Graph Traversal

Graph traversal is a technique used for a searching vertex in a graph. The graph traversal is also used to decide the order of vertices is visited in the search process. A graph traversal finds the edges to be used in the search process without creating loops. That means using graph traversal we visit all the vertices of the graph without getting into looping path.

There are two graph traversal techniques.

1. DFS (Depth First Search)
2. BFS (Breadth First Search)

DFS (Depth First Search)

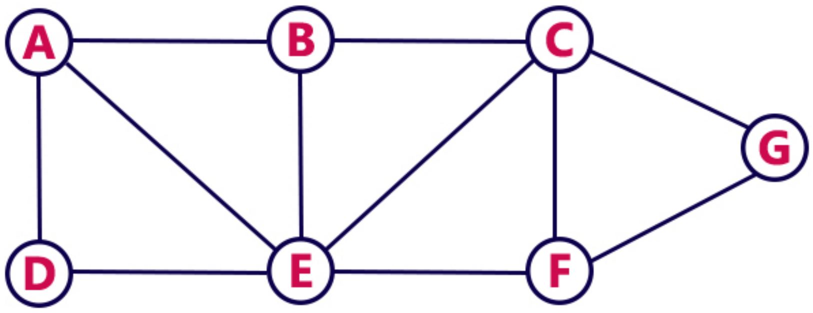
DFS traversal of a graph produces a **spanning tree** as final result. **Spanning Tree** is a graph without loops. We use **Stack data structure** with maximum size of total number of vertices in the graph to implement DFS traversal.
Steps to implement DFS traversal...

- **Step 1 -** Define a Stack of size total number of vertices in the graph.
- **Step 2 -** Select any vertex as **starting point** for traversal. Visit that vertex and push it on to the Stack.
- **Step 3 -** Visit any one of the non-visited **adjacent** vertices of a vertex which is at the top of stack and push it on to the stack.
- **Step 4 -** Repeat step 3 until there is no new vertex to be visited from the vertex which is at the top of the stack.
- **Step 5 -** When there is no new vertex to visit then use **back tracking** and pop one vertex from the stack.
- **Step 6 -** Repeat steps 3, 4 and 5 until stack becomes Empty.
- **Step 7 -** When stack becomes Empty, then produce final spanning tree by removing unused edges from the graph

Back tracking is coming back to the vertex from which we reached the current vertex.

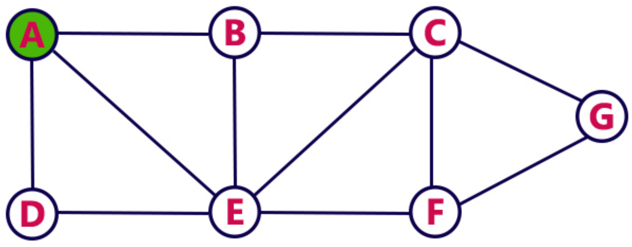
Example

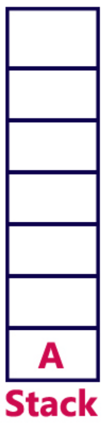
Consider the following example graph to perform DFS traversal



Step 1:

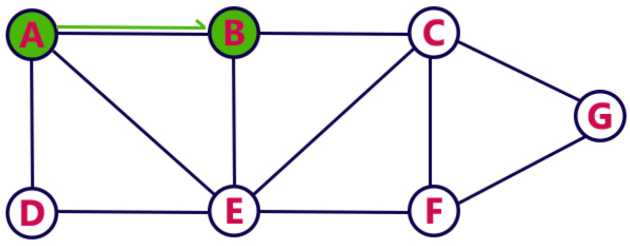
- Select the vertex **A** as starting point (visit **A**).
- Push **A** on to the Stack.

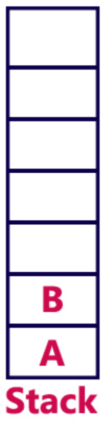




Step 2:

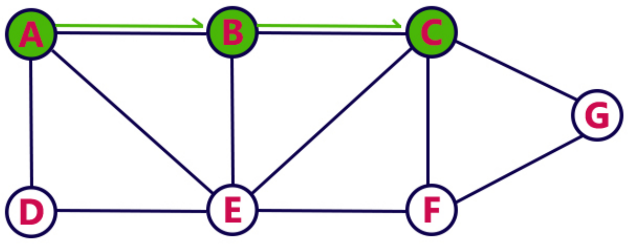
- Visit any adjacent vertex of **A** which is not visited (**B**).
- Push newly visited vertex B on to the Stack.

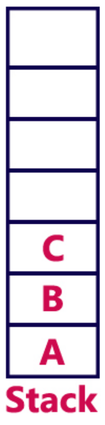




Step 3:

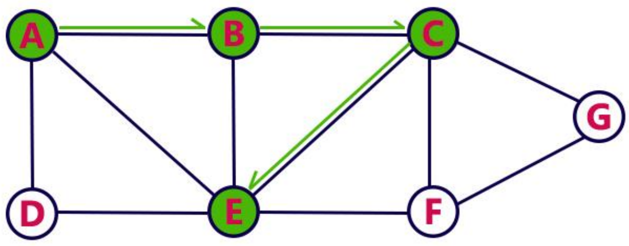
- Visit any adjacent vertext of **B** which is not visited (**C**).
- Push C on to the Stack.

