**实 验 报 告**

**课程名称 数据结构（c）（英文）**

**实验项目** 图的建立和遍历

**实验仪器** 计算机

**系 别** 计算机学院学院

**专 业** 计算机大类

**班级/学号** 计算机类1801 2018011137

**学生姓名**  武文斌

**实验日期**  2019.12.01

**成 绩**

**指导教师**  蔡英

实验四

1. 实验目的
2. 学习邻接表的储存结构
3. 学习使用深度遍历和广度遍历
4. 打印简单路径（不含回路）
5. 实验内容

1）建立无向非连通图的邻接表存储结构，要求顶点个数不少于15个。

2）用DFS及BFS对此邻接表进行遍历，打印出两种遍历的顶点访问顺序。

3）给定图中任意两个顶点v1和v2及整数k，判断是否存在从v1到v2的路径长度为k的简单路径，若有打印出路径上的顶点序列（要求路径上不含回路）。进一步：找出从v1到v2的所有路径长度为k的简单路径。（简单路径：顶点序列中不含重现的顶点的路径。）

三、设计思路

1、构件图

//在构建图时，需要两个结构体  
// 一个存储图中节点的信息，便是上面介绍的头节点  
// 一个是两个节点之间边的信息，便是上面的表（边）节点。  
// 采用一个一维数组存储头结点信息，然后为每个头结点建立一个链表  
// 让头结点作为这个链表的表头节点  
// 链表中存储的正是和这个头结点相关联的表节点（边的信息）  
// 这些边都是尾部和头结点相连。表节点中的adjvex存储的是与这条边关联的另一个头结点在数组中的索引（边的箭头所关联的头节点）  
// 表节点中的nextarc则是存储的该链表表头相关联的另一条边的信息。

1. 深度遍历

假设从顶点出发进行搜索，在访问了顶点v1之后，选择邻接点v2。因为v2未曾访问，则从v2出发进行搜索，以此类推，接着从v4，v8，v5出发尽心搜索。在访问了v5之后，由于v5的邻接点都被访问，则回到v8。然后......就这样一直回到v1，然后又从v1搜索v3，如此进行下去。

1. 广度遍历

访问图中的一个点之后，一次访问v的各个未曾访问过的邻接点，然后分别从这些邻接点出发，依次访问它们的邻接点，并且使“先被访问的顶点的邻接点”先于“后被访问的顶点的邻接点”被访问。就像向湖面投一粒石子，激起一层的波纹。我们访问一个点，先把这个点的所有未被访问的邻接点依次全部访问，然后再在已经访问的的邻接点中找到那个最早被访问的点，从这个点出发，访问这个点所有未被访问的邻接点，如此循环下去。

四、源代码

#include<iostream>  
#include<string>  
#include<queue>  
using namespace std;  
#define ERROR 1  
#define MAX\_VERTEX\_NUM 100  
typedef struct ArcNode{//定义点结构体  
 int adjvex;//表（边）节点（存储两个顶点之间边的信息）  
 struct ArcNode \*nextarc;//表（边）节点（存储两个顶点之间边的信息）  
 string info;//表（边）节点（存储两个顶点之间边的信息）  
}ArcNode;  
typedef struct VNode{  
 char date;//头结点(储存顶点信息)  
 ArcNode \* firstarc;//头结点(储存顶点信息)  
}VNode,AdjList[MAX\_VERTEX\_NUM];  
typedef struct{  
 AdjList vertices;  
 int vexnum,arcnum; //当前图的vexnum顶点数和arcnum弧数  
 int kind;  
}ALGraph;  
int LocateVex(ALGraph &G,char &v1)  
//查找节点V1在图G的存储节点数组中的索引位置  
{  
 int i;  
 for(i=0;i<G.vexnum;i++)  
 {  
 if(G.vertices[i].date==v1)  
 //如果数组中有这个节点，返回该节点在数组中的索引  
 return i;  
 }  
 if(i>=G.vexnum)  
 return ERROR;  
 else  
 return 0;  
}  
  
