❑ What is BFS ?????

- BFS stands for **B**readth **F**irst **S**earch.
- BFS is an algorithm for traversing or searching a tree or graph data structures.
- It uses a queue data structure for implementation.
- In BFS traversal we visit all the nodes level by level and the traversal completed when all the nodes are visited.

❑ Algorithm for BFS :

Step 1: Initialize all nodes with status=1.(ready state)

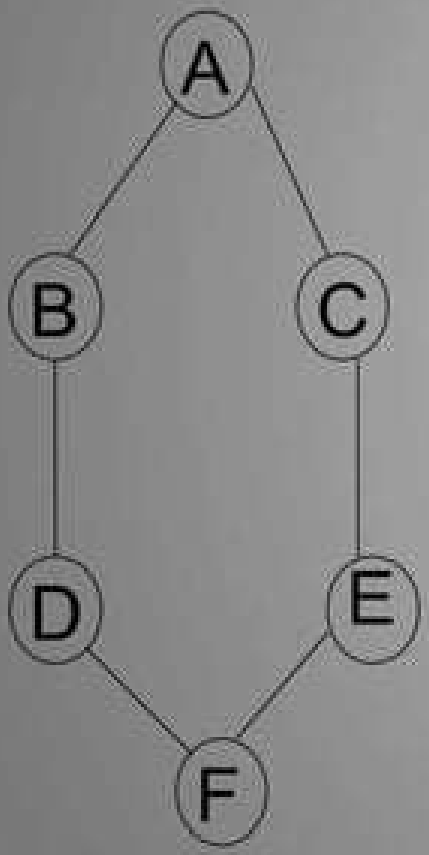
Step 2: Put starting node in a queue and change status to status=2.(waiting state)

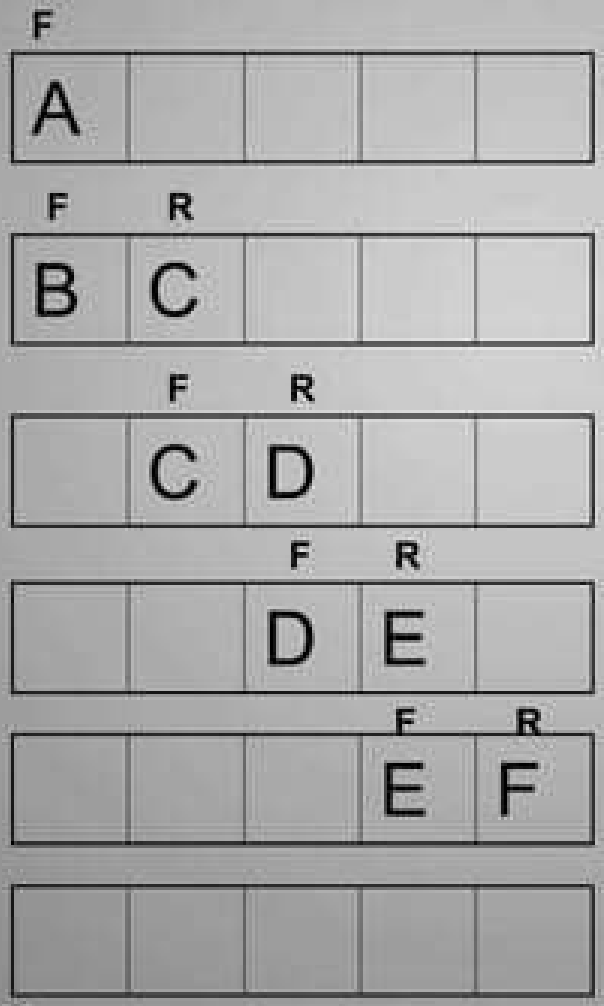
Step 3: loop:
repeat step 4 and step 5 until queue gets empty.

