GRAPH

Aim

Write a C Program to represent any given graph and to do the following operations:
a) Represent the graph using adjacency list and adjacency matrix.
b) Perform depth first search
c) Perform breadth first search

1 Graph

1.1 Algorithm

```
Step 1:Start
Step 2:Create a linked list for storing all the vertices.
Step 3:Ask input from the user to add edges.
Step 4:Store the edges of a vertex in the linked list starting at each vertex.
Step 5:Update the values in the Adjacency List according to the edges.
Step 6:Display the adjacency lis tand matrix corresponding to all the vertices.
Step 7:Call DFS() and BFS() to see the traversal using Depth-first search and
       Breadth First Search.
Step 8:Stop

Algorithm: BFS()
Step 1:Create a stack and push the first node of the graph to it
Step 2:While(stack_is_not_empty())
       value=pop()
       if(value is not visited)add it to the visit list and push to the
       stack all the adjacent vertices of value
    EndWhile

Algorithm: DFS()
Step 1:Create a queue and enqueue the first node of th egraph t oit
Step 2:While(queue_is_not_empty())
        value=pop()
        if(value is not visited)add to the visit list and enqueue all the
        adjacent vertices of value.
    EndWhile
```

1.2 Program

```
#include<stdlib.h>
#include<stdio.h>
struct vertex{
int data;
struct vertex * down;
struct edge * next;
}*head=NULL;
struct edge{
int data;
struct edge * next;
}*visit1=NULL,*visit2=NULL,*stacktop=NULL,*queuefront=NULL,*queuerear=NULL;
int A[100][100];
int numofvertices;
void initializematrix(int n)
{
for(int i=0;i<n;i++)
{
for(int j=0;j<n;j++)
A[i][j]=0;
}

}
void addEdge(int a,int b)
{
A[a][b]=A[a][b]+1;
}
void BuildAdjacencyList(int n)
{
for(int i=0;i<n;i++)
{
struct vertex * new = (struct vertex *) malloc(sizeof(struct vertex));
printf("Enter the vertex: ");
scanf("%d",&new->data);
new->next=NULL;
new->down=NULL;
if(head==NULL)
{
head = new;
}
else
{
struct vertex * temp=head;
while(temp->down!=NULL)
{
```

```
temp = temp->down;
}
temp->down=new;
}
}
struct vertex * list=head;

while(list!=NULL)
{
char choice='y';
printf("Do you want add a new edge for %d (y/n): ",list->data);
scanf(" %c",&choice);
while(choice=='y'||choice=='Y')
{
int val,flag=0;
struct edge * new = (struct edge *) malloc(sizeof(struct edge));
printf("Enter the edge vertex: ");
scanf("%d",&val);
struct vertex *temp= head;
while(temp!=NULL)
{
if(val==temp->data)
{
flag=1;
break;
}
temp=temp->down;
}
if(flag==0)
{
printf("INVALID VERTEX\n");
printf("Do you want add a new edge for %d (y/n): ",list->data);
scanf(" %c",&choice);
continue;
}
new->data=val;

new->next=NULL;
if(list->next==NULL)
{
list->next = new;
}
else
{
struct edge * temp = list->next;
while(temp->next!=NULL)
```

```
{
temp=temp->next;
}
temp->next=new;
}
printf("Do you want add a new edge for %d (y/n): ",list->data);
scanf(" %c",&choice);
}
list=list->down;
}
}
void BuildAdjacencyMatrix()
{
struct vertex * temp1= head;
initializematrix(numofvertices);
while(temp1!=NULL)
{
struct edge * temp2 = temp1->next;
while(temp2!=NULL)
{


addEdge((temp1->data)-1,(temp2->data)-1);
temp2=temp2->next;
}
temp1=temp1->down;
}
}
void printAdjacencyList()
{
struct vertex * temp1= head;
while(temp1!=NULL)
{
printf("\n%d",temp1->data);
struct edge * temp2 = temp1->next;
while(temp2!=NULL)
{
printf("-> %d",temp2->data);
temp2=temp2->next;
}
temp1=temp1->down;
}
}
void printAdjacencyMatrix(int n)
{
printf(" ");
```

```
for(int i=0;i<n;i++)