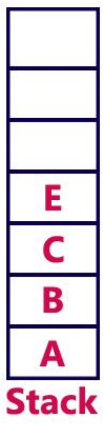




Step 4:

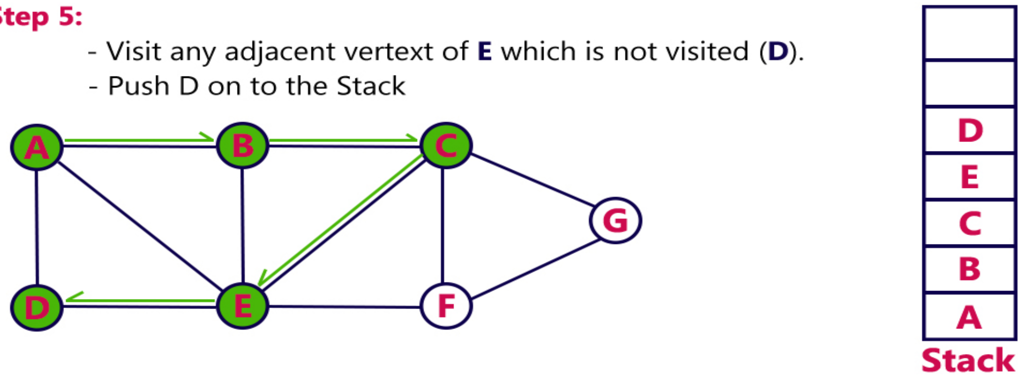
- Visit any adjacent vertext of **C** which is not visited (**E**).
- Push E on to the Stack





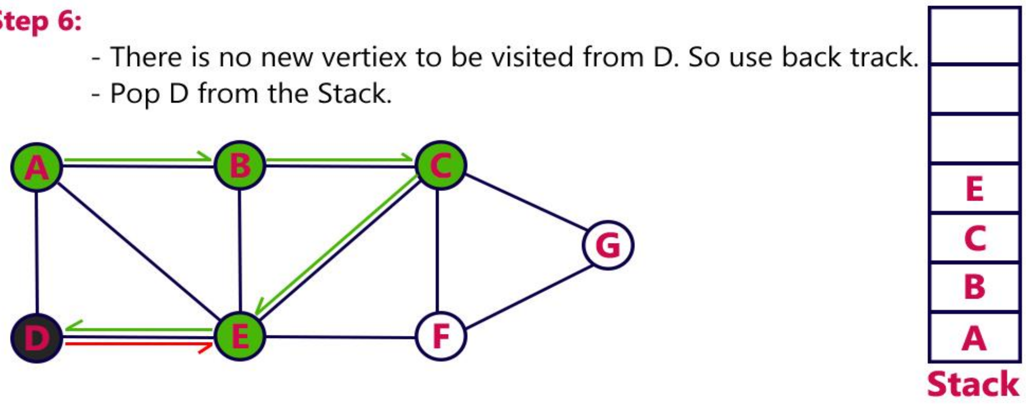
Step 5:

- Visit any adjacent vertext of **E** which is not visited (**D**).
- Push D on to the Stack



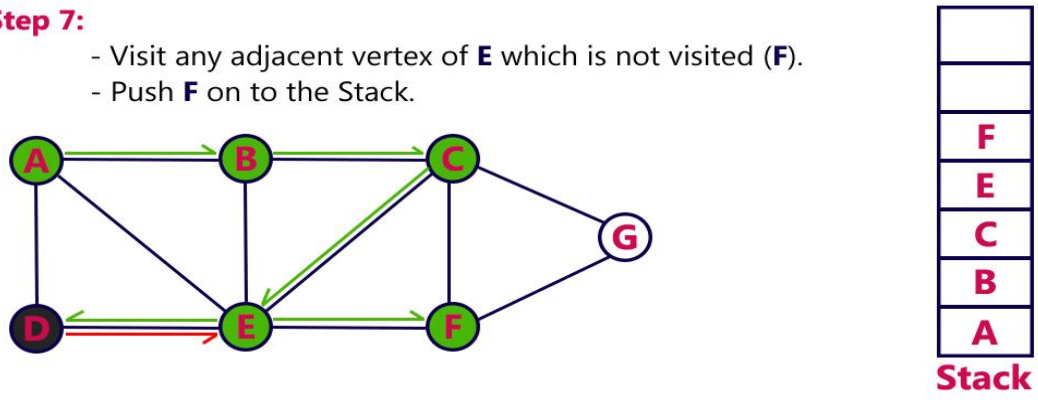
Step 6:

- There is no new vertiex to be visited from D. So use back track.
- Pop D from the Stack.



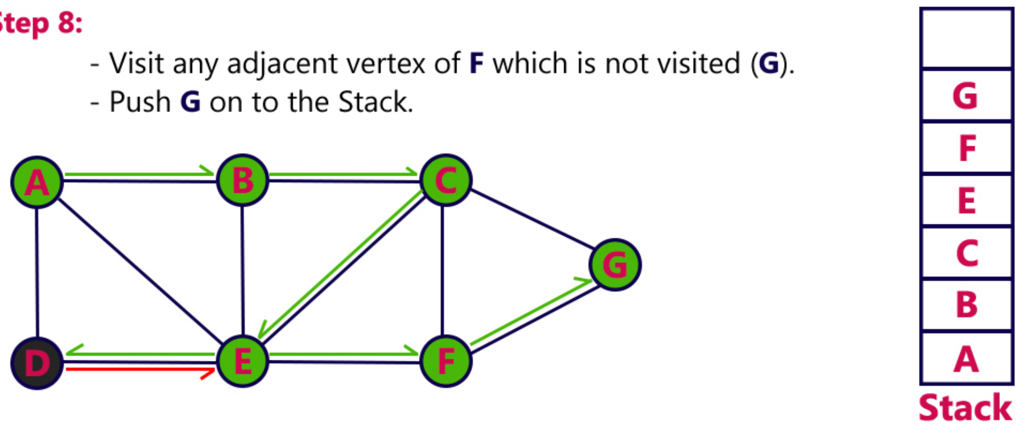
Step 7:

- Visit any adjacent vertex of **E** which is not visited (**F**).
- Push **F** on to the Stack.



