

```
/*
```

```
Assignment 2:
```

- a) Use Depth First Search for traversing an undirected as well as a directed graph.
 - b) Differentiate its different edges.
 - c) Also count the number of components of an undirected graph.
- ```
*/
```

```
/*Including the header files*/
```

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

```
#include<conio.h>
```

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

```
/*Forward declaration of the adjacent vertex structure*/
```

```
struct subvert;
```

```
/*Structure of the main vertex*/
```

```
struct mainvert
```

```
{
```

```
 int ver;
```

```
 int dfsno;
```

```
 int dfscmpno;
```

```
 int visited;
```

```
 struct mainvert *nextver;
```

```
 struct subvert *adver;
```

```
};
```

```
/*Structure of the adjacent vertex*/
```

```
struct subvert
```

```
{
```

```
 struct mainvert *vert;
```

```
 struct subvert *next;
```

```
};
```

```
/*Fuction to create a memory allocation for the main vertex*/
```

```
struct mainvert *getmain(int x)
```

```
{
```

```
 struct mainvert *new1;
```

```
 new1=(struct mainvert *)malloc(sizeof(struct mainvert));
```

```
 new1->ver=x;
```

```
 new1->dfsno=0;
```

```
 new1->dfscmpno=0;
```

```
 new1->visited=0;
```

```
 new1->nextver=NULL;
```

```
 new1->adver=NULL;
```

```
 return(new1);
```

```
}
```

```

/*Function to create a memory allocation for the adjacent vertex*/
struct subvert *getsub()
{
 struct subvert *new1;
 new1=(struct subvert *)malloc(sizeof(struct subvert));
 new1->vert=NULL;
 new1->next=NULL;
 return(new1);
}

/*Declarations of the global variables for Adjacency List and Number of Components*/
struct mainvert *head;
int n,m;

int main()
{
 /*Declaration of the prototypes of the functions to be used*/
 void adjacency_list_create(int);
 void sort();
 void resort();
 void dfs_trav(int);
 void edge_diff(int);
 void display();
 int c;
 /*Loop for user's choice for the type of graph user wants to enter*/
 do
 {
 printf("\n\tMenu:-");
 printf("\n1.Undirected Graph");
 printf("\n2.Directed Graph");
 printf("\n3.Exit");
 printf("\nEnter Choice :- ");
 scanf("%d",&c);
 switch(c)
 {
 /*Case for Undirected graph*/
 case 1:
 adjacency_list_create(0);
 display();
 sort();
 dfs_trav(0);
 resort();
 edge_diff(0);
 break;
 /*Case for Directed graph*/

```

```

 case 2:
 adjacency_list_create(1);
 display();
 sort();
 dfs_trav(1);
 resort();
 edge_diff(1);
 break;
 case 3:
 exit(0);
 default:
 printf("\nWrong Input:Re-Enter\n");
 }
}while(1);
return 0;
}

```

/\*Function that creates the adjacency list entered by the user\*/

```

void adjacency_list_create(int x)
{
 struct mainvert *new1,*ptr,*ptr1;
 struct subvert *new2,*ptrr1;
 int f1,c,a;
 n=0;

 head=NULL;
 do
 {
 /*Creating the vertex list*/
 n++;
 new1=getmain(n);
 if(head == NULL)
 head=new1;
 else
 {
 ptr=head;
 while(ptr->nextver != NULL)
 ptr=ptr->nextver;
 ptr->nextver=new1;
 }
 printf("Vertex %d created",n);
 printf("\nDo you want to add any more vertex?(YES=1,NO=0) :- ");
 scanf("%d",&c);
 }while(c == 1);
 /*Entering the adjacent vertices*/
 printf("Enter the adjacent vertices of the vertices");
}

```

```

ptr=head;
while(ptr != NULL)
{
 f1=1;
 c=0;
 /*Checking if there are any vertices of the vertex whose adjacent vertices
are to be entered*/
 if(ptr->adver != NULL)
 {
 printf("\nVertices adjacent to %d are : ",ptr->ver);
 ptrr1=ptr->adver;
 while(ptrr1 != NULL)
 {
 printf("%d\t",ptrr1->vert->ver);
 ptrr1=ptrr1->next;
 }
 printf("\nDoes %d have any more adjacent
vertices?(YES=1,NO=0) :- ",ptr->ver);
 scanf("%d",&c);
 if(c == 0 || c > 1)
 f1=0;
 }
 /*If the vertex has adjacent vertices then those are entered by the user*/
 if(f1 == 1)
 {
 do
 {
 if(c == 1)
 {
 printf("\nEnter the vertex adjacent to %d (else 0):-
",ptr->ver);

 scanf("%d",&a);
 c=0;
 }
 else
 {
 printf("\nEnter the vertex adjacent to %d (if no
adjacent vertex enter 0):- ",ptr->ver);
 scanf("%d",&a);
 }
 if(a == 0)
 break;
 ptr1=head;
 while(ptr1 != NULL && ptr1->ver != a)
 ptr1=ptr1->nextver;
 if(ptr1 == NULL)

