PES UNIVERSITY ONLINE

DATA STRUCTURES AND ITS APPLICATIONS

Graphs

Saritha

Department of Computer Science & Engineering

DATA STRUCTURES AND ITS APPLICATIONS

Finding all the paths from a given source to destination

Saritha

Department of Computer Science & Engineering

Applications of BFS and DFS

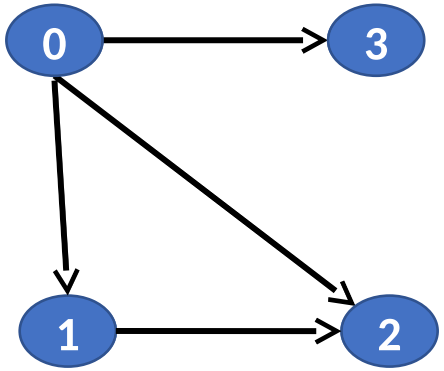
Application of DFS

- 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

Applications of BFS and DFS

Path

- Sequence of edges that allows the user to go from vertex A to vertex B



- The paths from vertex 0 to vertex 2:
- 1.0->1->2
- 2.0->2

Finding all the paths from the given source to destination

- **Methodology**
- 1.Start from any traversal Method from a given source node
- 2.Store all the visited vertices in an array
- 3.Once the destination vertex is reached, print all the contents of Array.

Function to print all the paths using DFS traversal method

```
//Function to print all the paths from a given source to destination
void path_find(int source,int destination)
{
    int visited[10]  //An array to store the vertices as visited or not
    int path[10]    //An array to store the path
    int count=0;
    for(int i=0;i<n;i++)
        visited[i]=0   //initilize all the vertices as not visited.
    printallpaths(source,destination,visited,path);
}
```

Function to print all the paths using DFS traversal method

```
void printallpaths(int u,int d,int visited[10],int path[10])
{
   visited[u]=1;//Mark the current node and and store it in the array path
   path[count]=u;
   count++;
    if(u==d) //if the current vertex is same as the destination then print the array
which has stored the path
    {
      for(i=0;i<count;i++)
             printf(“%d”,path[i]);
     }
   else // if the current vertex is not the destination
   {
      for(NODE temp=a[u];temp!=NULL;temp=temp->link)
```

Function to print all the paths using DFS traversal method

```
if(!visited[temp->data])
{
   printallpaths(temp->data,d,visited,path);
}
}
count--  //Remove the current vertex from the path[] and mark it as
unvisited
Visited[u]=0;
}
```

Function to read adjacency list

```
void read_adjacency_list(NODE a[],int n)
{
    int ele,m;
    for(int i=0;i<n;i++)
    {
        printf("enter the number of nodes adjacent to %d:",i);
         scanf("%d",&m);
         if(m==0)
          continue;
         printf("enter the nodes adjacent to %d:",i);
         for(int j=0;j<m;j++)
        {
           scanf("%d",&ele);
         a[i]=insert_rear(ele,a[i]); // insert at rear end
         }    } }
```

Function to insert a node at rear end of the list

Function to insert a node at rear end

```
NODE insert_rear(int v,NODE head)
 {
    NODE temp;
    NODE cur;
    temp=getnode(); // create a node
    temp->info=v;
     temp->link=NULL;
    if(head==NULL)
          return temp;
     cur=head;
   while(cur->link!=NULL)
   {
      cur=cur->link;
    }
   cur->link=temp;
 return(head); }
```

Function to create a node

```
NODE getnode()
 {
   NODE temp;
   temp=(NODE)malloc(sizeof(struct node)); //Dynamic allocation
   if(temp==NULL)
   {
      printf("out of memory");
       return;
    }
   return temp;
 }
```

THANK YOU

Saritha

Department of Computer Science & Engineering

Saritha.k@pes.edu

9844668963