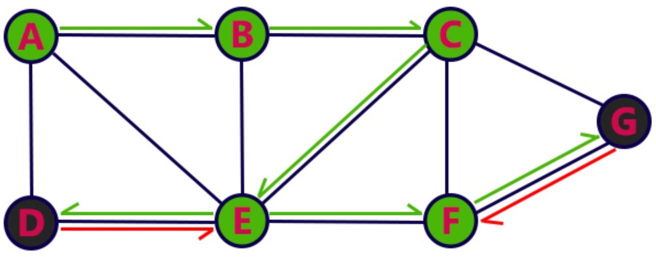
Step 8:

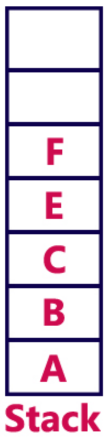
- Visit any adjacent vertex of **F** which is not visited (**G**).
- Push **G** on to the Stack.



Step 9:

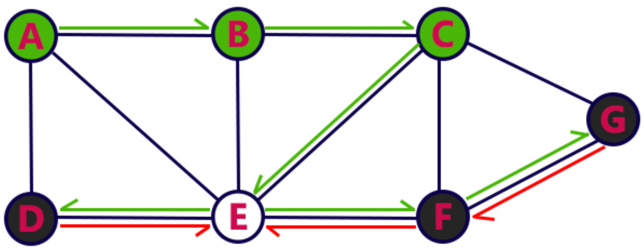
- There is no new vertiex to be visited from G. So use back track.
- Pop G from the Stack.

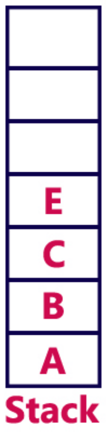




Step 10:

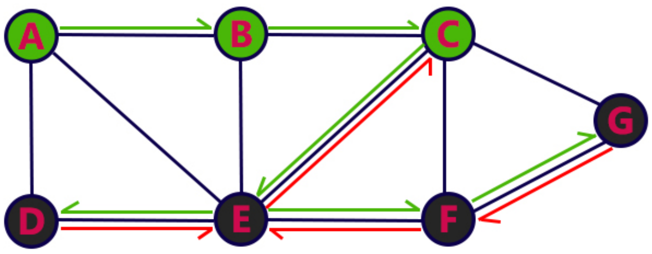
- There is no new vertiex to be visited from F. So use back track.
- Pop F from the Stack.

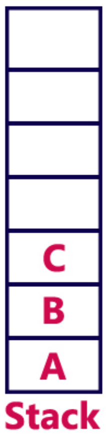




Step 11:

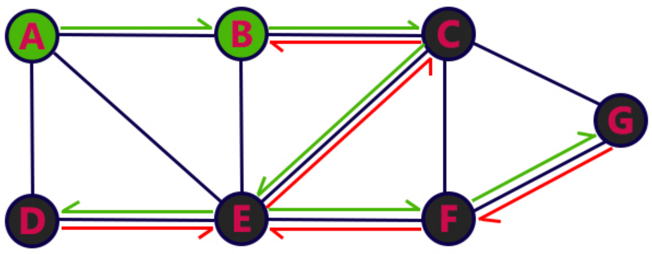
- There is no new vertiex to be visited from E. So use back track.
- Pop E from the Stack.

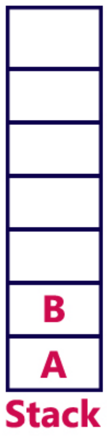




Step 12:

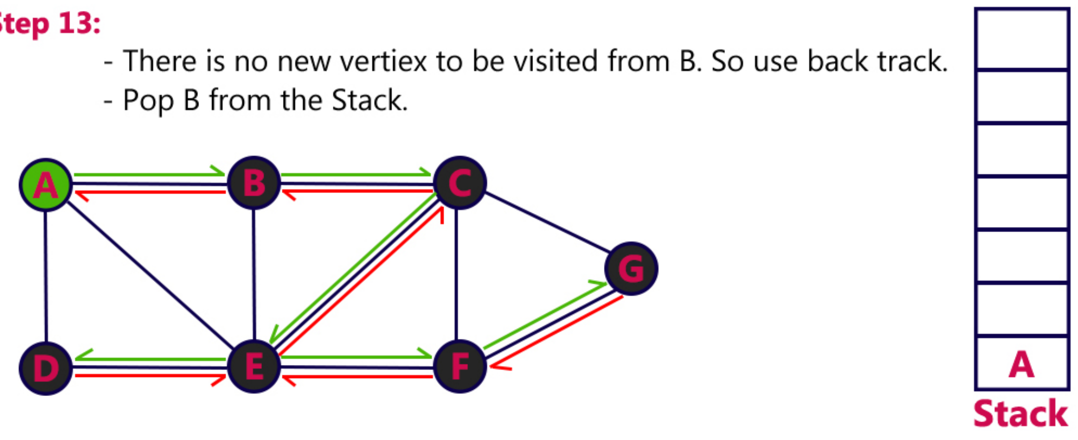
- There is no new vertiex to be visited from C. So use back track.
- Pop C from the Stack.





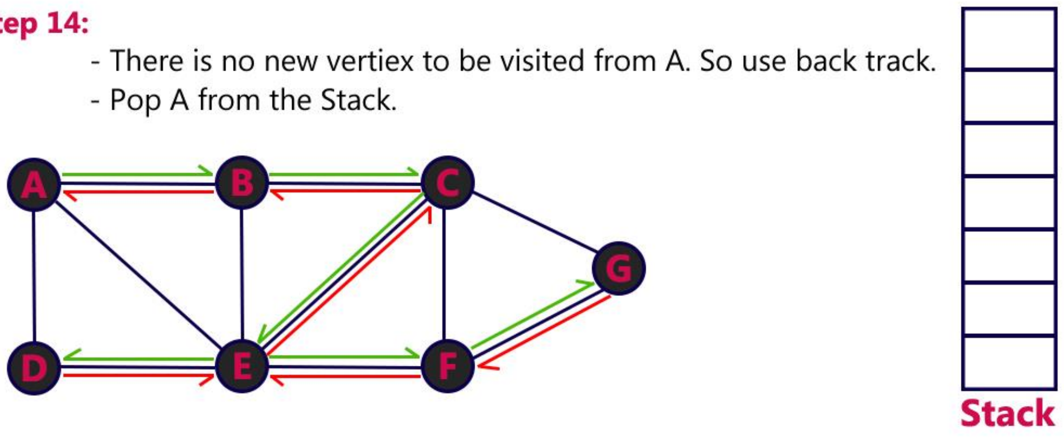
Step 13:

- There is no new vertiex to be visited from B. So use back track.
- Pop B from the Stack.