```

```

 {
 printf("\nWrong Input : Re-Enter\n");
 continue;
 }
 ptrr1=ptr->adver;
 while(ptrr1 != NULL && ptrr1->vert->ver != a)
 ptrr1=ptrr1->next;
 if(ptrr1 != NULL)
 {
 if(ptrr1->vert->ver == a)
 printf("\n%d is already adjacent to
%d",a,ptr->ver);
 }
 else
 {
 new2=getsub();
 new2->vert=ptrr1;
 if(ptr->adver == NULL)
 ptr->adver=new2;
 else
 {
 ptrr1=ptr->adver;
 while(ptrr1->next != NULL)
 ptrr1=ptrr1->next;
 ptrr1->next=new2;
 }
 if(x == 0)
 {
 new2=getsub();
 new2->vert=ptr;
 if(ptrr1->adver == NULL)
 ptrr1->adver=new2;
 else
 {
 ptrr1=ptrr1->adver;
 while(ptrr1->next != NULL)
 ptrr1=ptrr1->next;
 ptrr1->next=new2;
 }
 }
 }
 }while(1);
}
ptr=ptr->nextver;
}
}

```

```

/*Funtion for the DFS traversal of the entered graph*/
void dfs_trav(int x)
{
 struct mainvert *mstack[50],*tstack[50],*ptr,*ptr1,*ptr2,*temp;
 struct subvert *ptrr1;
 int s,dfsn,dfsc,mtop,ttop,c,f,f2,min,m=0;
 /*Entering the starting vertex*/
 printf("\nDFS Traversal:-");
 do
 {
 printf("\nEnter starting node :- ");
 scanf("%d",&s);
 ptr=head;
 while(ptr->ver != s)
 ptr=ptr->nextver;
 if(ptr == NULL)
 {
 printf("Wrong Input : %d is not a vertex of the given graph\n->To
re-enter press 1 else 0 :- ");
 scanf("%d",&c);
 if(c == 0)
 break;
 }
 else
 break;
 }while(1);
 /*Initializing variables*/
 mtop=-1;
 ttop=-1;
 dfsn=1;
 dfsc=1;
 tstack[++ttop]=ptr;
 ptr->visited=1;
 /*Giving the vertices DFS Number and DFS Completion Number as they are
pushed and popped out of the temporary and main stacks*/
 do
 {
 f=0;
 f2=1;
 temp=tstack[ttop--];
 if(temp->dfsno == 0)
 temp->dfsno=dfsn++;
 ptrr1=temp->adver;
 while(ptrr1 != NULL)
 {

```

```

 if(ptrr1->vert->visited == 0)
 {
 tstack[++ttop]=ptrr1->vert;
 ptrr1->vert->visited=1;
 f=1;
 }
 if(ptrr1->vert->dfsno == 0)
 f2=0;
 ptrr1=ptrr1->next;
 }
 mstack[++mtop]=temp;
 if(f == 0 && f2 == 1)
 {
 ptr=mstack[mtop--];
 ptr->dfscmpno=dfsc++;
 if(mtop != -1)
 ptr=mstack[mtop--];
 else
 {
 ptr1=head;
 min=ptr1->ver;
 ptr2=ptr1;
 while(ptr1 != NULL)
 {
 if(ptr1->ver < min && ptr1->visited != 1 && ptr1-
>ver != ptr->ver)
 {
 ptr2=ptr1;
 min=ptr1->ver;
 }
 ptr1=ptr1->nextver;
 }
 ptr=ptr2;
 m++;
 }
 tstack[++ttop]=ptr;
 }
 if(dfsc > n)
 break;
}while(1);
/*Displaying the DFS traversal Tree*/
for(s=1;s<=n;s++)
{
 ptr=head;
 while(ptr != NULL)
 {

```

```

 if(s == ptr->dfsno)
 {
 printf("%d\t",ptr->ver);
 break;
 }
 ptr=ptr->nextver;
 }
}
/*If undirected then displaying the number of components*/
if(x == 1)
 printf("\nNumber of components = %d",m);
printf("\n");
}

/*Differentiating the edges*/
void edge_diff(int x)
{
 struct mainvert *ptr;
 struct subvert *ptrr;
 printf("\nTree Edge:-\n");
 ptr=head;
 while(ptr != NULL)
 {
 ptrr=ptr->adver;
 while(ptrr !=NULL)
 {
 if((ptr->dfsno)-(ptrr->vert->dfsno) == 1 || (ptr->dfsno)-(ptrr->vert-
>dfsno) == -1 || (ptr->dfscmpno)-(ptrr->vert->dfscmpno) == 1 || (ptr->dfscmpno)-(ptrr-
>vert->dfscmpno) == -1)
 printf("(%d , %d)\t",ptr->ver,ptrr->vert->ver);
 ptrr=ptrr->next;
 }
 ptr=ptr->nextver;
 }
 printf("\nForward Edge:-\n");
 ptr=head;
 while(ptr != NULL)
 {
 ptrr=ptr->adver;
 while(ptrr !=NULL)
 {
 if(x == 1)
 {
 if(((ptr->dfsno) < (ptrr->vert->dfsno) && (ptr->dfscmpno)
> (ptrr->vert->dfscmpno)) && ((ptr->dfsno)-(ptrr->vert->dfsno) != 1 && (ptr->dfsno)-

