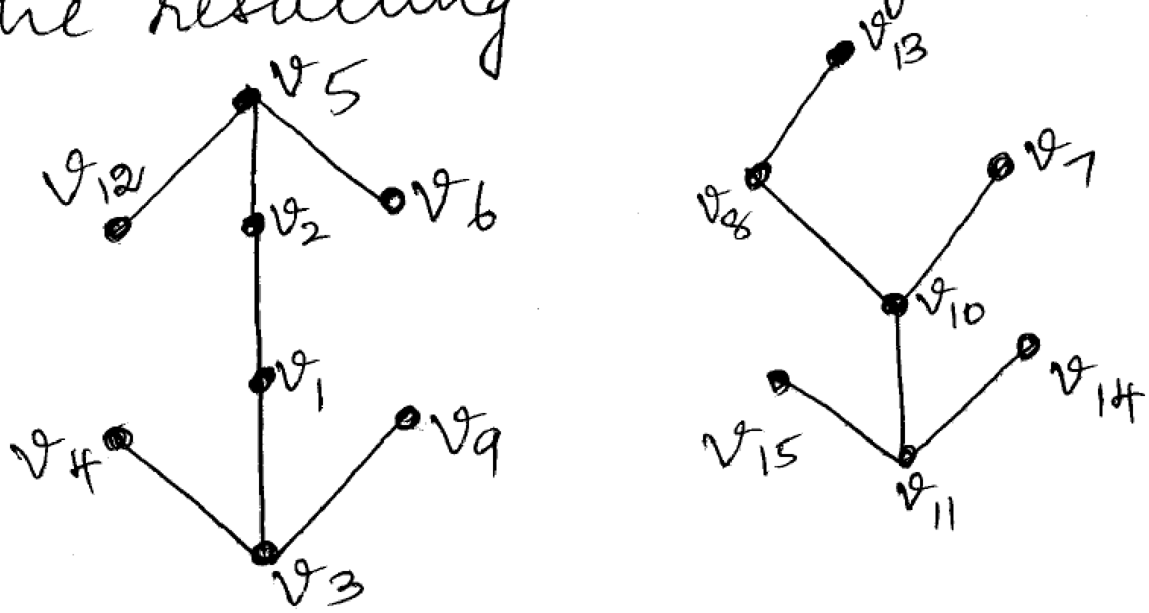


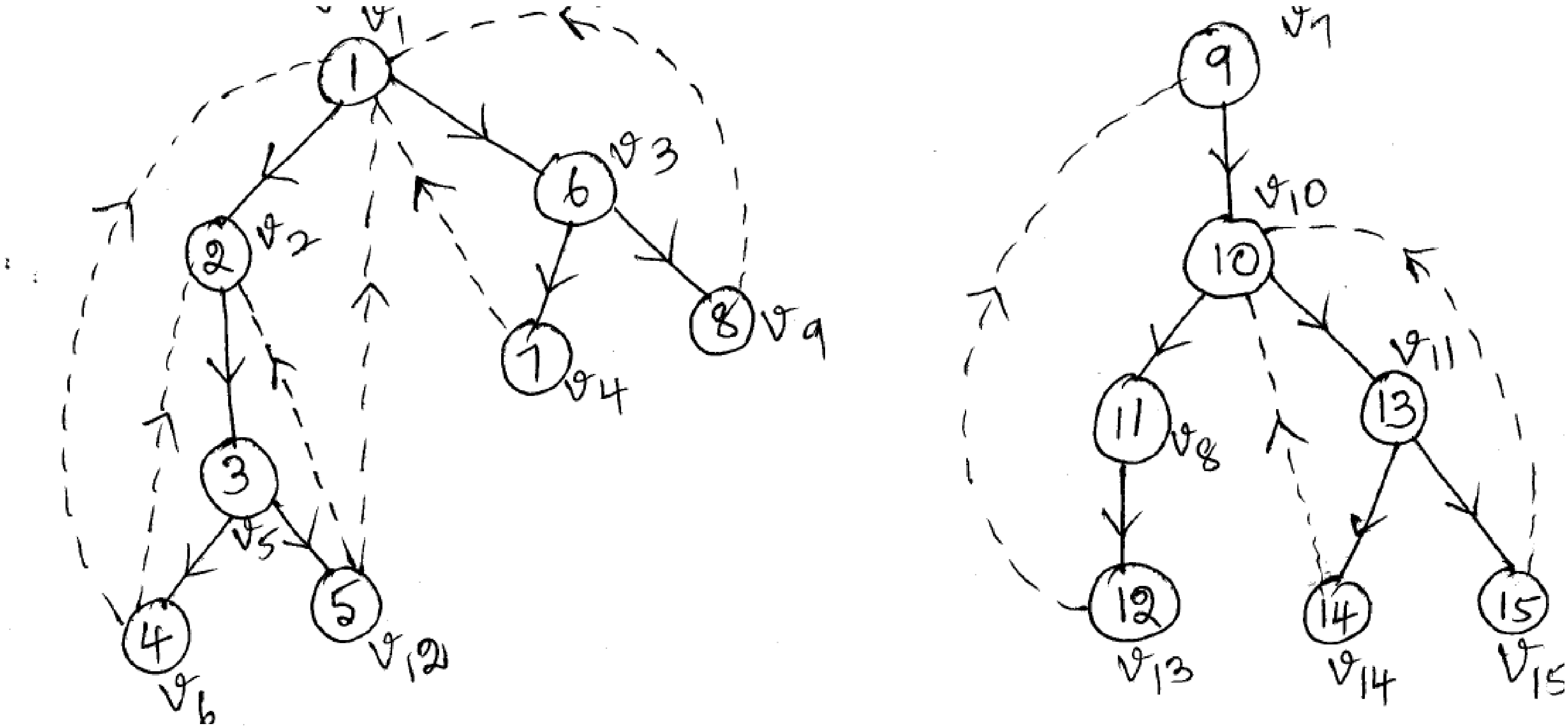
Depth first search forest of G.

G is a disconnected graph with 2 components.

The resulting DFS forest is

F:





The back edges of G are shown in the below figure by dotted lines.

v	$dfi(v)$
v_1	1
v_2	2
v_3	6
v_4	7
v_5	3
v_6	4
v_7	9
v_8	11
v_9	8
v_{10}	10
v_{11}	13
v_{12}	5
v_{13}	12
v_{14}	14
v_{15}	15

v_{15}
v_{14}
v_{11}
v_{13}
v_8
v_{10}
v_7
v_9
v_4
v_3
v_{12}
v_6
v_5
v_2
v_1

Stack

Applications of Depth First Search

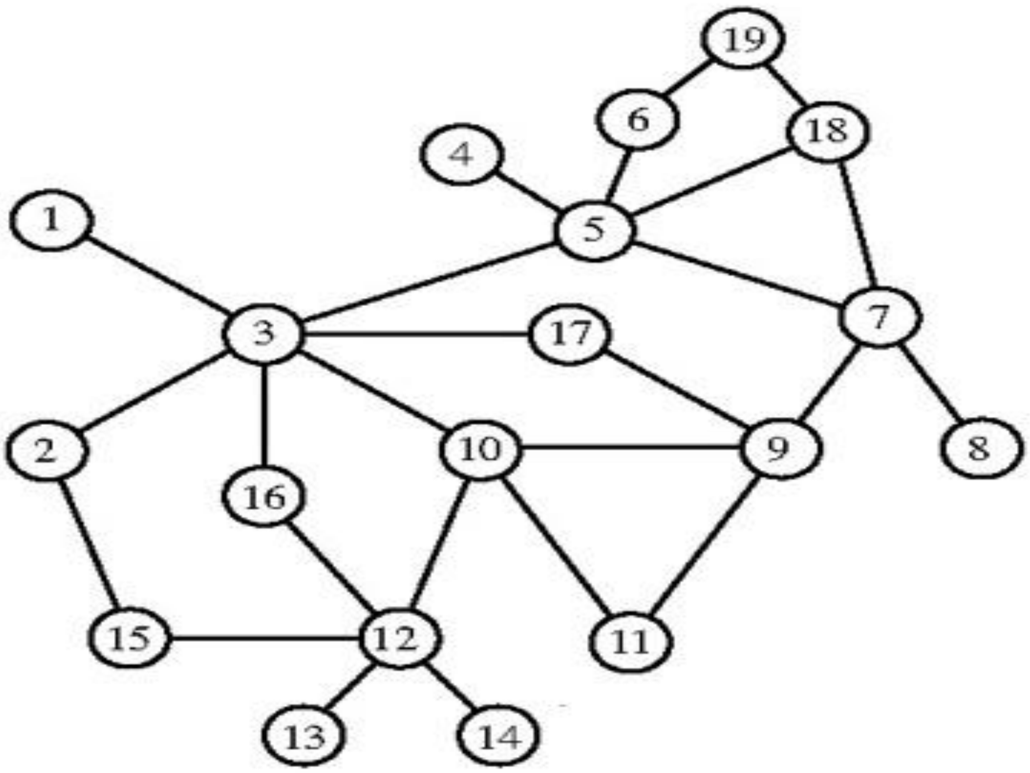
- To check if a graph is connected or disconnected.

- Finding the components of a disconnected graph

- Other applications include finding cut vertices, bridges and blocks in a graph.

Example:

- Apply a depth first search to find the DFS tree. Also, write down the stack formed and the table of depth first search index for all vertices.



- Breadth First Search

innovate achieve lead

Breadth First Search (BFS)

- Another useful tool- searching technique for a graph

•The BFS visits systematically the vertices of a graph or digraph beginning at some vertex *r* of *G* also called a root.

•The root is the first active vertex.

•At any stage during the search, all the vertices adjacent from the current active vertex are scanned for vertices that have not yet been visited, that is a “broad” search is performed for unvisited vertices.

Breadth First Search (BFS)

- Each time a vertex is visited for the first time, it is labelled and added to a back of a queue.

- Note that a queue is used rather than a stack.

- The current active vertex is the one at the front of the queue.

- As soon as its neighbours have been visited, it is deleted from the queue.

Breadth First Search (BFS)

- If the queue is empty and some vertices of the graph or digraph have not yet been visited, we select any unvisited vertex , assign it a label and add it to the queue.

- When all the vertices of the graph have been visited, the search is complete.

- Assume that *G* is a graph represented by its adjacency lists.

- Initially, all vertices of *G* are labelled 0.

Breadth First Search (BFS)

- We begin by assigning *r*, the label 1 and placing *r* on a queue *Q*.

- At the next step, we delete *r* from *Q* and scan its adjacent vertices(if such vertices exist) sequentially in the order in which they appear on the adjacency list for *r*.

- The first vertex that appears on the adjacency list for *r* is assigned the next available label, namely, 2 and this vertex is then added to the back of the *Q*.

Breadth First Search (BFS)

- We continue to label the vertices adjacent to r and add them to Q until the last vertex adjacent with r is labelled $deg(r) + 1$ and is added to Q.

- We then delete the next vertex from the front of the Q, say w and scan its adjacent vertices in the order in which they appear on the adjacency list of w.

