**广度优先搜索和深度优先搜索**

广度优先遍历：又叫层次遍历，从上往下对每一层依次访问，在每一层中，从左往右（也可以从右往左）访问结点，访问完一层就进入下一层，直到没有结点可以访问为止。广度优先遍历的非递归的通用做法是采用队列。

深度优先遍历：对每一个可能的分支路径深入到不能再深入为止，而且每个结点只能访问一次。二叉树的深度优先遍历的非递归的通用做法是采用栈，要特别注意的是，二叉树的深度优先遍历比较特殊，可以细分为先序遍历、中序遍历、后序遍历。具体说明如下：  
  
先序(根)遍历：对任一子树，先访问根，然后遍历其左子树，最后遍历其右子树。  
中序(根)遍历：对任一子树，先遍历其左子树，然后访问根，最后遍历其右子树。  
后序(根)遍历：对任一子树，先遍历其左子树，然后遍历其右子树，最后访问根

广度优先BFS

一．代码实现

#include<iostream>

using namespace std;

#define len 100

class Queue{

public:

int front;

int rear;

int count;

int data[len];

Queue(){

front=0;

rear=0;

count=0;

}

void Enter\_Queue(int number){

data[rear++]=number;

count++;

}

int get\_Queue(){

int temp=data[front++];

count--;

return temp;

}

bool ls\_Empty(){

if(front==rear){

return true;

}

else{

return false;

}

}

};

class Graph{

public:

char node[len];

int edge[len][len];

int n;

int e;

int visited[len];

Graph(Queue& que){

n=7;

e=7;

string nodes="ABCDEFG";

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

node[i]=nodes[i];

}

char edges[][2]={

{'A','C'},

{'A','D'},

{'A','F'},

{'C','B'},

{'C','D'},

{'F','G'},

{'G','E'}

};

for(i=0;i<n;i++){

for(int j=0;j<n;j++){

edge[i][j]=0;

}

}

for(i=0;i<e;i++){

int start=get\_Node\_Index(edges[i][0]);

int end=get\_Node\_Index(edges[i][1]);

edge[start][end]=1;

edge[end][start]=1;

}

for(i=0;i<n;i++){

visited[i]=0;

}

for(i=0;i<n;i++){

if(!visited[i]){

BFS(i,que);

}

}

}

int get\_Node\_Index(char number){

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

if(number==node[i]){

return i;

}

}

return -1;

}

void BFS(int number,Queue&que){

que.Enter\_Queue(number);

cout<<"广度优先遍历序列为:"<<node[number]<<endl;

visited[number]=1;

while(!que.ls\_Empty()){

int k=que.get\_Queue();

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

if(edge[k][i]==1&&!visited[i]){

cout<<"广度优先 遍历序列为:"<<node[i]<<endl;

que.Enter\_Queue(i);

visited[i]=1;

}

}

}

}

};

int main(){

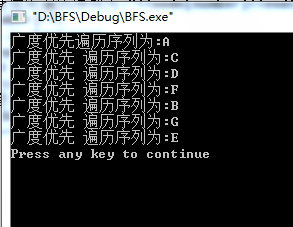
Queue que;

Graph graph(que);

return 0;

}

二．运行结果



深度优先遍历

1. 代码实现

#include <iostream>

#include <stack>

using namespace std;

#define MaxNode 20

#define MAX 2000

#define StartNode 1

int map[MaxNode+1][MaxNode+1];

void dfs\_stack(int start, int n){

int visited[MaxNode],s\_top;

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

visited[i] = 0;

}

visited[start] = 1;

stack <int> s;

cout<<start<<" ";

for(int i = 1; i <= n; i++){

if(map[i][start] == 1 && !visited[i] ){

visited[i] = 1;

s.push(i);

}

}

while(!s.empty()){

s\_top = s.top();

visited[s\_top] = 1;

cout<<s\_top<<" ";

s.pop();

for(int i = 1; i <= n; i++){

if(map[i][s\_top] == 1 && !visited[i] ){

visited[i] = 1;

s.push(i);

}

}

}

}

int main(int argc, const char \* argv[]) {

int num\_edge,num\_node;

int x,y;

cout<<"Input number of nodes and edges >"<<endl;

cin>>num\_node>>num\_edge;

for(int i =0;i<num\_node;i++){

for(int j=0;j<num\_node;j++){

map[i][j] = 0;

}

}

for(int i = 1; i <= num\_edge; i++){

cin>>x>>y;

map[x][y] = map[y][x] = 1;

}

dfs\_stack(StartNode, num\_node);

return 0;

}

运行结果