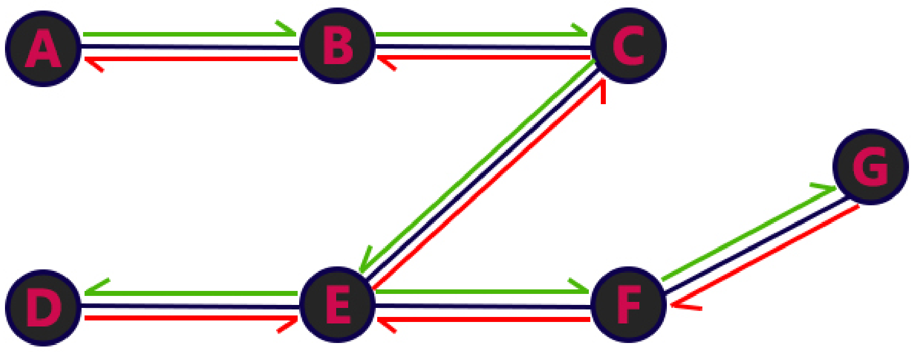


Step 14:

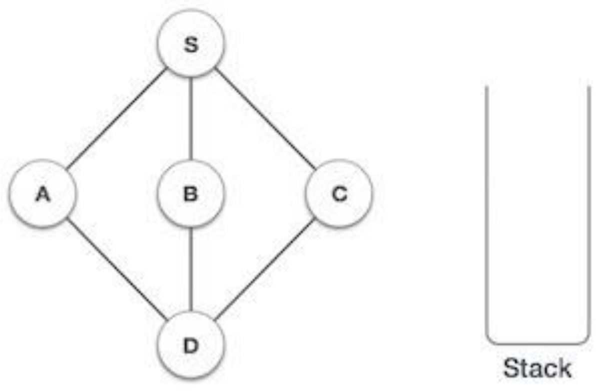
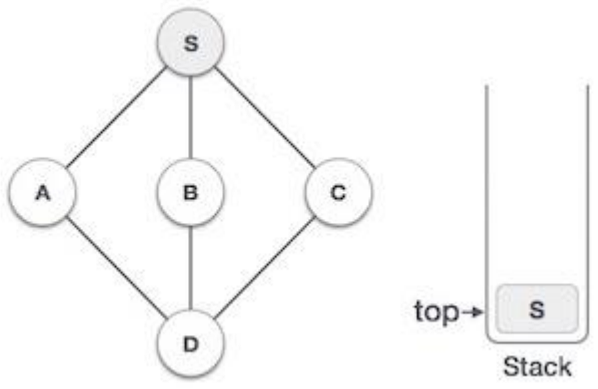
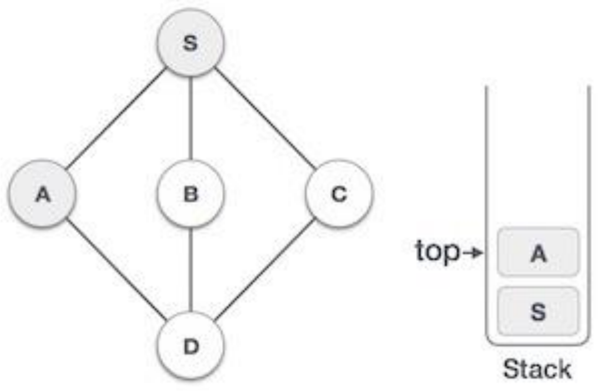
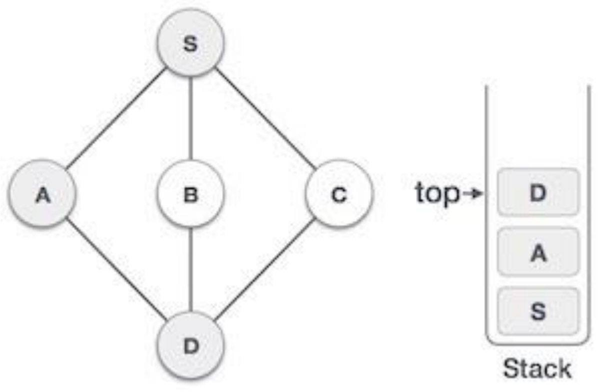
- There is no new vertiex to be visited from A. So use back track.
- Pop A from the Stack.