- If a vertex adjacent with w still has label 0, then we assign it the next available label and add it to Q; Otherwise, we do not change its label.

Breadth First Search (BFS)

- We continue in this manner until Q is empty.

- If all the vertices of G are labelled with a positive integer, we stop.

- Suppose G still contains vertices labelled 0 which will happen if G is disconnected.

- Then, we select such a vertex, assign it the next available label and we continue as before.

Breadth First Search (BFS)

• This breadth first algorithm terminates once every vertex has been assigned a positive integer label.

•This algorithm actually determines a spanning forest F of G called a **breadth first search forest** where each component of F is a rooted tree.

•The root of a component of F is then the vertex having the smallest label in that component.

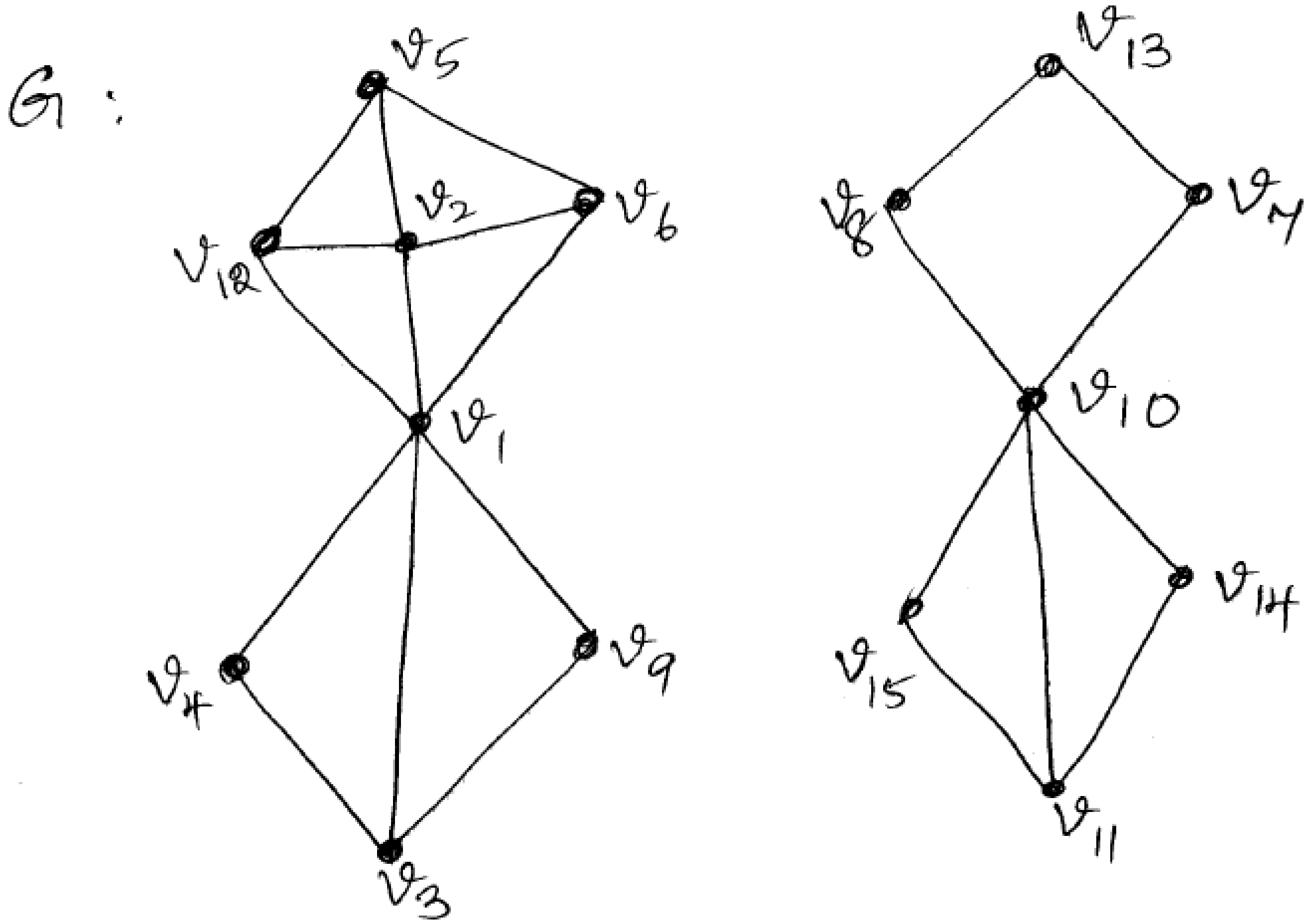
Breadth First Search (BFS)

- Further, an edge vw of G is added to F if either v is deleted from Q and w still has label 0, or w is deleted and v still has label 0.

- Time complexity of a breadth first search is $\Theta(|V|+|E|)$.

Example:

- Apply a breadth first search to the graph G.



Example:

Constructing a BFS forest

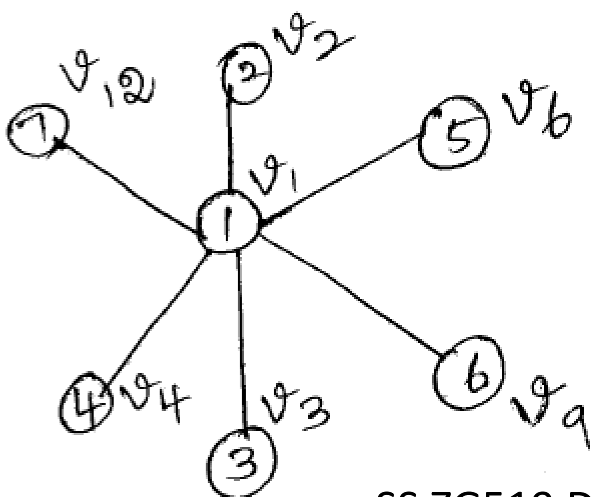
Step 1

Queue

v_1

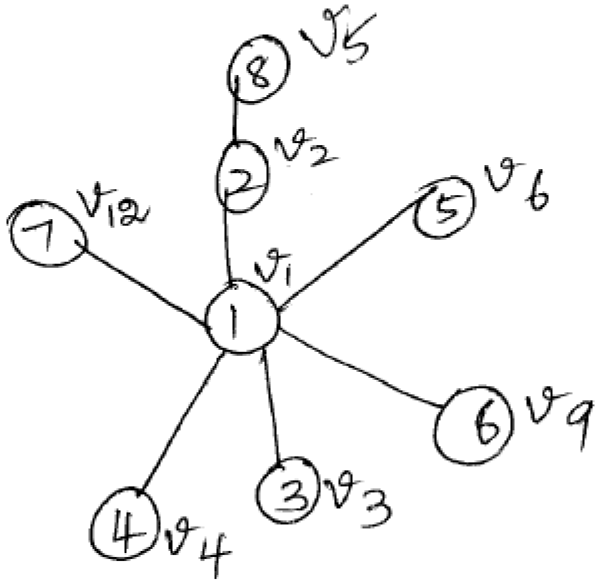
Step 2

$\cancel{v_1}\ v_2\ v_3\ v_4\ v_6\ v_9\ v_{12}$



Example:

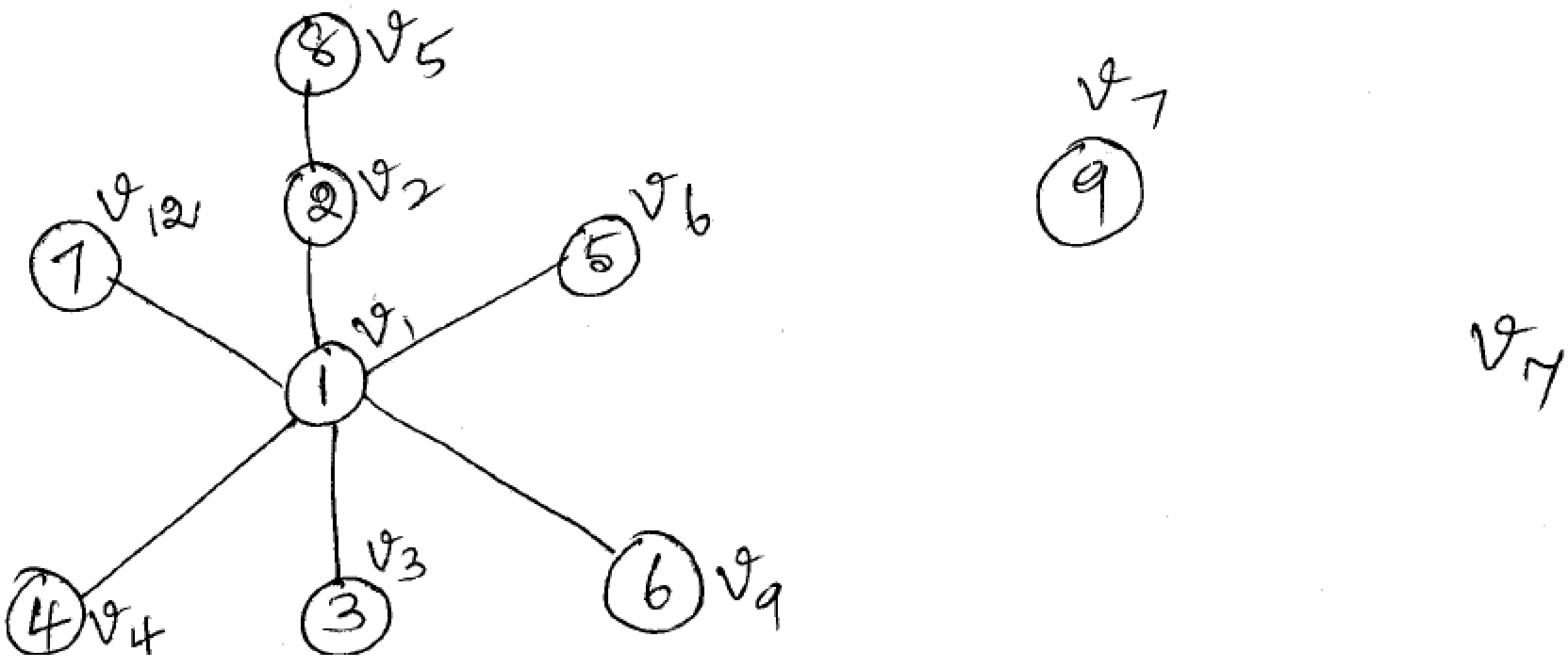
Step 3



$\cancel{v}_2\ \cancel{v}_3\ \cancel{v}_4\ \cancel{v}_6\ \cancel{v}_9\ \cancel{v}_{12}\ v_5$

