Department of Computer science and Engineering
PES UNIVERSITY
UE19CS202: Data Structures and its Applications (4-0-0-4-4)

Applications of BFS and DFS

Abstract

Finding the path in a network using dfs and bfs traversal method.

Dr.Sandesh and Saritha

Sandesh_bj@pes.edu

Saritha.k@pes.edu

Applications of BFS and DFS:

- The different applications of DFS are
- Detecting whether a cycle exist in graph.
- Finding a path in a network
- Topological Sorting: Used for job scheduling
- To check whether a graph is strongly connected or not: A directed graph is said to be strongly connected if there exist a path between every pair of vertex.

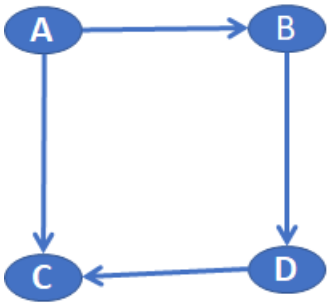
Applications of BFS are

- Finding the shortest path
- Social Networking websites like twitter, Facebook etc.
- GPS Navigation system
- Web crawlers
- Finding a path in network
- In Networking to broadcast the packets.

Application of DFS and BFS

Finding the path in a Network

Given an graph with N vertices and E edges and two vertices(x,y) from the graph, we need to print the path between these two vertices if the path exists and no otherwise .



The path from A to D=1.A→B→D

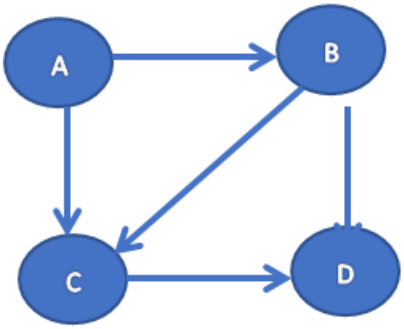
There exists no path from C to A

This can be achieved in 2 ways
1.DFS (Depth First Search)
2.BFS (Breadth First Search)

Function to traverse the graph using bfs

Finding all path in a network

- Path is defined as route between any two nodes. For Example given a directed graph, a source vertex s and a destination vertex d, print all the paths from s to d.
  For Example



Directed Graph

	A	B	C	D
A	0	1	1	0
B	0	0	1	1
C	0	0	0	1
D	0	0	0	0

Output:

Path from A to D: A->B->D

A->C->D

A->B->C->D

/*Program to print all the paths from a given source to vertex using dfs- matrix representation*/

```
#include<stdio.h>
#include<stdlib.h>
int a[10][10],n,p=0;
void read_ad_mat()
{
    for(int i=0;i<n;i++)
    {
        for(int j=0;j<n;j++)
        {
            scanf("%d",&a[i][j]);
        }
```

```
    }
}


void printall(int u,int d,int visited[10],int path[10])
{
    visited[u]=1;
    path[p]=u;
    p++;
    if(u==d)
    {

        for(int i=0;i<p;i++)
        {
            printf("%d ",path[i]);
        }
        printf("\n");
    }
    else{
        for(int v=0;v<n;v++)
            if(a[u][v]==1 && visited[v]==0)
            {
                printall(v,d,visited,path);
            }
    }
```

```
        p--;
        visited[u]=0;
    }

    void printpath(int s,int d)
    {
        int visited[10];
        int path[10];
        int p=0;
        for(int i=0;i<n;i++)
                visited[i]=0;
        printall(s,d,visited,path);
    }


    int main()
    {
        int i,so,de;
        printf("enter the number of nodes\n");
        scanf("%d",&n);
        printf("enter the adjacency list\n");
        read_ad_mat();
        printf("enter the source and destination\n");
        scanf("%d%d",&so,&de);
        printpath(so,de);
```

```
    }
```

/*program to find all the paths from a given source to destination using dfs list representation*/

```
#include<stdio.h>
#include<stdlib.h>

struct node
{
        int info;
        struct node *link;
};
typedef struct node *NODE;
NODE a[10];
int p=0;

NODE getnode()
{
        NODE x;
        x=(NODE) malloc(sizeof(struct node));
        if(x==NULL)
        {
                printf("out of memory\n");
                exit(0);
        }
        return(x);
```

```
}
NODE insert_rear(int ele,NODE first)
{
        NODE temp;
        NODE cur;
        temp=getnode();
        temp->info=ele;
        temp->link=NULL;
        if(first==NULL)
                return temp;
        cur=first;
        while(cur->link!=NULL)
                cur=cur->link;
        cur->link=temp;
        return first;
}

void read_ad_list(NODE a[],int n)
{
        int i,j,m,ele;
        for(i=0;i<n;i++)
        {
                printf("enter the number of nodes adjacent to %d\n",i);
                scanf("%d",&m);
                if(m==0)
```

```
                continue;
            printf("enter the nodes adjacent to %d\n",i);
            for(j=0;j<m;j++)
            {
                scanf("%d",&ele);
                a[i]=insert_rear(ele,a[i]);
            }
        }
}

void printall(int u,int d,int visited[10],int path[10])
{
        visited[u]=1;
        path[p]=u;
        p++;
        if(u==d)
        {

            for(int i=0;i<p;i++)
            {
                printf("%d ",path[i]);
            }
            printf("\n");
        }
        else{
```

```
                for(NODE temp=a[u];temp!=NULL;temp=temp->link)
                        if(!visited[temp->info])
                        {
                                printall(temp->info,d,visited,path);
                        }
        }
        p--;
        visited[u]=0;
}

void printpath(int s,int d,int n)
{
        int visited[10];
        int path[10];
        int p=0;
        for(int i=0;i<n;i++)
                visited[i]=0;
        printall(s,d,visited,path);
}


int main()
{
        int i,so,de;
        printf("enter the number of nodes\n");
```

```
        scanf("%d",&n);
        printf("enter the adjacency list\n");
        read_ad_list(a,n);
        printf("enter the source and destination\n");
        scanf("%d%d",&so,&de);
        printpath(so,de,n);
}
```