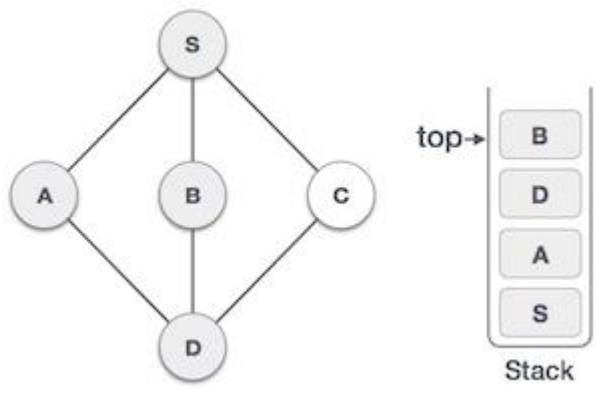
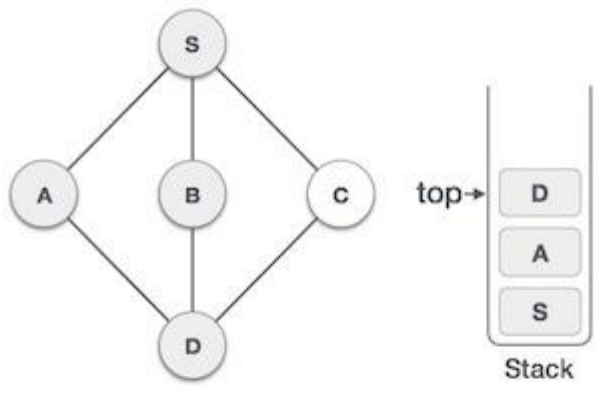
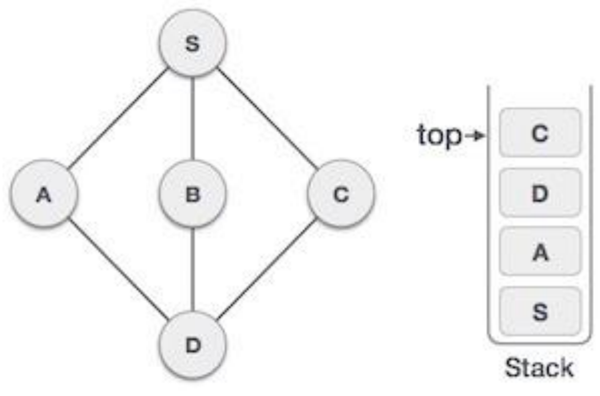


- Stack became Empty. So stop DFS Treversal.
- Final result of DFS traversal is following spanning tree.



Example 2

Step	Traversal	Description
1		Initialize the stack.
2		Mark **S** as visited and put it onto the stack. Explore any unvisited adjacent node from **S**. We have three nodes and we can pick any of them. For this example, we shall take the node in an alphabetical order.
3		Mark **A** as visited and put it onto the stack. Explore any unvisited adjacent node from A. Both **S** and **D** are adjacent to **A** but we are concerned for unvisited nodes only.
4		Visit **D** and mark it as visited and put onto the stack. Here, we have **B** and **C** nodes, which are adjacent to **D** and both are unvisited. However, we shall again choose in an alphabetical order.

5		We choose **B**, mark it as visited and put onto the stack. Here **B** does not have any unvisited adjacent node. So, we pop **B** from the stack.
6		We check the stack top for return to the previous node and check if it has any unvisited nodes. Here, we find **D** to be on the top of the stack.
7		Only unvisited adjacent node is from **D** is **C** now. So we visit **C**, mark it as visited and put it onto the stack.

As **C** does not have any unvisited adjacent node so we keep popping the stack until we find a node that has an unvisited adjacent node. In this case, there's none and we keep popping until the stack is empty.

BFS (Breadth First Search)

BFS traversal of a graph produces a **spanning tree** as final result. We use **Queue data structure** with maximum size of total number of vertices in the graph to implement BFS traversal.

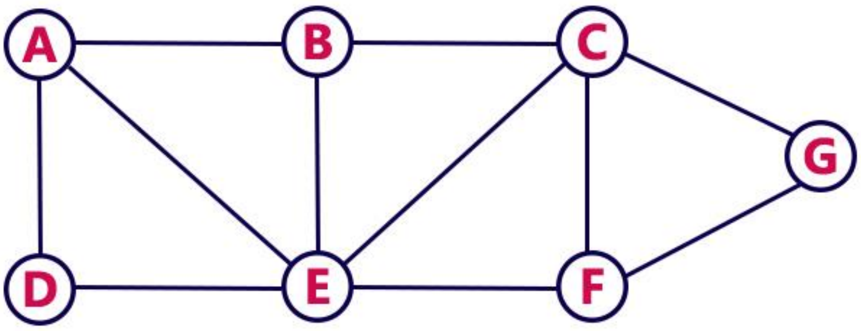
Steps to implement BFS traversal...

- **Step 1 -** Define a Queue of size total number of vertices in the graph.
- **Step 2 -** Select any vertex as **starting point** for traversal. Visit that vertex and insert it into the Queue.
- **Step 3 -** Visit all the non-visited **adjacent** vertices of the vertex which is at front of the Queue and insert them into the Queue.

- Step 4 - When there is no new vertex to be visited from the vertex which is at front of the Queue then delete that vertex.
- Step 5 - Repeat steps 3 and 4 until queue becomes empty.
- Step 6 - When queue becomes empty, then produce final spanning tree by removing unused edges from the graph

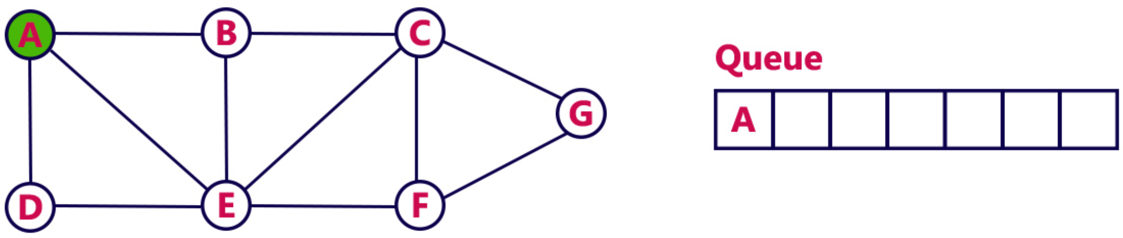
Example

Consider the following example graph to perform BFS traversal



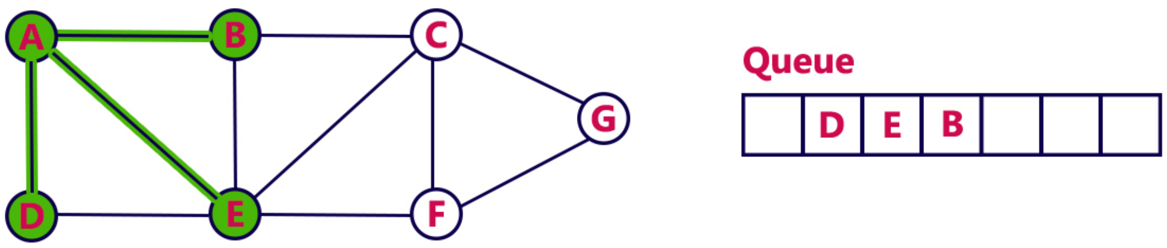
Step 1:

- Select the vertex **A** as starting point (visit **A**).
- Insert **A** into the Queue.