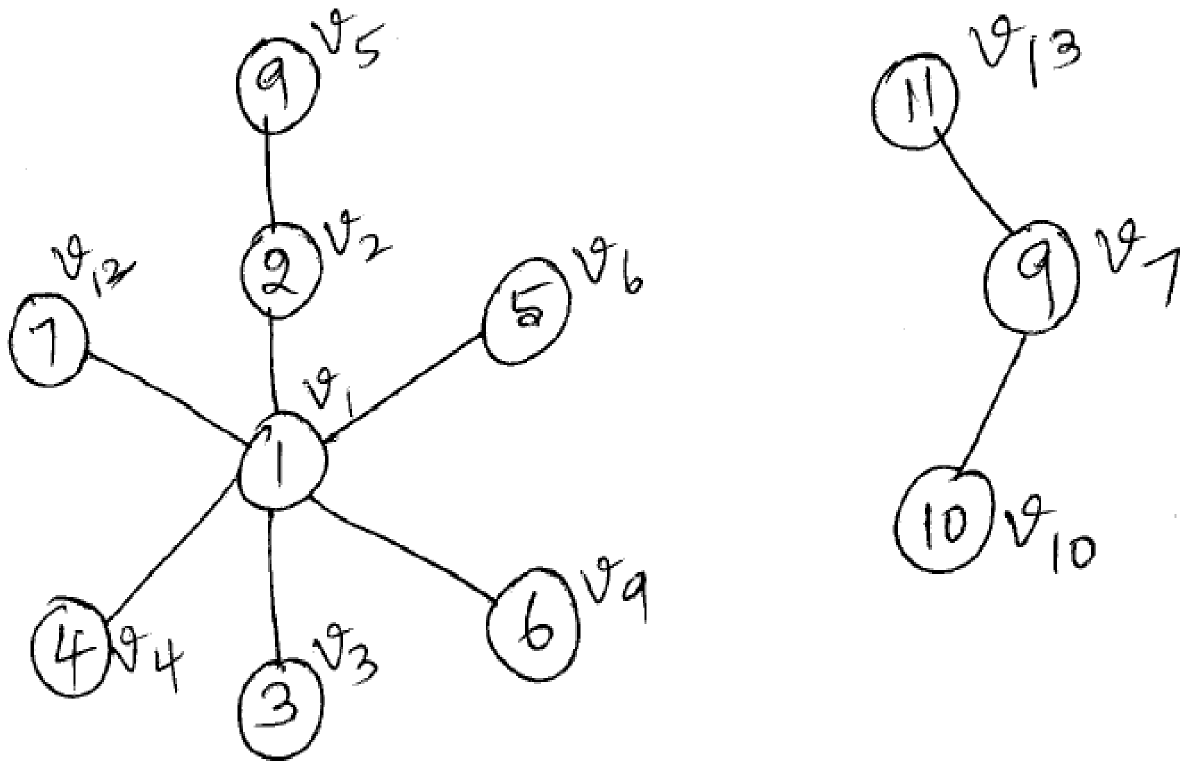
Example:

Step 4



Example:

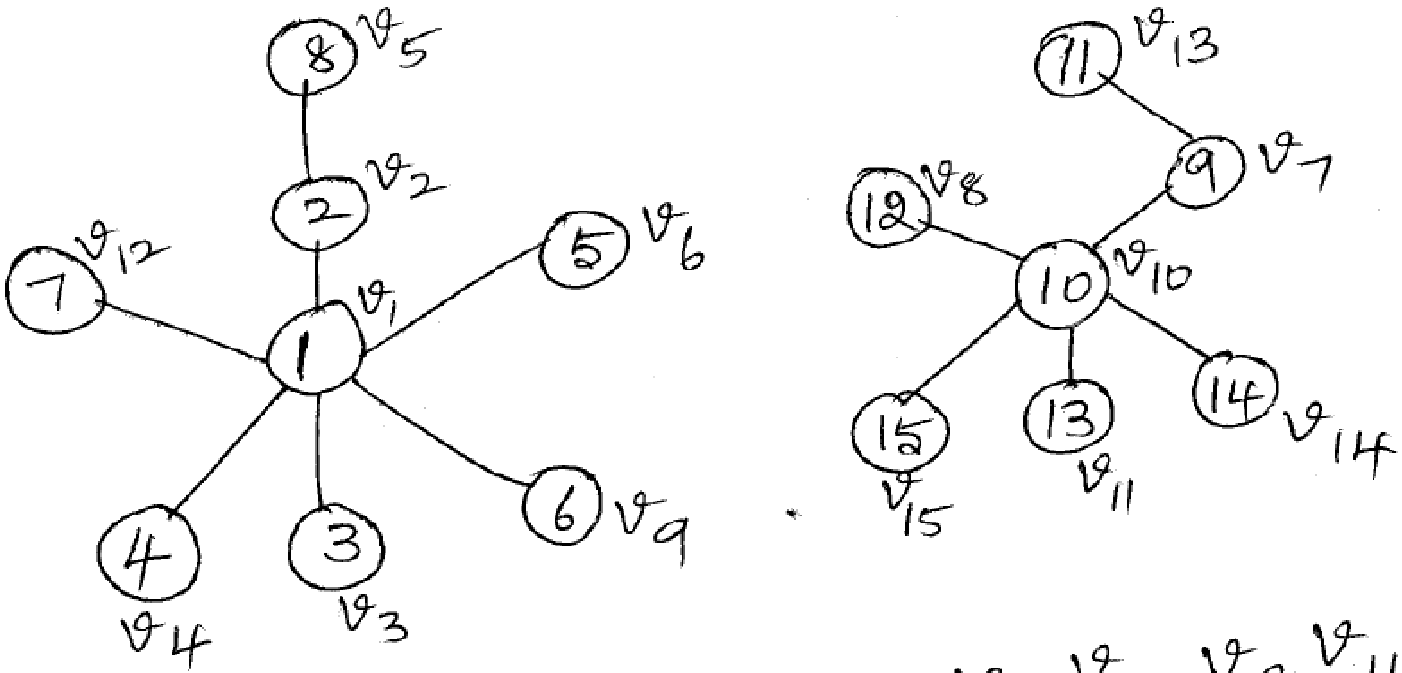
Step 5



$\cancel{v_7}$ v_{10} v_{13}

Breadth First search forest of G

Step 6

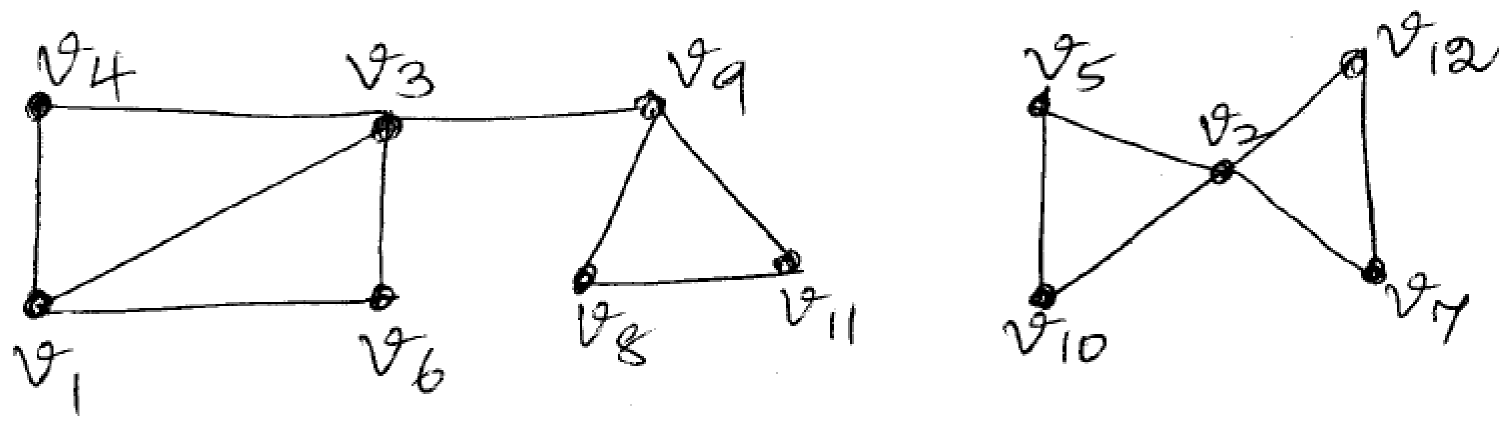


$v_{10}\ v_{13}\ v_8\ v_{11}\ v_{14}\ v_{15}$

Example

- Apply a breadth first search to the graph G.

G :



