**BFS and DFS**

**Done By:** Rohit Karunakaran **Roll No:** 58

**Aim:** Implementation of depth first search and breadth first search using array

**Data Structures used:** Graphs, Array

**Algorithm for Breadth First Search (bfs)**

**Input:** The Graph data structure (G) and the starting node S

**Output :** The nodes in the graph traversed in bfs order

**Data Structure :** Graphs, queue

**Steps**

* 1. Step 1: Start
  2. Step 2: let Q be a queue
  3. Step 3: Q.enque(s)
  4. Step 4: visit(s)
  5. Step 5: mark s as visited
  6. Step 6: while(Q is not empty) do
  7. Step 1: v= Q.dequeue()
  8. Step 2: for all nodes w of v in the graph g do
  9. Step 1: if(w is not visited) then
  10. Step 1: Q.enqueue(w)
  11. Step 2: visit(w)
  12. Step 3: mark w as visited
  13. Step 2: endif
  14. Step 3: done
  15. Step 7: done
  16. Step 8: stop

**Algorithm for Depth First Search (dfs)**

**Input:** The graph G and the starting node A

**Output :** All the elements in G traversed in DFS order

**Data Structure used:** Graph, stack

**Steps**

1. Step 1: Start
2. Step 2: let S be a stack
3. Step 3: S.push(A)
4. Step 4: while S is not empty do
5. Step 1: v = S.pop()
6. Step 2: if (v is not visited) then
7. Step 1: visit(v)
8. Step 2: mark v as visited
9. Step 3: push all adjacent vertex of v into the stack
10. Step 3: endif
11. Step 5: endWhile
12. Step 6: stop

**Result:** The program was successfully compiled and the desired output was obtained

**Program Code**

/\* Breadth first and depth first search

\* Done By: Rohit Karunakaran

\*/

#include<stdlib.h>

#include<stdio.h>

int dequeue(int \*q,int \*f, int \*b){

int elem = q[\*f];

if((\*f)==(\*b)){

(\*f)=(\*b)=-1;

}

else{

(\*f)++;

}

return elem;

}

void enqueue(int \*q, int \*f, int \*b, int elem)

{

(\*b) = (\*b)+1;

q[(\*b)] = elem;

if((\*f)==-1){

(\*f)++;

}

}

void bfs(int\* vert, int\*\* a\_m, int nv,int ne){

if(nv!=0){

int queue[2\*nv];

int f=0, b=0;

int visited[nv];

int vc=0;

int i=0; //nodes accessed.

visited[0]=i; //visited the 0th node

queue[f] = i;

while(f!=-1){

int c = dequeue(queue,&f,&b);

for(int i = 0; i<nv;i++){ //itereate through all the edges

if(a\_m[c][i]==1){ //If an edge is connected to c

int flag=1;

for(int j = 0;j<=vc;j++) //check if the edge is visited

{

flag ==1;

if(visited[j]==i){

flag = 0;

break;

}

}

if(flag){ //If the edge is not visited then visit it....

enqueue(queue,&f,&b,i);

visited[++vc] = i;

}

}

}

}

for(int i = 0;i<=vc;i++){

printf("%d ",vert[visited[i]]);

}

}

}

void dfs(int\* vert, int\*\* a\_m, int nv,int ne){

if(nv!=0){

int stack[2\*nv];

int top=0;

int visited[nv];

int vc=-1;

int i=0; //nodes accessed.

stack[top] = i;

while(top!=-1){

int c = stack[top--];

int flag = 1;

for(int j = 0;j<=vc;j++){ //check if the edge is visited{

if(visited[j]==c){

flag = 0;

break;

}

}

if(flag){

visited[++vc] = c;

for(int i = 0;i<nv;i++){

if(a\_m[c][i]==1){

stack[++top] = i;

}

}

}

}

for(int i = 0;i<=vc;i++){

printf("%d ",vert[visited[i]]);

}

}

}

void main(){

int nv,ne;

printf("BFS and DFS implementation\n");

printf("Enter the number of vertices: ");

scanf("%d%\*c",&nv);

int\*\* adj\_matrix = (int\*\*)calloc(nv,sizeof(int\*));

for(int i = 0;i<nv;i++)

adj\_matrix[i] = (int\*)calloc(nv,sizeof(int));

int \*vertices = (int\*)malloc(nv\*sizeof(int ));

printf("\nEnter the vertices of the Graph: ");

for(int i=0;i<nv;i++){

scanf("%d",&vertices[i]);

}

printf("Enter the number of edges: ");

scanf("%d",&ne);

printf("Enter the vetices connected by the edges in the form-> start end\n");

for(int i=0;i<ne;){

int s,e;

if(scanf("%d %d",&s,&e)==2){

adj\_matrix[s][e]=1;

adj\_matrix[e][s]=1;

i++;

}

else{

printf("Enter the vertices in the correct format\n");

}

}

printf("The Breadth first traversl: ");

bfs(vertices, adj\_matrix,nv,ne);

printf("\n");

printf("The Depth first traversal: ");

dfs(vertices,adj\_matrix,nv,ne);

printf("\n");

}

**Sample input and output**