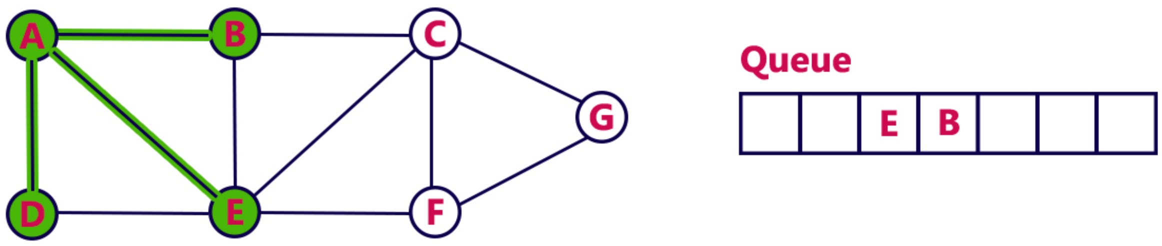
Step 2:

- Visit all adjacent vertices of **A** which are not visited (**D**, **E**, **B**).
- Insert newly visited vertices into the Queue and delete A from the Queue..



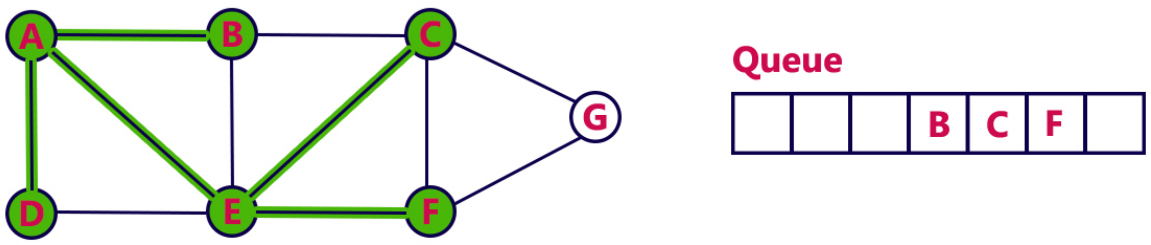
Step 3:

- Visit all adjacent vertices of **D** which are not visited (there is no vertex).
- Delete D from the Queue.



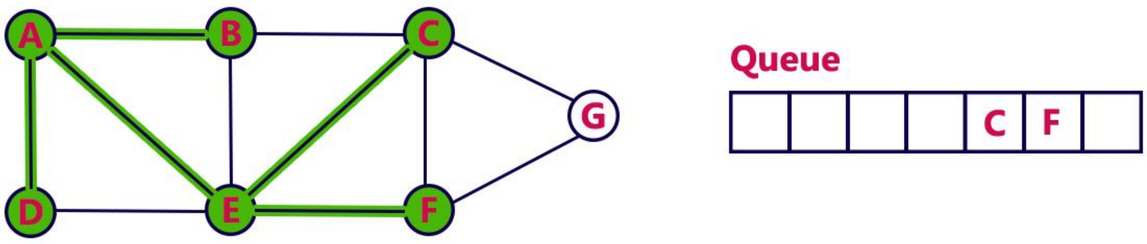
Step 4:

- Visit all adjacent vertices of **E** which are not visited (**C**, **F**).
- Insert newly visited vertices into the Queue and delete E from the Queue.



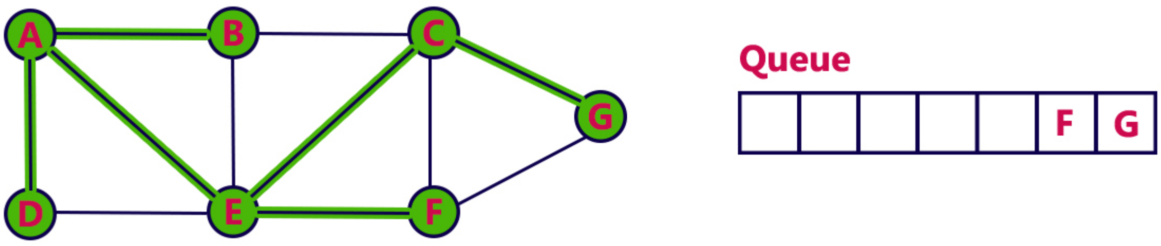
Step 5:

- Visit all adjacent vertices of **B** which are not visited (**there is no vertex**).
- Delete **B** from the Queue.



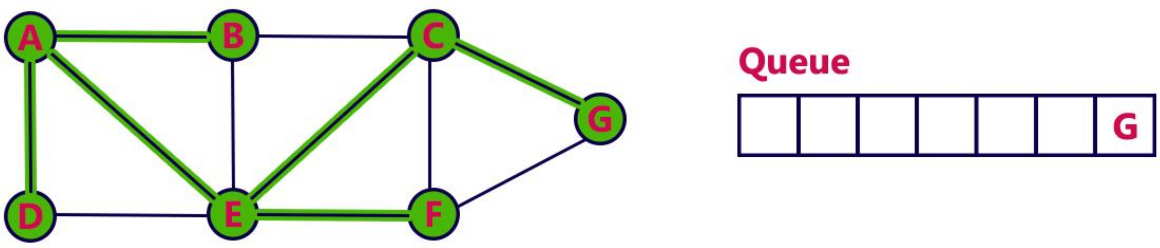
Step 6:

- Visit all adjacent vertices of **C** which are not visited (**G**).
- Insert newly visited vertex into the Queue and delete **C** from the Queue.