Example

Step 1

Constructing a BFS forest

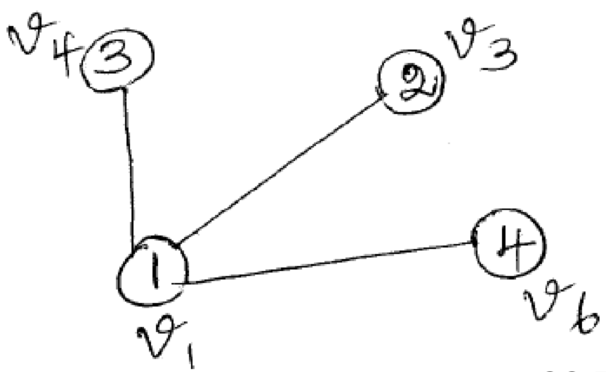
Queue

v_1

Queue

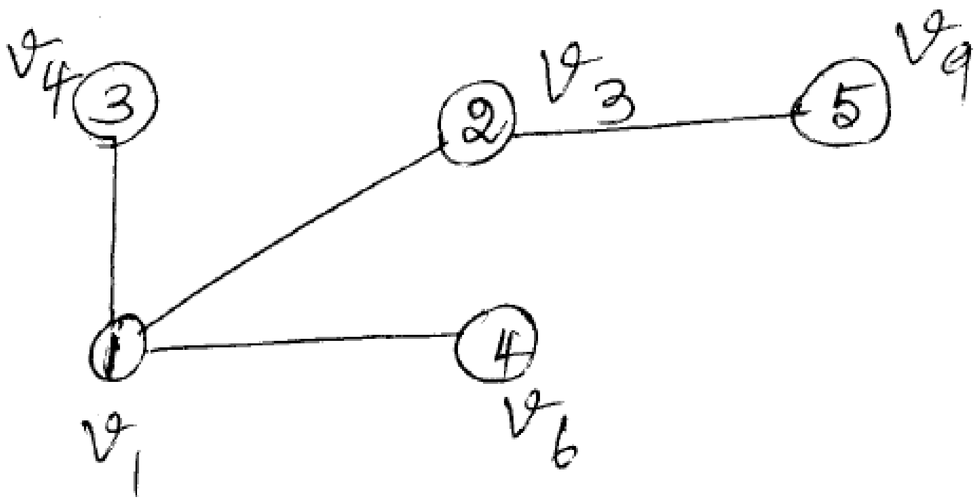
Step2

$\cancel{v_1}\ v_3\ v_4\ v_6$



Example

Step3



Queue

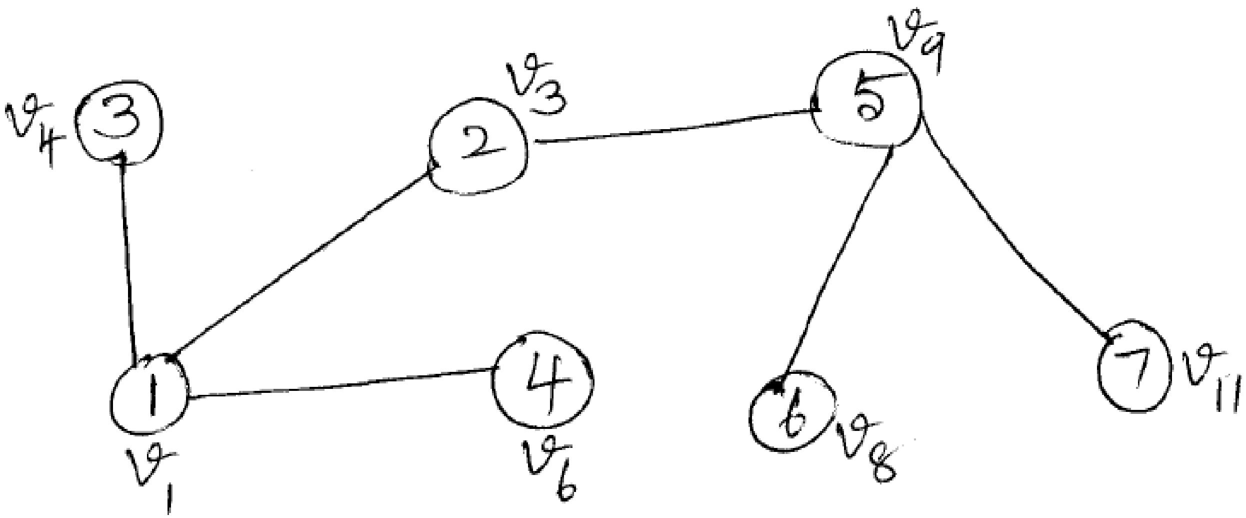
~~v_3~~ v_4 v_6 v_9

Example

Step 4

Queue

~~v_4~~ ~~v_6~~ ~~v_9~~ ~~v_8~~ ~~v_{11}~~



F_1

Example

step 5

F1

8 v_2

Queue

v_2

innovate achieve lead

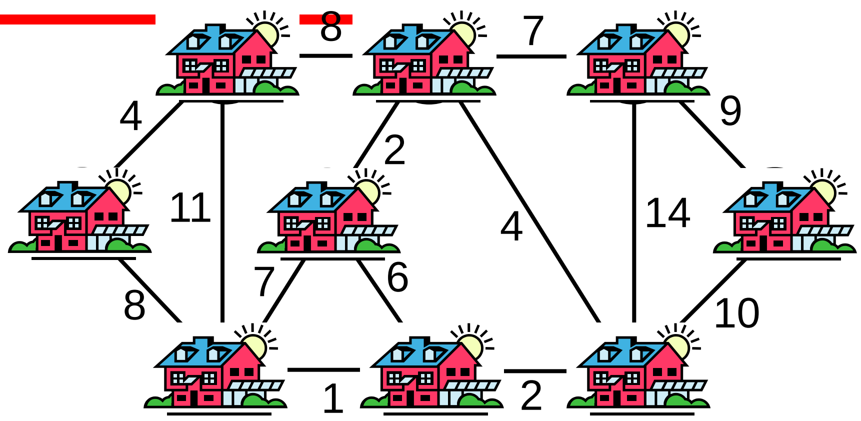
BITS Pilani
Pilani Campus

- Minimum Spanning Trees

Minimum Spanning Trees

- Kruskal's algorithm
- Prim's algortihm

Minimum Spanning Trees



Problem

- A town has a set of houses and a set of roads
- A road connects 2 and only 2 houses
- A road connecting houses u and v has a repair cost w(u, v)

Goal: Repair enough (and no more) roads such that:

1. Everyone stays connected: can reach every house from all other houses, and
2. Total repair cost is minimum

Minimum Spanning Trees

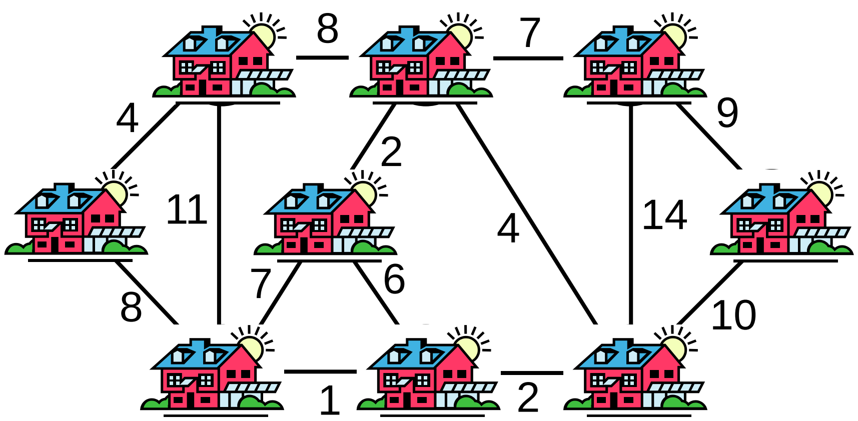
A connected, undirected graph:

– Vertices = houses, Edges = roads

A **weight** $w(u, v)$ on each edge $(u, v) \in E$

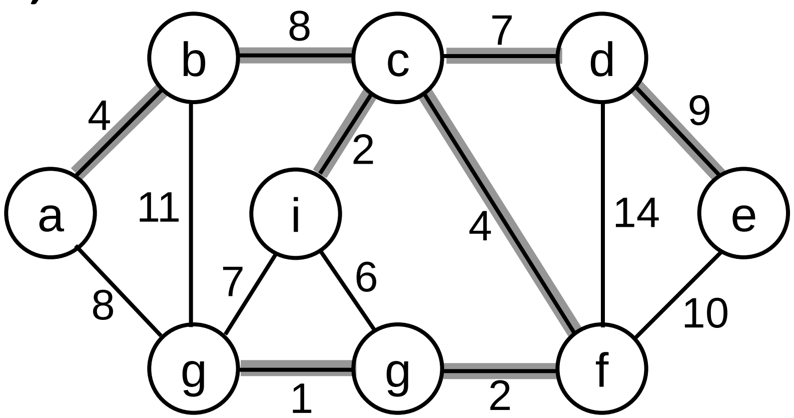
Find $T \subseteq E$ such that:

1. T connects all vertices
2. $w(T) = \Sigma_{(u,v)\in T}\, w(u, v)$ is minimized



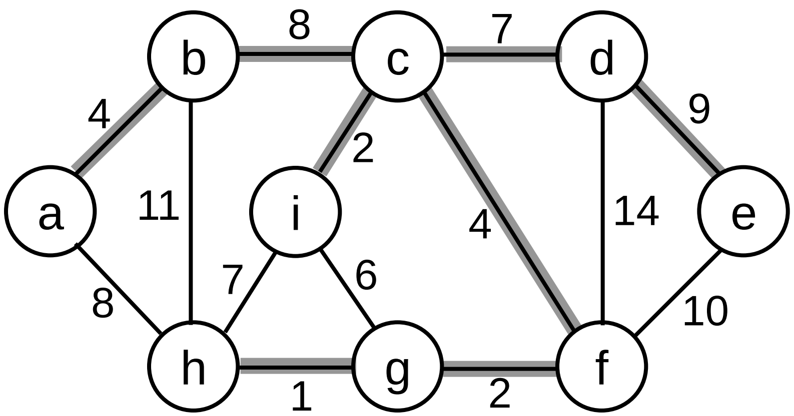
Minimum Spanning Trees

- T forms a tree = **spanning tree**
- A spanning tree whose weight is minimum over all spanning trees is called a ***minimum spanning tree***, or ***MST***.



Properties of Minimum Spanning Trees

- Minimum spanning trees are not unique
  - Can replace (b, c) with (a, h) to obtain a different spanning tree with the same cost



- MST have no cycles
  - We can take out an edge of a cycle, and still have the vertices connected while reducing the cost

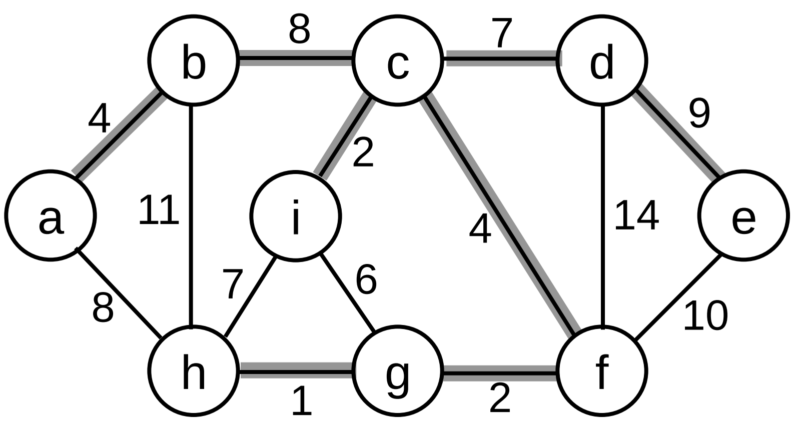
- # of edges in a MST:
  - |V| - 1

Growing a MST

- **Minimum-spanning-tree problem**: find a MST for a connected, undirected graph, with a weight function associated with its edges

A generic solution:

- Build a set A of edges (initially empty)
- Incrementally add edges to A such that they would belong to a MST
- An edge (u, v) is **safe** for A if and only if A ∪ {(u, v)} is also a subset of some MST



We will add only safe edges

The Minimum Spanning Tree Problem

• Suppose that a group of volunteers have entered an under developed country to assist the residents of several villages.

•Telephone lines must be built along some existing roads that connect the villages with each other.

•We wish to erect these telephone lines in such a way that every pair of villages can communicate by telephone.

•Moreover, the total number of miles of telephone line is to be minimised.

The Minimum Spanning Tree Problem

- The problem is to determine along which roads these telephone lines should be erected to produce the desired telecommunication system.

- Constructing such a telecommunications network has obvious graphical overtones.

- We can associate a graph with this situation where each vertex corresponds to a village and an edge between 2 vertices represents a road between the corresponding villages.

The Minimum Spanning Tree Problem

- The length of such a road is indicated in the graph by assigning a weight to the corresponding edge.

- Thus, we have produced a weighted graph G.

- The weight W(H) of a subgraph H of a weighted graph is the sum of the weights of the edges of H.

- The solution of our problem requires us to find a connected spanning subgraph H of the weighted graph G with the least possible weight.

The Minimum Spanning Tree Problem

- The spanning subgraph H is a tree.

- Thus a desired telecommunications network (having a minimum number of miles of telephone line) corresponds to a spanning tree T of G having minimum weight.

- Such a tree is called a **minimum spanning tree.**

Kruskal's Algorithm

• Find a minimum spanning tree(MST) in a connected weighted graph.

•The MST problem was originally stated by Boruvka in 1926 while considering the rural electrification of southern Moravia in Czechoslovakia.

•A number of algorithmic solutions of this problem have been given.

.

Kruskal's Algorithm

- Most famous is due to Kruskal.