```



```

(ptrr->vert->dfsno) != -1 && (ptr->dfscmpno)-(ptrr->vert->dfscmpno) != 1 && (ptr-
>dfscmpno)-(ptrr->vert->dfscmpno) != -1))
 printf("(%d , %d)\t",ptr->ver,ptrr->vert->ver);
}
else
{
 if((ptr->dfsno) < (ptrr->vert->dfsno) && (ptr->dfscmpno)
> (ptrr->vert->dfscmpno) && ((ptr->dfsno)-(ptrr->vert->dfsno) == 1 || (ptr->dfsno)-(ptrr-
>vert->dfsno) == -1 || (ptr->dfscmpno)-(ptrr->vert->dfscmpno) == 1 || (ptr->dfscmpno)-
(ptrr->vert->dfscmpno) == -1))
 printf("(%d , %d)\t",ptr->ver,ptrr->vert->ver);
}
ptrr=ptrr->next;
}
ptr=ptr->nextver;
}
printf("\nBack Edge:-\n");
ptr=head;
while(ptr != NULL)
{
 ptrr=ptr->adver;
 while(ptrr !=NULL)
 {
 if((ptr->dfsno) > (ptrr->vert->dfsno) && (ptr->dfscmpno) < (ptrr-
>vert->dfscmpno) && ((ptr->dfsno)-(ptrr->vert->dfsno) != 1 || (ptr->dfsno)-(ptrr->vert-
>dfsno) != -1 || (ptr->dfscmpno)-(ptrr->vert->dfscmpno) != 1 || (ptr->dfscmpno)-(ptrr-
>vert->dfscmpno) != -1))
 printf("(%d , %d)\t",ptr->ver,ptrr->vert->ver);
 ptrr=ptrr->next;
 }
 ptr=ptr->nextver;
}
printf("\nCross Edge:-\n");
ptr=head;
while(ptr != NULL)
{
 ptrr=ptr->adver;
 while(ptrr !=NULL)
 {
 if((ptr->dfsno) > (ptrr->vert->dfsno) && (ptr->dfscmpno) > (ptrr-
>vert->dfscmpno))
 printf("(%d , %d)\t",ptr->ver,ptrr->vert->ver);
 ptrr=ptrr->next;
 }
 ptr=ptr->nextver;
}

```

```

 printf("\n");
 }

 /*Initially sorting the adjacent vertices for DFS traversal*/
 void sort()
 {
 struct mainvert *ptr1,*ptr2;
 struct subvert *ptrr1,*ptrr2;
 ptr1=head;
 while(ptr1 != NULL)
 {
 ptrr1=ptr1->adver;
 while(ptrr1 != NULL)
 {
 ptrr2=ptrr1->next;
 while(ptrr2 != NULL)
 {
 if(ptr1->vert->ver < ptrr2->vert->ver)
 {
 ptr2=ptrr1->vert;
 ptr1->vert=ptrr2->vert;
 ptrr2->vert=ptr2;
 }
 ptrr2=ptrr2->next;
 }
 ptrr1=ptrr1->next;
 }
 ptr1=ptr1->nextver;
 }
 }

 /*ReSorting the adjacent vertices for the edge differentiation*/
 void resort()
 {
 struct mainvert *ptr1,*ptr2;
 struct subvert *ptrr1,*ptrr2;
 ptr1=head;
 while(ptr1 != NULL)
 {
 ptrr1=ptr1->adver;
 while(ptrr1 != NULL)
 {
 ptrr2=ptrr1->next;
 while(ptrr2 != NULL)
 {
 if(ptrr1->vert->ver > ptrr2->vert->ver)

```

```

 {
 ptr2=ptrr1->vert;
 ptrr1->vert=ptrr2->vert;
 ptrr2->vert=ptr2;
 }
 ptrr2=ptrr2->next;
 }
 ptrr1=ptrr1->next;
}
ptr1=ptr1->nextver;
}
}

```

/\*Displaying the adjacency list\*/

void display()

```

{
 struct mainvert *ptr1;
 struct subvert *ptrr1;
 /*Adjacency List Representation*/
 printf("\nAdjacency List\n");
 ptr1=head;
 printf("\nVertex:\tAdjacent Vertices\n");
 while(ptr1 != NULL)
 {
 printf("%d\t:",ptr1->ver);
 ptrr1=ptr1->adver;
 while(ptrr1 != NULL)
 {
 printf("%d\t",ptrr1->vert->ver);
 ptrr1=ptrr1->next;
 }
 printf("\n");
 ptr1=ptr1->nextver;
 }
}

```