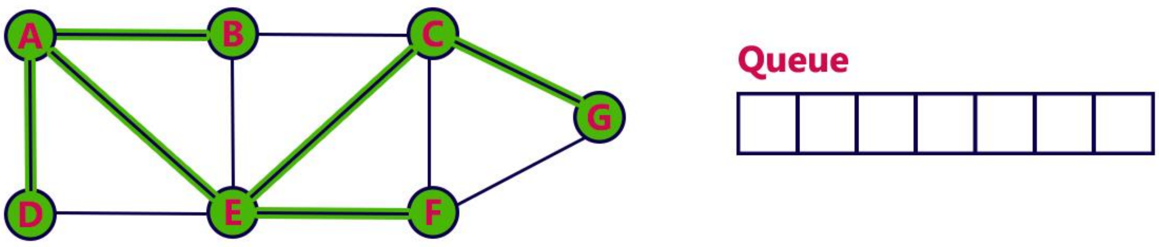
Step 7:

- Visit all adjacent vertices of **F** which are not visited (**there is no vertex**).
- Delete **F** from the Queue.

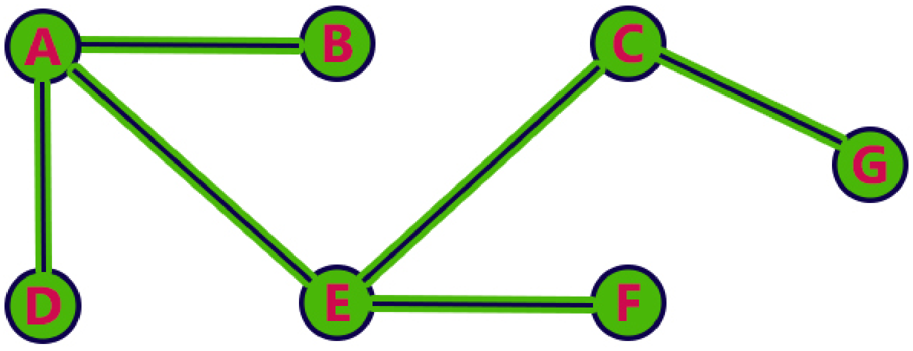


Step 8:

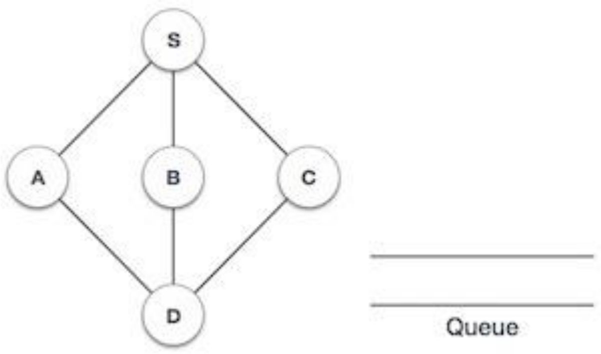
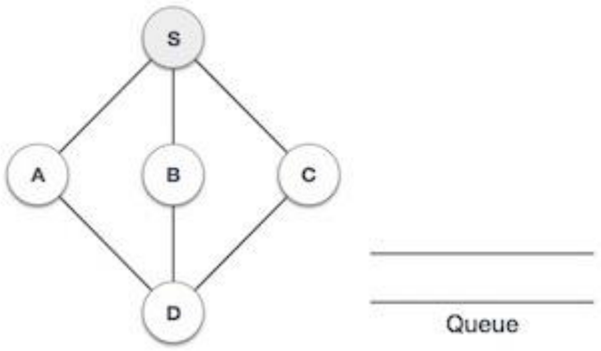
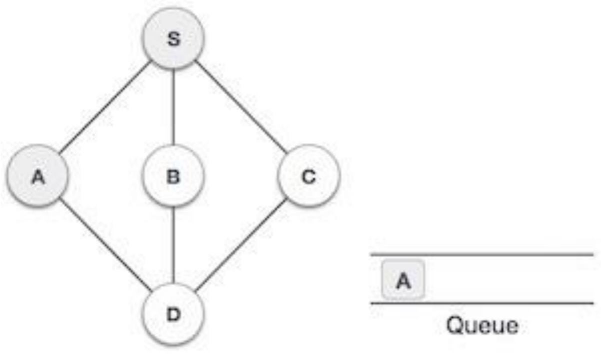
- Visit all adjacent vertices of **G** which are not visited (**there is no vertex**).
- Delete **G** from the Queue.