•The object of Kruskal's algorithm is to select edges of minimum weight successively from a connected weighted graph without forming cycles until a spanning tree has been produced.

•Running time of Kruskal's algorithm is $O(|E| \log |V|)$

innovate achieve lead

Kruskal's Algorithm

Kruskal's Algorithm

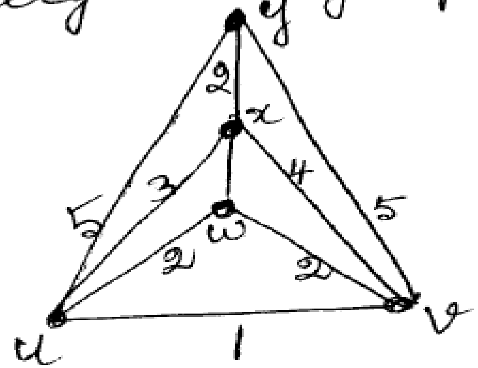
To determine a minimum spanning tree in a nontrivial connected weighted graph G, (p, q)

1. [Initialize the set S which will consist of the edges of a minimum spanning tree]
   $S \leftarrow \phi$.
2. [The set S is incremented]
   Let e be an edge of minimum weight such that $e \notin S$ and $\langle S \cup \{e\} \rangle$ is acyclic, and let $S \leftarrow S \cup \{e\}$.
3. If $|S| = p-1$, then output S; otherwise return to step 2.

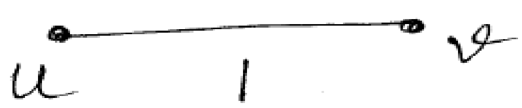
Example

<u>Example</u> Apply Kruskal's algorithm to the weighted graph G.

G :

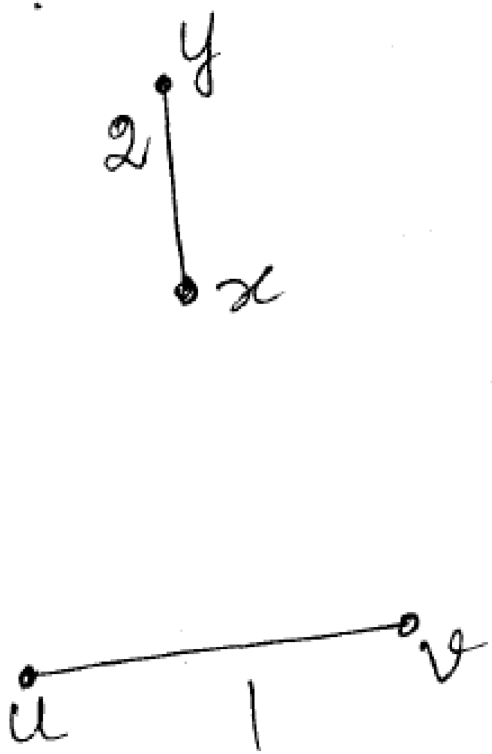


<u>Step 1</u>



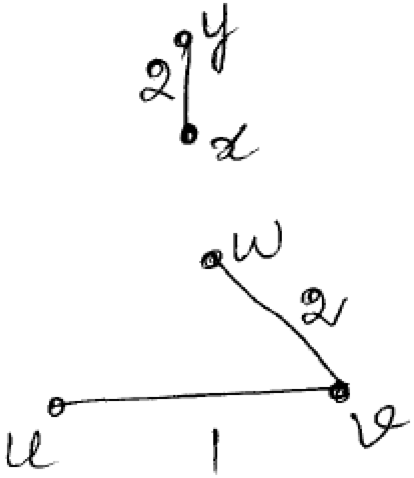
Example

step 2

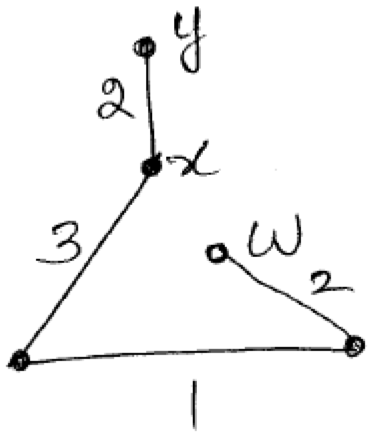


Example

step 3



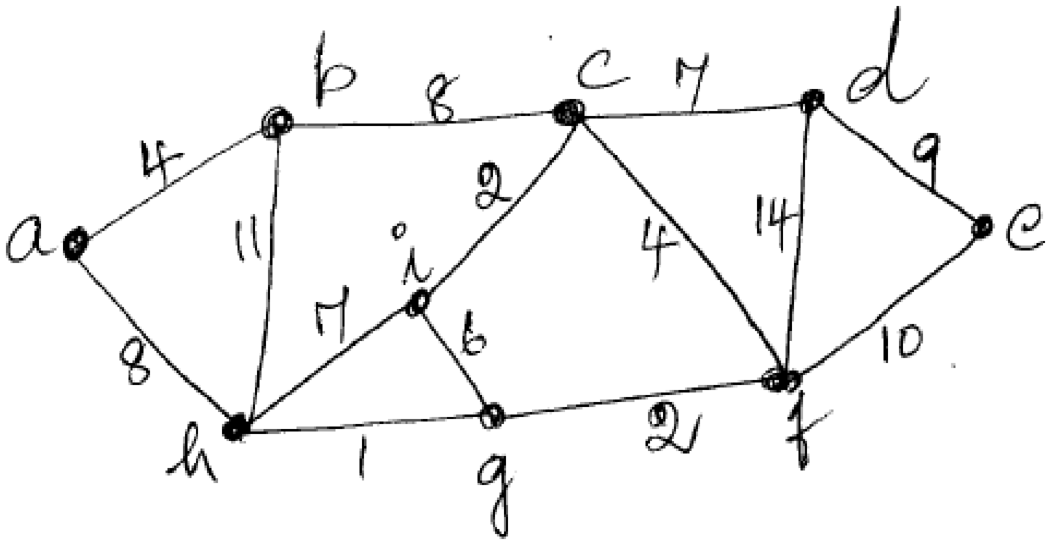
Step 4



is the minimum spanning tree of weight 8.

Example

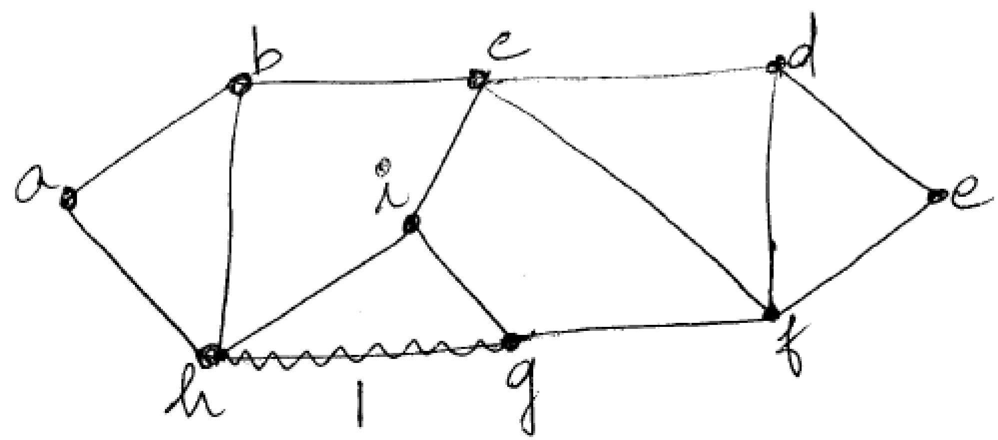
<u>Example</u> Find the minimum spanning tree of the graph G.