printf("%d ",i+1);
for(int i=0;i<n;i++)
{
printf("\n%d ",i+1);
for(int j=0;j<n;j++)
printf("%d ",A[i][j]);
}
}
int searchvisit1(int num)
{
struct edge *curr=visit1,*prev=NULL;
while(curr!=NULL)
{
if(curr->data==num)
return 1;
prev=curr;
curr=curr->next;
}
struct edge * new= (struct edge *)malloc(sizeof(struct edge));
new->next=NULL;
new->data=num;
if(visit1==NULL)
{
visit1=new;
}
else
prev->next=new;
printf(" %d ",num);
return 0;
}
void push(int item)
{
struct edge * new= (struct edge *)malloc(sizeof(struct edge));

new->next=stacktop;
new->data=item;
stacktop = new;
}
int pop()
{
int item = stacktop->data;
struct edge * temp=stacktop;
stacktop=stacktop->next;
free(temp);
return item;
```

```
}
int searchvisit2(int num)
{
struct edge *curr=visit2,*prev=NULL;
while(curr!=NULL)
{
if(curr->data==num)
return 1;
prev=curr;
curr=curr->next;
}
struct edge * new= (struct edge *)malloc(sizeof(struct edge));
new->next=NULL;
new->data=num;
if(visit2==NULL)
{
visit2=new;
}

else
prev->next=new;
printf(" %d ",num);
return 0;
}
void enqueue(int item)
{
struct edge * new=(struct edge *)malloc(sizeof(struct edge));
new->data = item;
new->next=NULL;
if(queuerear==NULL)
{
queuefront=queuerear=new;
}
else
{
queuerear->next=new;
queuerear=new;
}
}
int dequeue()
{
struct edge * list=queuefront;
int item=queuefront->data;
if(queuefront==queuerear)
{
queuefront=queuerear=NULL;
```

```
}
else

{
queuefront=queuefront->next;
}
free(list);
return item;
}
void SearchNeighbours1(int val)
{
struct vertex *temp=head;
while(temp!=NULL)
{
if(temp->data==val)
{
struct edge *temp2 = temp->next;
while(temp2!=NULL)
{
push(temp2->data);
temp2=temp2->next;
}
return;
}
temp=temp->down;
}
}
void SearchNeighbours2(int val)
{
struct vertex *temp=head;
while(temp!=NULL)
{


if(temp->data==val)
{
struct edge *temp2 = temp->next;
while(temp2!=NULL)
{
enqueue(temp2->data);
temp2=temp2->next;
}
return;
}
temp=temp->down;
}
```

```
}

void BreadthFirstSearch()
{
int item;
enqueue(head->data);
while(queuefront!=NULL)
{
item=dequeue();
if(!searchvisit2(item))
SearchNeighbours2(item);
}
}
void DepthFirstSearch()
{
int item;
push(head->data);

while(stacktop!=NULL)
{
item=pop();
if(!searchvisit1(item))
SearchNeighbours1(item);
}
}
void main()
{
printf("TO BUILD A GRAPH\nEnter the number of vertices: ");
scanf("%d",&numofvertices);
BuildAdjacencyList(numofvertices);
BuildAdjacencyMatrix();
printf("\nADJACENCY LIST: \n");
printAdjacencyList();
printf("\nADJACENCY MATRIX: \n");
printAdjacencyMatrix(numofvertices);
printf("\nDEPTH FIRST SEARCH: ");
DepthFirstSearch();
printf("\nBREADTH FIRST SEARCH: ");
BreadthFirstSearch();
}
}
```

1.3 Sample Output

```
TO BUILD A GRAPH
Enter the number of vertices: 3
Enter the vertex: 1
Enter the vertex: 2
Enter the vertex: 3
Do you want add a new edge for 1 (y/n): y
Enter the edge vertex: 2
Do you want add a new edge for 1 (y/n): n
Do you want add a new edge for 2 (y/n): y
Enter the edge vertex: 1
Do you want add a new edge for 2 (y/n): y
Enter the edge vertex: 2
Do you want add a new edge for 2 (y/n): n
Do you want add a new edge for 3 (y/n): n

ADJACENCY LIST:

1-> 2
2-> 1-> 2
3
ADJACENCY MATRIX:
 1 2 3
1 0 1 0
2 1 1 0
3 0 0 0
DEPTH FIRST SEARCH:  1  2
BREADTH FIRST SEARCH:  1  2
```

Figure 1: Input and Output

1.4 Result

A C program was made to represent any given graph and to do the following operations represent the graph using adjacency list and matrix, perform depth-first search and breadth-first search.