Step 4: Remove front node N from queue, process them and change the status of N to status=3.(processed state)

Step 5: Add all the neighbours of N to the rear of queue and change status to status=2.(waiting status)

❑ Working of BFC :





queue data structure

A

A,B

A,B,C

A,B,C,D

A,B,C,D,E,F

Queue gets empty and the algorithm ends

Code for BFC in C

```
#include<stdio.h>
#include<conio.h>
#define n 8

      int A[8][8]={          {0,1,1,1,0,0,0,0},
                  {1,0,0,0,1,0,0,0},
                  {1,0,0,0,0,1,0,0},
                  {1,0,0,0,0,0,1,0},
                  {0,1,0,0,0,0,0,1},
                  {0,0,1,0,0,0,0,1},
                  {0,0,0,1,0,0,0,1},
                  {0,0,0,0,1,1,1,0}
                              };
int Front=-1;
int Rear=-1;
int Q[n];
int Visit[n];
void enqueue(int);
int dequeue();
void BFS();
```

```
void main()
{
      int i,j,s;
      clrscr();

      printf("\n Adjacency Matrix is : \n");
      for(i=0;i<=n-1;i++);
      {
            for(j=0;j<=n-1;j++)
            {
                  printf(" %d",A[i][j]);
            }
            printf("\n");
      }

      printf("Enter source code : ");
      scanf("%d",&s);
      printf("BFS traversal is : ");
      BFS(s);
       getch();
}
```

```
void BFS(int s)
{
      int i,p;
      enqueue(s);
      Visit[s]=1;
      loop:
      p=dequeue();
      if(p!=-1)
      {
      printf(" %d",p);

      for(i=0;i<=n-1;i++)
      {
              if(A[p][i]==1 && Visit[i]==0)
        {
      enqueue(i);
          Visit[i]=1;
          }
      }
      goto loop;
      }
}
```

```
void enqueue(int s)
{
      if(Rear==n-1)
              printf("Queue is overflow");

      else
      {
              Rear++;
              Q[Rear]=s;
              if(Front==-1)
              {
                      Front=0;
              }
      }
}
```

```
int dequeue()
{
	int item;
	if(Front==-1)
	{
	return -1;
	}
	else
	{
		item=Q[Front];
		if(Front==Rear)
		{
			Front=-1;
			Rear=-1;
		}
		else
		{
			Front++;
		}
		return item;
	}
}
```

What is DFS ?????

- DFS stands for Depth First Search.
- DFS is an algorithm for traversing or searching a tree or graph data structures.
- It uses a stack data structure for implementation.
- In DFS one starts at the root and explores as far as possible along each branch before backtracking.

Algorithm of DFS

[1]-- Initialize all nodes with status=1.(ready state)

[2]– Put starting node in the stack and change status to status=2(waiting state).

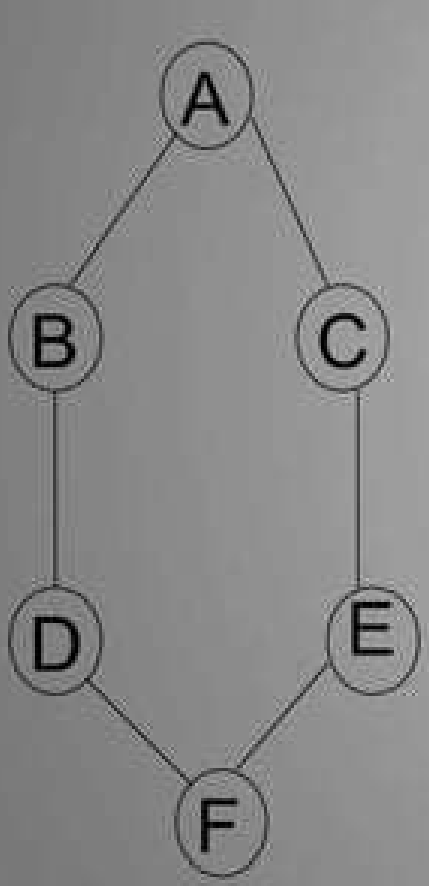
[3]– Loop:-

Repeat step- 4 and step- 5 until stack Get empty.