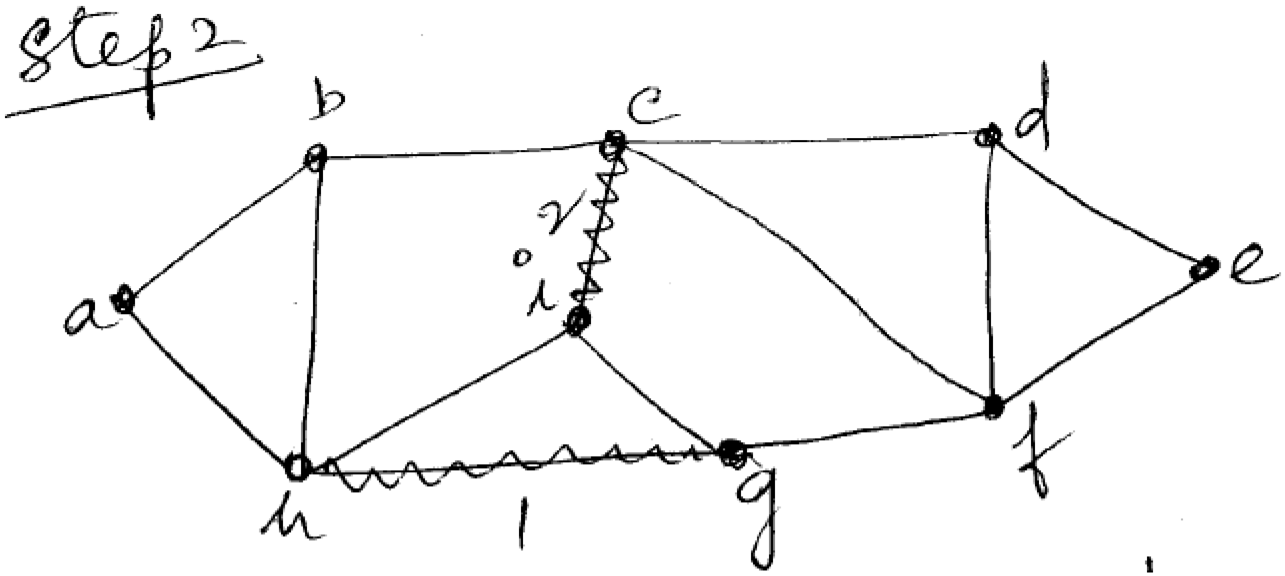
Example

Step 1

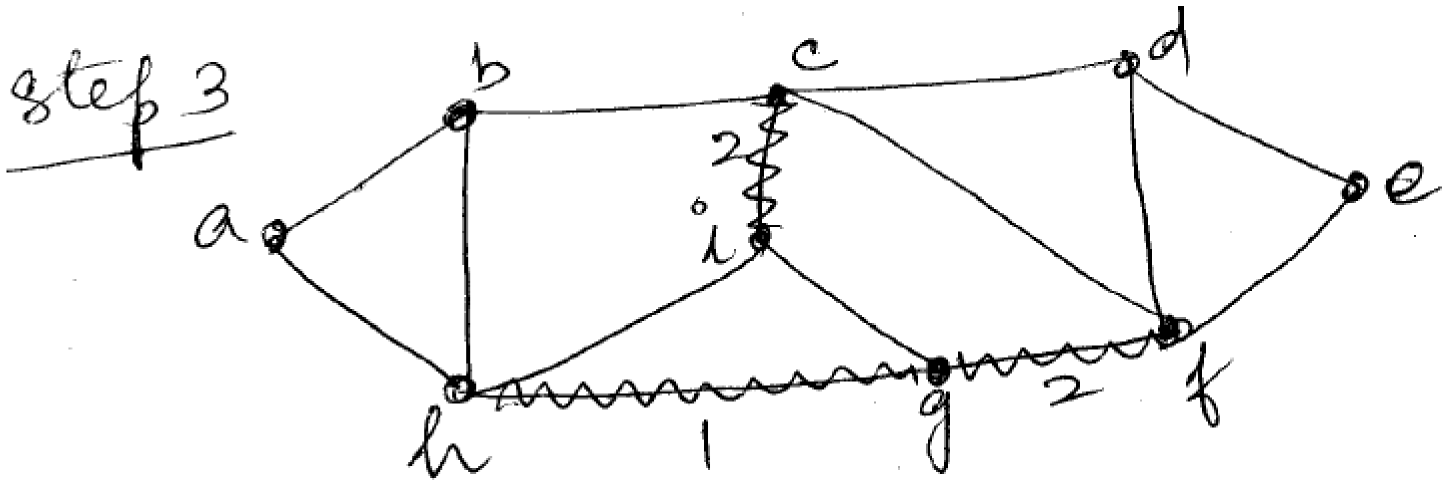
choose hg which is of least weight



Example

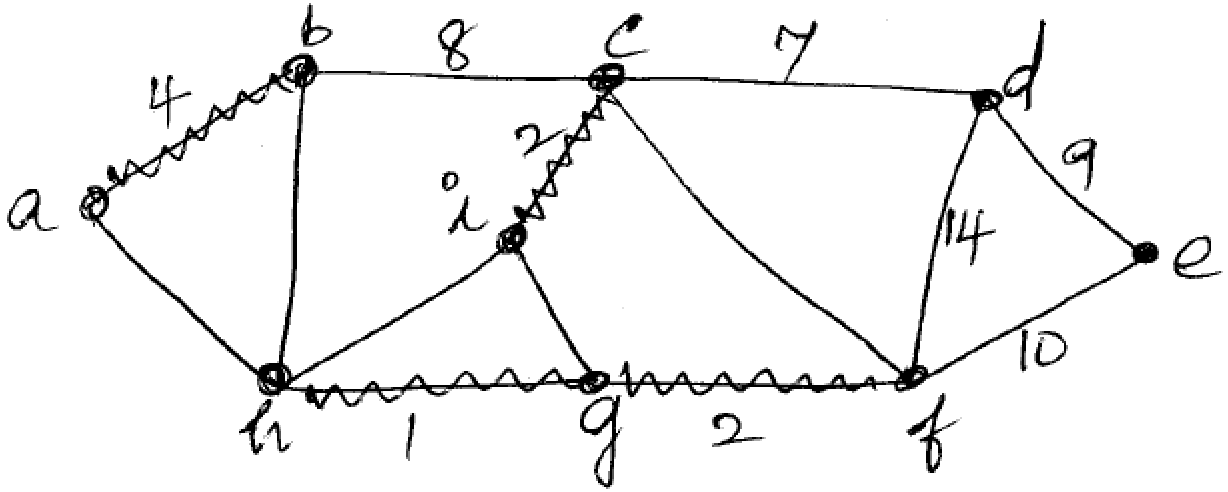


Example



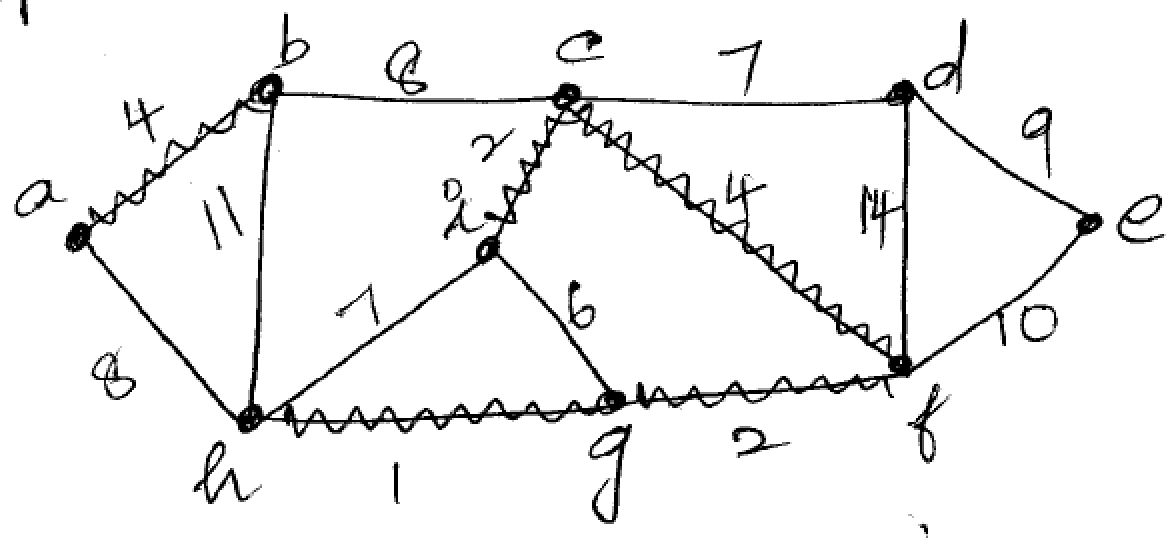
Example

step 4



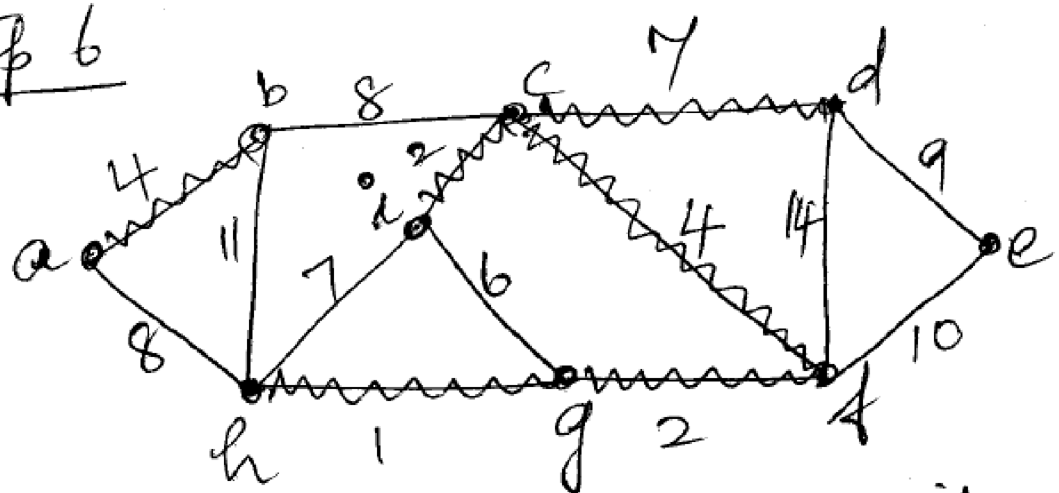
Example

Step 5



Example

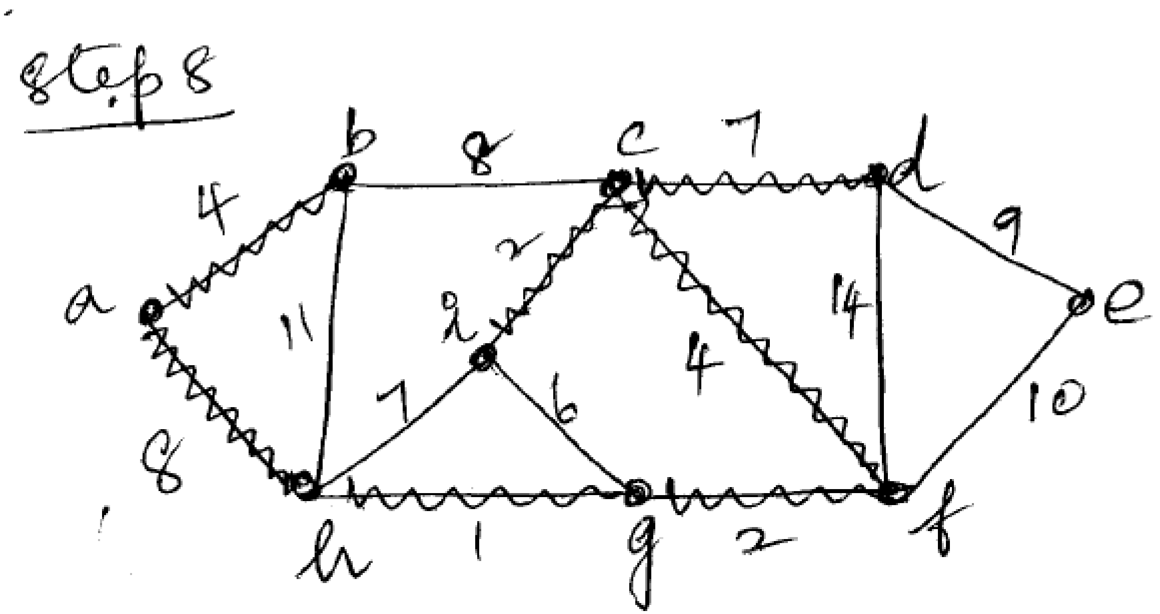
Step 6



ig is not chosen since it forms a cycle with the edges chosen.

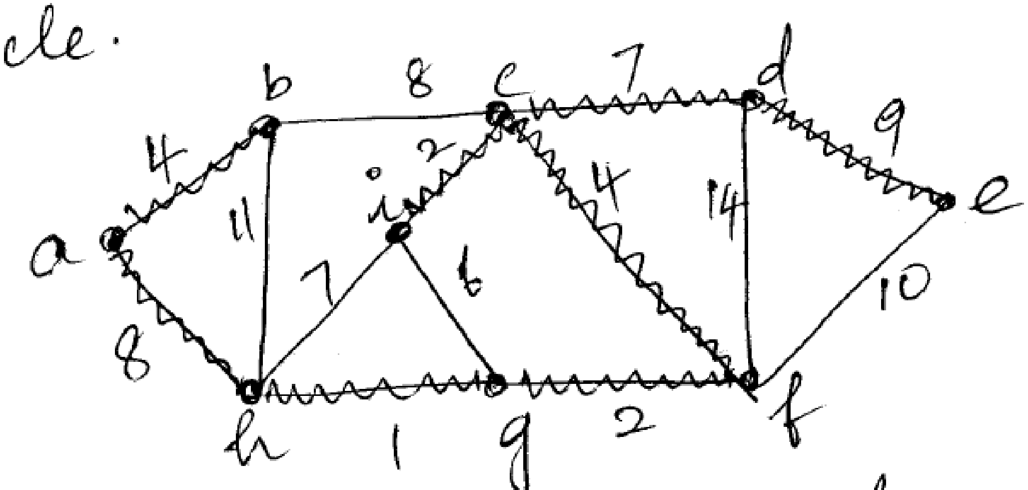
Step 7 cd is chosen. But hi is not chosen since a cycle will be formed.