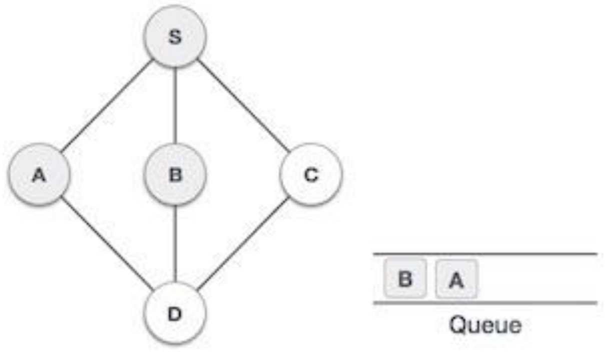
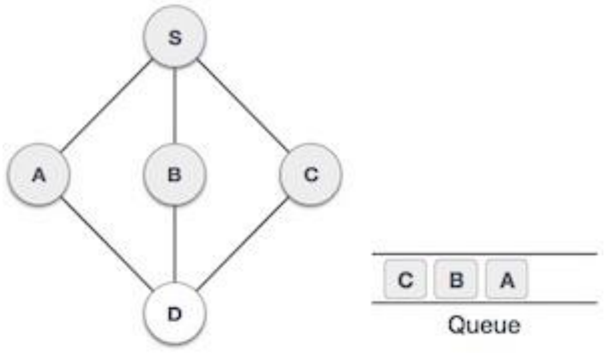
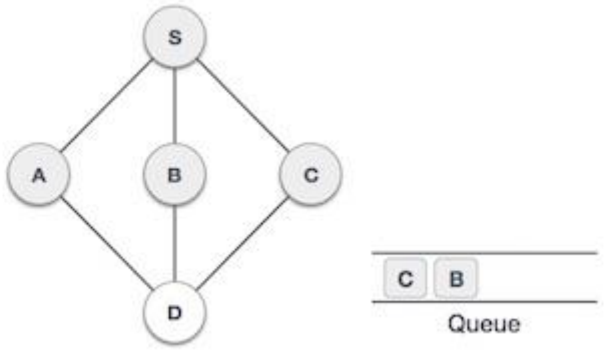
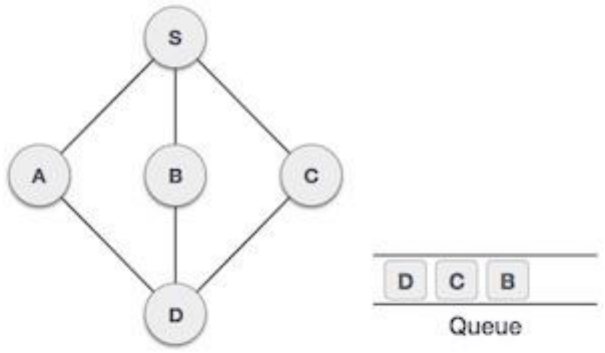


- Queue became Empty. So, stop the BFS process.
- Final result of BFS is a Spanning Tree as shown below...



Example 2

Step	Traversal	Description
1		Initialize the queue.
2		We start from visiting **S** (starting node), and mark it as visited.
3		We then see an unvisited adjacent node from **S**. In this example, we have three nodes but alphabetically we choose **A**, mark it as visited and enqueue it.

4		Next, the unvisited adjacent node from **S** is **B**. We mark it as visited and enqueue it.
5		Next, the unvisited adjacent node from **S** is **C**. We mark it as visited and enqueue it.
6		Now, **S** is left with no unvisited adjacent nodes. So, we dequeue and find **A**.
7		From **A** we have **D** as unvisited adjacent node. We mark it as visited and enqueue it.

At this stage, we are left with no unmarked (unvisited) nodes. But as per the algorithm we keep on dequeuing in order to get all unvisited nodes. When the queue gets emptied, the program is over.

```
/*graph traversals */
#include<stdio.h>

int q[20],top=-1,front=-1,rear=-1,a[20][20],vis[20],stack[20];
int delete();
void add(int item);
void bfs(int s,int n);
void dfs(int s,int n);
void push(int item);
int pop();

void main()
{
 int n,i,s,ch,j;
 char c,dummy;
 printf("ENTER THE NUMBER VERTICES ");
 scanf("%d",&n);
 for(i=1;i<=n;i++)
 {
  for(j=1;j<=n;j++)
  {
   printf("ENTER 1 IF %d HAS A NODE WITH %d ELSE 0 ",i,j);
   scanf("%d",&a[i][j]);
  }
 }
 printf("THE ADJACENCY MATRIX IS\n");
 for(i=1;i<=n;i++)
 {
  for(j=1;j<=n;j++)
  {
   printf(" %d",a[i][j]);
  }
  printf("\n");
 }

 do
 {
   for(i=1;i<=n;i++)
     vis[i]=0;
   printf("\nMENU");
   printf("\n1.B.F.S");
   printf("\n2.D.F.S");
   printf("\nENTER YOUR CHOICE");
   scanf("%d",&ch);
   printf("ENTER THE SOURCE VERTEX :");
   scanf("%d",&s);

 switch(ch)
 {
   case 1:
     bfs(s,n);
     break;
   case 2:
     dfs(s,n);
     break;
 }
 printf("DO U WANT TO CONTINUE(Y/N) ? ");
 scanf("%c",&dummy);
 scanf("%c",&c);
 }while((c=='y')||(c=='Y'));
}


void bfs(int s,int n)
{
 int p,i;
 add(s);
 vis[s]=1;
 p=delete();
 if(p!=0)
  printf(" %d",p);
 while(p!=0)
 {
 for(i=1;i<=n;i++)
  if((a[p][i]!=0)&&(vis[i]==0))
  {
   add(i);
   vis[i]=1;
  }
  p=delete();
  if(p!=0)
    printf(" %d ",p);
 }
 for(i=1;i<=n;i++)
  if(vis[i]==0)
    bfs(i,n);
}


void add(int item)
{
 if(rear==19)
  printf("QUEUE FULL");
 else
 {
  if(rear==-1)
  {
    q[++rear]=item;
    front++;
  }
  else
    q[++rear]=item;
 }
}

int delete()
{
 int k;
 if((front>rear)||(front==-1))
   return(0);
 else
 {
   k=q[front++];
   return(k);
 }
}

void dfs(int s,int n)
{
 int i,k;
 push(s);
 vis[s]=1;
 k=pop();
 if(k!=0)
   printf(" %d ",k);
 while(k!=0)
 {
 for(i=1;i<=n;i++)
  if((a[k][i]!=0)&&(vis[i]==0))
  {
   push(i);
   vis[i]=1;
  }
  k=pop();
  if(k!=0)
    printf(" %d ",k);
 }
 for(i=1;i<=n;i++)
   if(vis[i]==0)
    dfs(i,n);
}

void push(int item)
{
 if(top==19)
   printf("Stack overflow ");
 else
   stack[++top]=item;
}

int pop()
{
 int k;
 if(top==-1)
  return(0);
 else
 {
  k=stack[top--];
  return(k);
 }
}
```