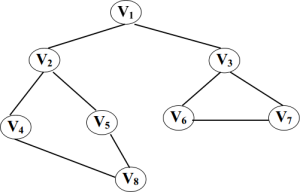
实验四 广度优先算法

171491328 罗晓璐

一、实验内容

广度优先遍历：

所谓广度，就是**一层一层的，向下遍历，层层堵截**，下面这幅图，我们如果要是广度优先遍历的话，我们的结果是V1 V2 V3 V4 V5 V6 V7 V8。



**广度优先搜索的思想：**

　　① 访问顶点vi ；

　　② 访问vi 的所有未被访问的邻接点w1 ,w2 , …wk ；

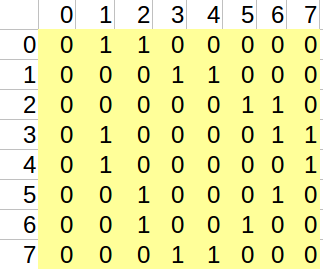
　　③ 依次从这些邻接点（在步骤②中访问的顶点）出发，访问它们的所有未被访问的邻接点; 依此类推，直到图中所有访问过的顶点的邻接点都被访问；

**说明：**

为实现③，需要保存在步骤②中访问的顶点，而且访问这些顶点的邻接点的顺序为：先保存的顶点，其邻接点先被访问。 这里用标准模板库中的queue队列来实现这种先进现出的服务。

二、实验过程

首先用邻接矩阵来存储这个图，通过广度优先来遍历该图。



算法：

#include <queue>

using namespace std;

#define MaxVnum 50

typedef double AdjMatrix[MaxVnum][MaxVnum]; //表示一个矩阵，用来存储顶点和边连接关系

typedef struct {

int vexnum,arcnum;//顶点的个数，边的个数

AdjMatrix arcs;//图的邻接矩阵

}Graph;

void CreateGraph(Graph &G)

{

G.vexnum=8;

G.arcnum=9;

G.arcs[0][1]=1;

G.arcs[0][2]=1;

G.arcs[1][3]=1;

G.arcs[1][4]=1;

G.arcs[2][5]=1;

G.arcs[2][6]=1;

G.arcs[3][1]=1;

G.arcs[3][7]=1;

G.arcs[3][6]=1;

G.arcs[4][1]=1;

G.arcs[4][7]=1;

G.arcs[5][2]=1;

G.arcs[5][6]=1;

G.arcs[5][5]=1;

G.arcs[6][2]=1;

G.arcs[6][5]=1;

G.arcs[7][3]=1;

G.arcs[7][4]=1;

}

bool visited[MaxVnum];

void BFSTraverse(Graph G)

{

for (int v=0;v<G.vexnum;v++) //先将其所有顶点都设为未访问状态

visited[v]=false;

queue<int> Q;

for(int v=0;v<G.vexnum;v++)

{

if(visited[v]==false) //若该点没有访问

{

Q.push(v);//将其加入到队列中

visited[v]=true;

while (!Q.empty())//只要队列不空，遍历就没有结束

{

int t =Q.front();//取出对头元素

Q.pop();

printf(" %d ",t+1);

for(int j=0;j<G.vexnum;j++) //将其未访问过的邻接点加进入队列

if(G.arcs[t][j]==1&&visited[j]== false)

{

Q.push(j);

visited[j]=true; //在这里要设置true，因为这里该顶点我们已经加入到了队列，为了防止重复加入！

}

}

}

}

}

void main()

{

void CreateGraph(Graph &G);

void BFSTraverse(Graph G);

Graph G;

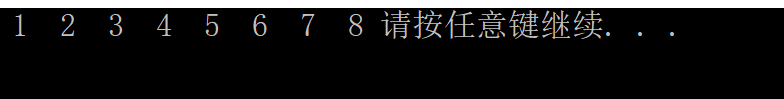
CreateGraph(G);

BFSTraverse(G);

return;

}

结果：



三、总结

广度优先搜索算法就是根据树或图的结构一层一层从左到右地去遍历找到答案的算法。

查找每个顶点的邻接点所需时间为O(n2)，n为顶点数，算法的时间复杂度为O(n2)。