Example



Example

step 9   bc is not chosen since it forms a cycle.



is the minimum spanning tree

since if the other edges like ef, df or bh is chosen, cycle will be formed.
The weight of the tree is 37.

Greedy Algorithms

•Kruskal's algorithm is a greedy algorithm since it repeatedly selects an edge of minimum weight from the remaining ones, provided no cydle is produced.

•Greedy algorithm is one in which we make the best possible choice at each step, regardless of the subsequent effect of that choice.

Greedy Algorithms

- Choose the best possible solution at every step.

- The choice must be

1. **Feasible** : it has to satisfy the problem's constraints.

2. **Locally Optimal:** it has to be the best local choice among all feasible choices available in that iteration.

3. **Irrevocable:** Once made, the choice cannot be changed on subsequent steps of the algorithm

Prim's Algorithm

- Prim's Algorithm is a minimum spanning tree algorithm.
- To find the MST in a weighted graph.
- This is another example of a greedy algorithm.

Prim's Algorithm

Prim's algorithm for finding a minimal spanning tree in a weighted connected graph G.

To get a minimum spanning tree T starting at a vertex u of G.

(1) Put the vertex u in T.

(2) Now, add edge e of minimum weight which connects a vertex u of T to a vertex which is not in T.

$$T \leftarrow T + e.$$

(3) $|E(T)| = n-1$, output T, otherwise go to step 2.

Prim's Algorithm

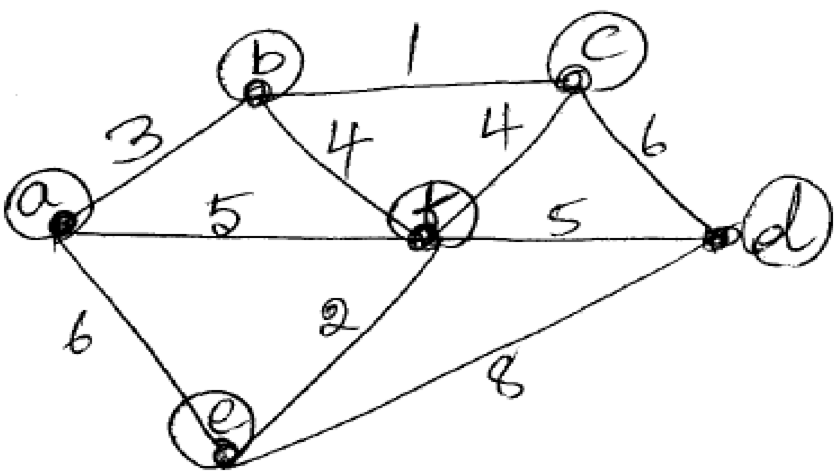
Remark At every stage T is a tree.

Time complexity : $O(|E|\log|V|)$

Example

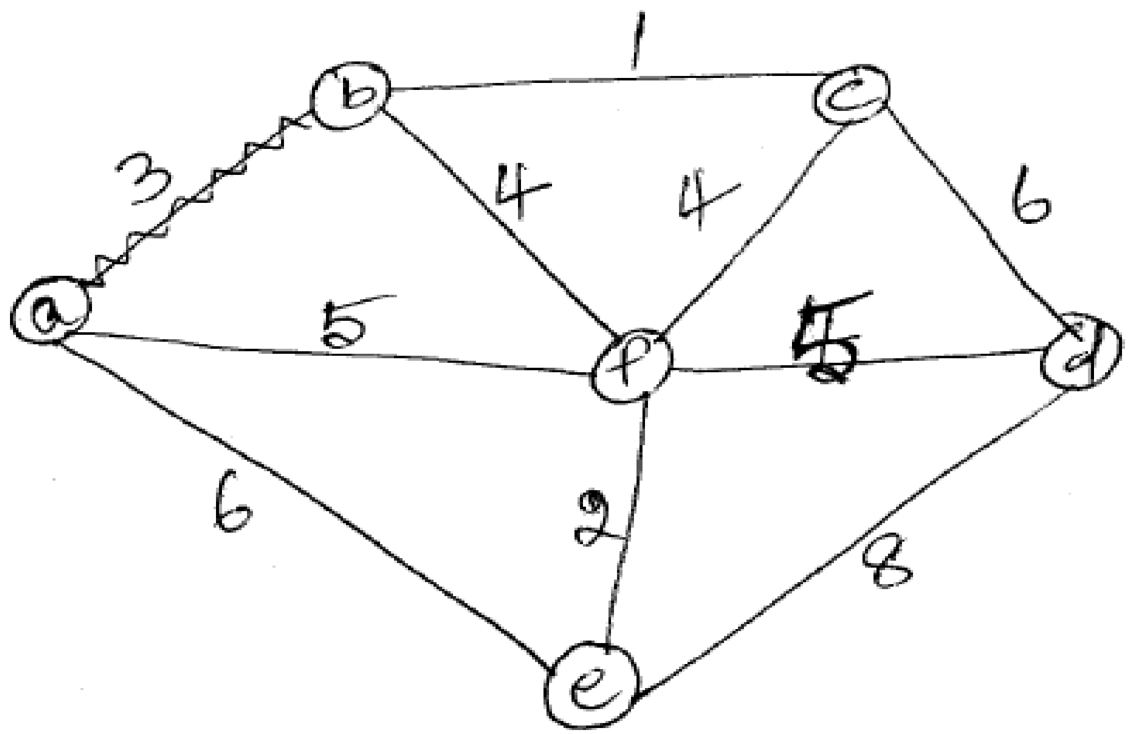
Example : 1

Apply Prim's algorithm to the foll: graph.