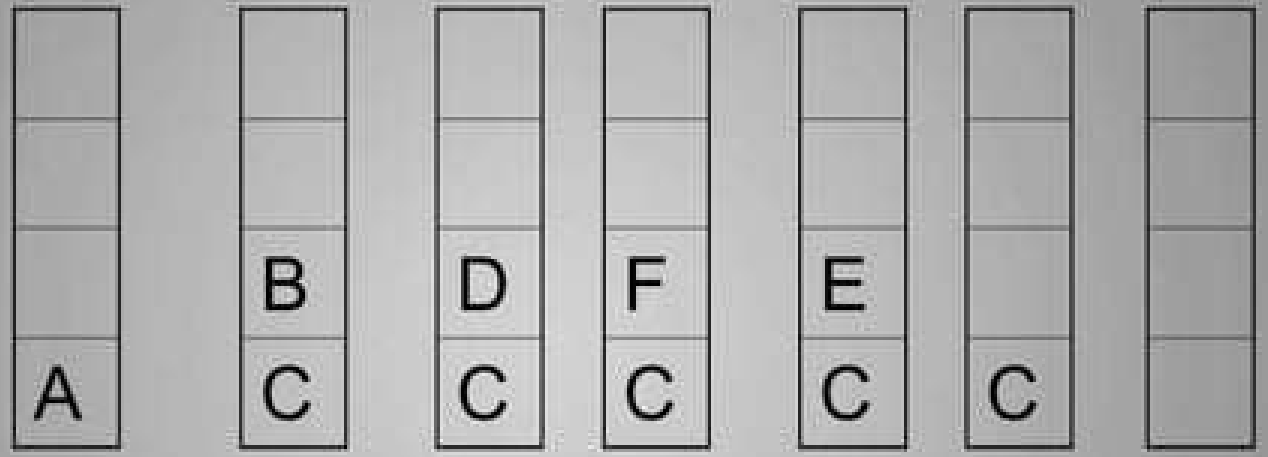
[4]– Remove top node N from stack process them and change the status of N processed state (status=3).

[5]– Add all the neighbours of N to the top of stack and change status to waiting status-2.

❑ Working of DFC :



Stack data structure



output

A

A
B

A
B
D

A
B
D
F

A
B
D
F
E

A
B
D
F
E
C

Code for DFS in C :

```
#include<stdio.h>
#include<conio.h>
#define n 8
Int a[8][8]={
            {0,1,1,1,0,0,0,0},
            {1,0,0,0,1,0,0,0},
            {1,0,0,0,0,1,0,0},
            {1,0,0,0,0,0,1,0},
            {0,1,0,0,0,0,0,1},
            {0,0,1,0,0,0,0,1},
            {0,0,0,1,0,0,0,1},
            {0,0,0,0,1,1,1,0}
      } ;
Int stack[20];
Int visit[8];
Int top=-1;
void dfs(int s);
void push(int item);
Int pop();
```

```
void main()
{
     int i,j,s;
     clrscr();

     printf("n\n THE ADJACENCY
     MATRIX IS \n\n");

     for(i=0;i<=n-1;i++)
     {
           for(j=0;j<=n-1;j++)
           {
           printf(" %d ",a[i][j]);
           }
           printf("\n");
     }
```

```
printf("\n\n ENTER THE
SOURCE VERTEX : ");

Scanf("%d",&s);

printf("\n\n DFS
TRAVERSAL IS  :  ");

dfs(s);

getch();

}
```

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

        if(k!=-1)
            {
                if(visit[k]==0)
                {
                printf(" %d ",k);
                visit[k]=1;
                getch();
                }
            }
    }
}
```

```
void push(int item)
{
    if(top==9)
    {
            printf("Stack overflow ");
    }

    else
    {     top = top + 1;
           stack[top]=item;
    }
}
```

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

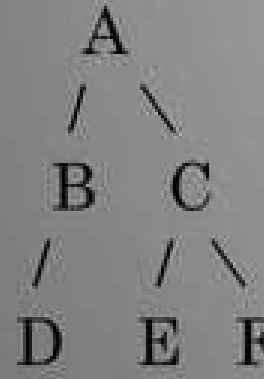
DFS V/S BFS

- DFS stands for Depth First Search.
- DFS can be done with the help of STACK i.e., LIOF.
- In DFS has higher time and space complexity, because at a time it needs to back tracing in graph for traversal.

- BFS stands for Breadth First Search.
- BFS can be done with the help of QUEUE i.e., FIOF.
- In BFS the space & time complexity is lesser as there is no need to do back tracing

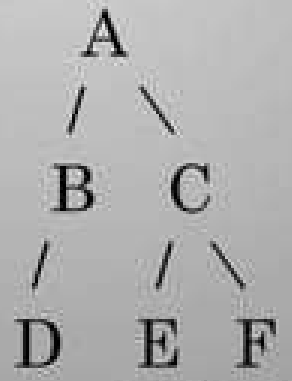
DFS V/S BFS

- DFS is more faster then BFS.
- DFS requires less memory compare to BFS.
- DFS is not so useful in finding shortest path.
- Example :



Ans : A,B,D,C,E,F

- BFS is slower than DFS.
- BFS requires more memory compare to DFS.
- BFS is useful in finding shortest path.
- Example :



Ans : A,B,C,D,E,F

Thank You