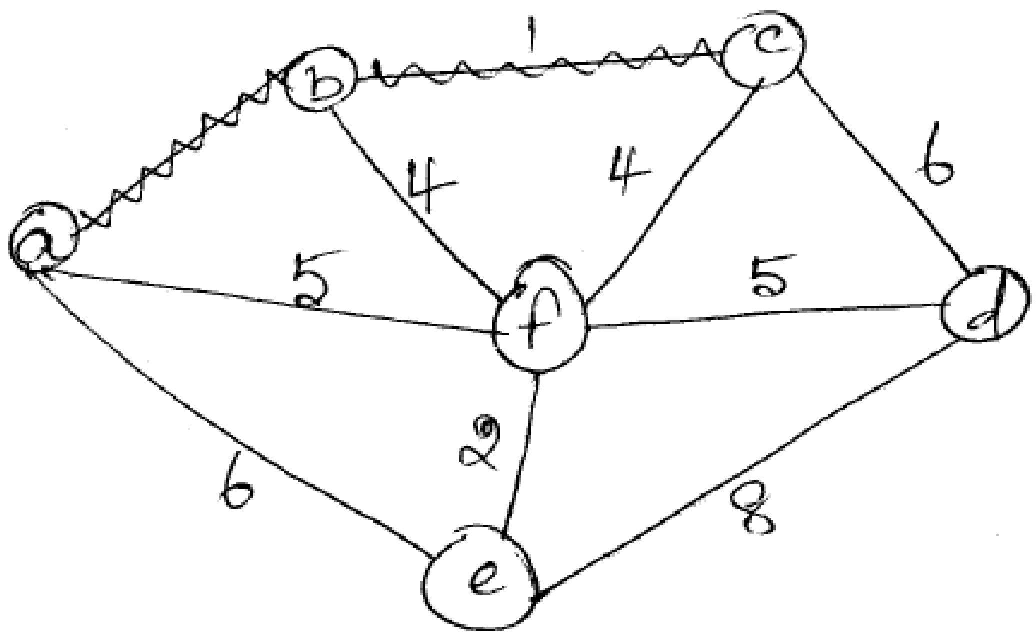
Example

Step 1



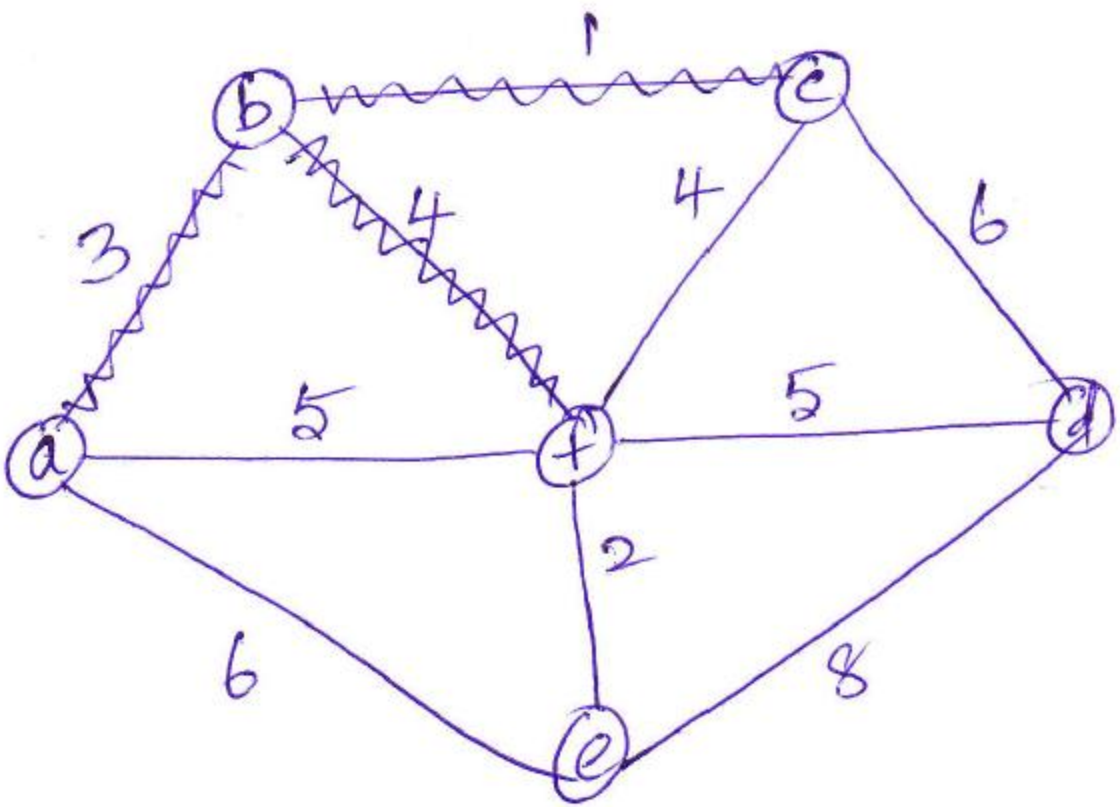
Example

step 2



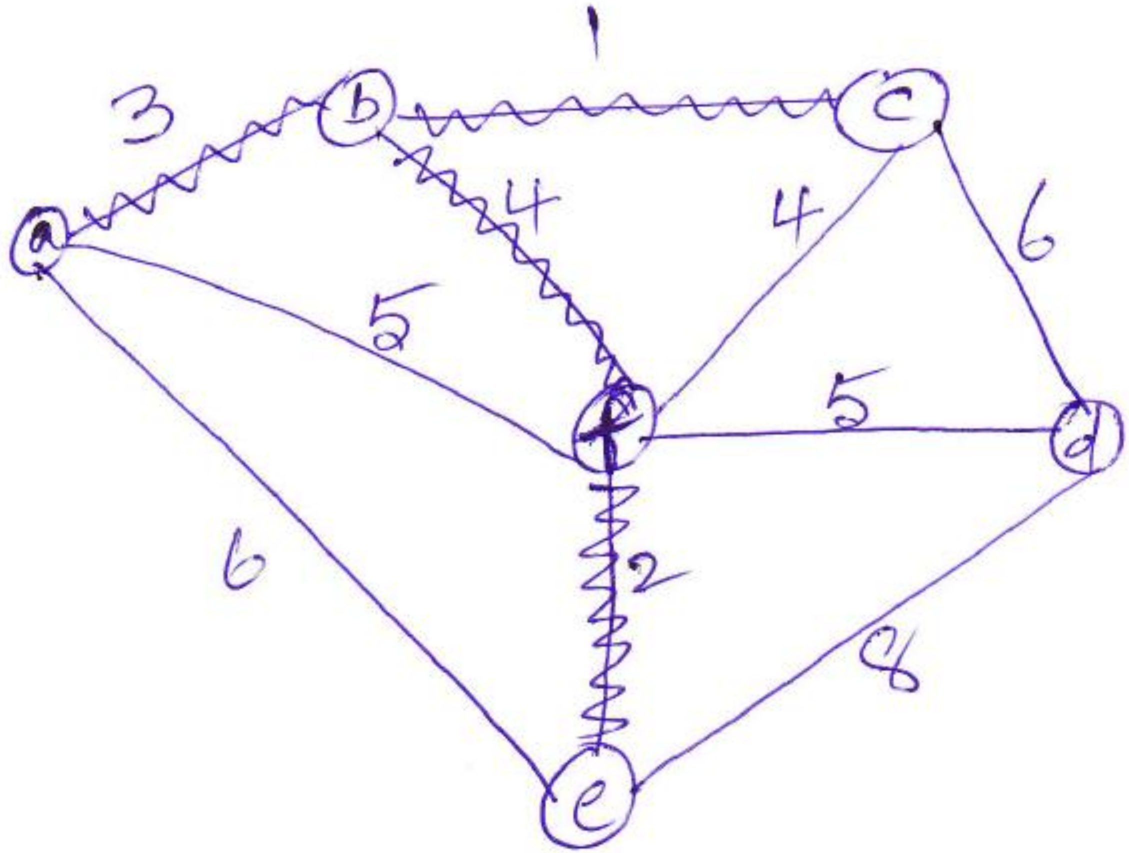
Example

Step 3



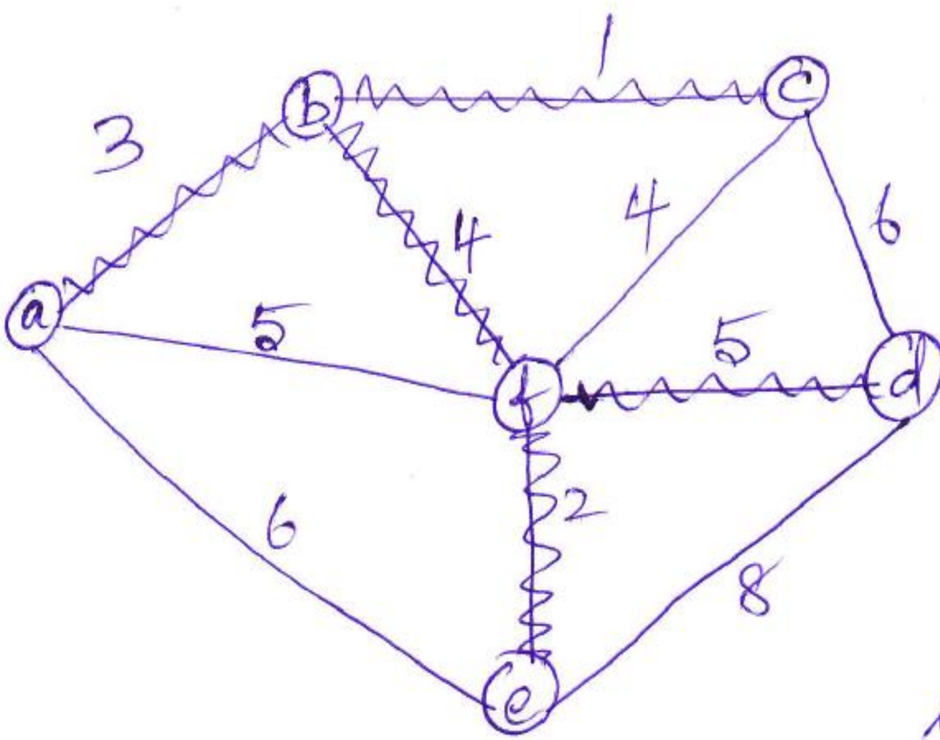
Example

Step 4



Example

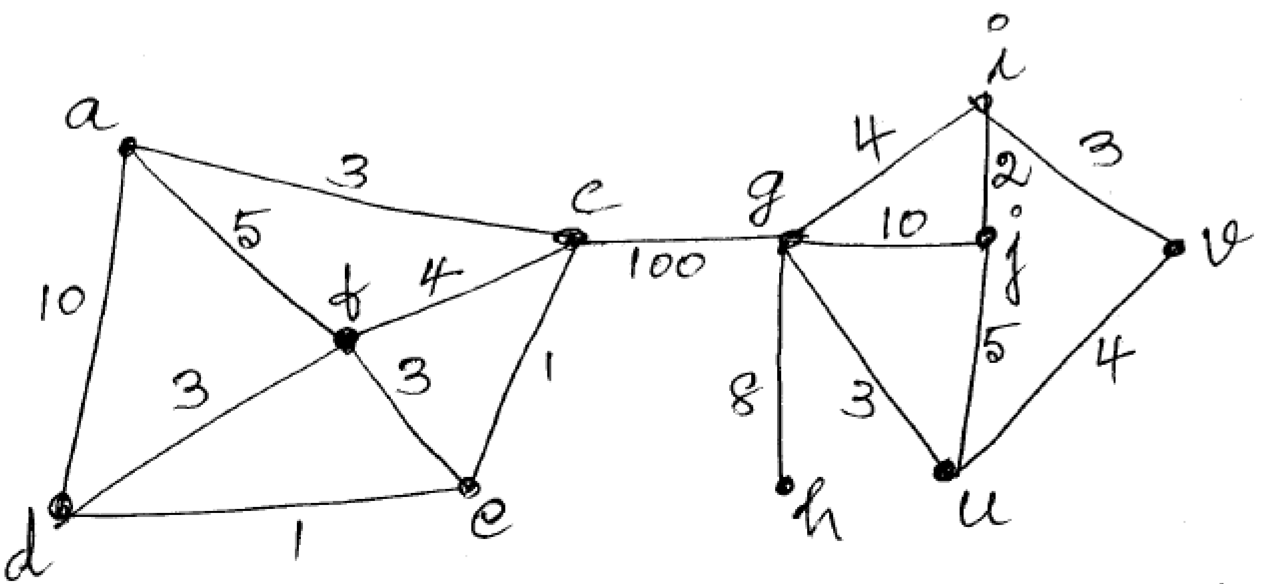
Step 5



MST:
of weight
15.

Example

Example - 2



Find a minimum spanning tree T starting at vertex j.

Example

Iteration	Edge set (T)	Weight of Tree T
I	{ji}	2
II	{ji, iv}	2+3=5
III	{ji, iv, vu}	5+4=9.
IV	{ji, iv, vu, ug}	12
V	{ji, iv, vu, ug, gh}	20
VI	{ji, iv, vu, ug, gh, gc}	120
VII	{ji, iv, vu, ug, gh, gc, ce}	121

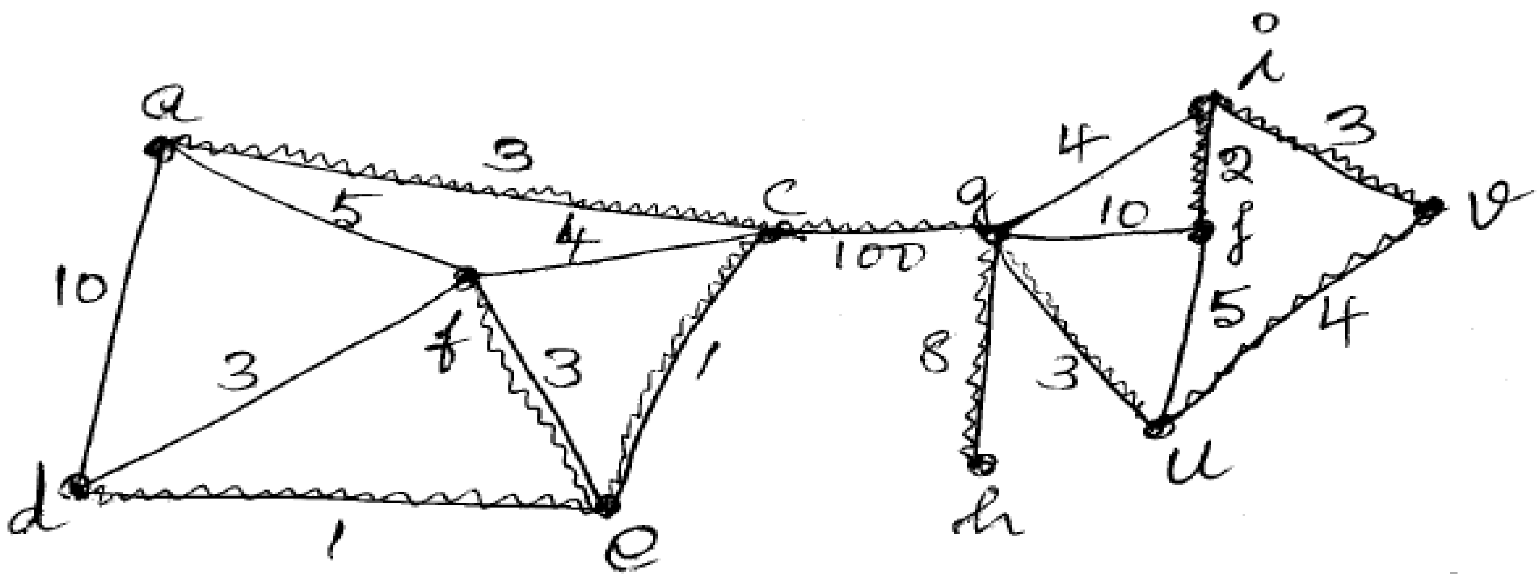
Example

VIII {ji, iv, vu, ug, gh, gc, ce, ed} 122

IX {ji, iv, vu, ug, gh, gc, ce, ed, ef} 125

X {ji, iv, vu, ug, gh, gc, ce, ed, ef, ca} 128

Example



Minimum spanning tree T.