//创建图  
void CreateDG(ALGraph &G)  
{  
 ArcNode \*p,\*q;  
 char v1,v2;  
 char v;  
 int i,j,k,n;  
 cout<<"Please enter the number of vertices and arcs of the graph:"<<endl;//请输入图的顶点数和弧数：  
 cin>>G.vexnum;  
 //输入图的顶点数量  
 cin>>G.arcnum;  
 //输入图的弧（边）的数量  
 cout<<"Please enter vertex:"<<endl;//请输入顶点：  
  
 for(i=0;i<G.vexnum;i++)  
 //创建头结点  
 {  
 cin>>v;  
 //输入顶点名字  
 G.vertices[i].date=v;  
 G.vertices[i].firstarc=NULL;  
 }  
 cout<<"Please enter arc tail and arc head:";//请输入弧尾和弧头：  
 for(k=0;k<G.arcnum;k++)  
 //创建边，并连接头结点  
 {  
 cin>>v1;  
 //v1为弧尾  
 cin>>v2;  
 //v2为弧头  
 i=LocateVex(G,v1);j=LocateVex(G,v2);  
  
 if(G.vertices[i].firstarc==NULL)  
 //如果链表为空新建一个表节点，让头节点的指针指向该表节点  
 {  
 p=(ArcNode \*)new ArcNode;  
 G.vertices[i].firstarc=p;  
 q=G.vertices[i].firstarc;  
 }  
 else  
 //链表的插入  
 {  
 q=G.vertices[i].firstarc;  
 //获取头结点的表头指针  
 for(n=0;n<G.arcnum;n++,q=q->nextarc)  
 //将q指针移动至链表的尾巴处  
 {  
 if(!q->nextarc)  
 break;  
 }  
 p=(ArcNode \*)new ArcNode;  
 q->nextarc=p;  
 //将该边（弧）加入到链表中  
 q=q->nextarc;  
 }  
 q->adjvex=j;  
 //记录弧头的索引  
 q->nextarc=NULL;  
 }  
 cout<<"successful";  
}  
  
//深度优先遍历  
bool visited[MAX\_VERTEX\_NUM];  
//访问标志数组，通过该数组表示顶点是否已访问  
// 当visited[i]为false时，表示点i并未被访问。  
int FirstAdjVex(ALGraph &G,int v)  
//找到在图G中的，与顶点G.vertices[v]相邻的未曾被访问的邻接点  
{  
 int i;  
 int n=-1;  
 ArcNode\*p;  
 p=G.vertices[v].firstarc;  
 if(p)  
 {  
 i=p->adjvex;  
 if(visited[i]==false)  
 n=i;  
 }  
 return n;  
}  
int NextAdjVex(ALGraph &G,int v)  
//功能与上面的函数类似  
{  
 int i;  
 int n=-1;  
 ArcNode \*p;  
 p=G.vertices[v].firstarc;  
 for(i=p->adjvex;i<G.vexnum,p!=NULL;)  
 {  
 i=p->adjvex;  
 if(visited[i]==false)  
 {  
 n=i;  
 break;  
 }  
 else  
 p=p->nextarc;  
 }  
 return n;  
}  
  
void VisitFuc(ALGraph &G,int v)  
//访问输出  
{  
 cout<<G.vertices[v].date<<" ";  
}  
void DFS(ALGraph &G,int v)  
//对图G做深度优先遍历，遍历点从索引为v的顶点开始  
{  
 int w;  
 visited[v]=true;  
 //设置索引为v的顶点为已访问  
 VisitFuc(G,v);  
 //访问索引为v的顶点  
 //核心：循环递归  
 //然后找到v的未曾访问的另一个邻接点访问，直至v的所有邻接点都被访问为止  
 for(w=FirstAdjVex(G,v);w>=0;w=NextAdjVex(G,v))  
 if(!visited[w]) DFS(G,w);//递归调用DFS  
  
}  
void DFSTraverse(ALGraph &G)  
//深度优先遍历的起始函数，调用此函数开始遍历。  
{  
 int v;  
 for(v=0;v<G.vexnum;v++)  
 visited[v]=false;  
 //初始化，所有点都为被访问，统统设为false  
 cout<<"depth:"<<endl;  
 for(v=0;v<G.vexnum;v++)  
 //确保遍历所有的点  
 {  
 if(!visited[v])  
 //如果未被访问  
 DFS(G,v);  
 //对该顶点v调用DFS方法  
 }  
}  
  
//广度优先遍历  
void BFSTraverse(ALGraph &G)  
{  
 int v;  
 int w;  
 queue<int> q; //STL队列  
 for(v=0;v<G.vexnum;v++)  
 visited[v]=false;  
 //初始化，标记数组设置为false  
// InitQueue(Q);  
 cout<<"breadth:";  
 for(v=0;v<G.vexnum;v++)  
 {  
 if(!visited[v])  
 //如果未曾被访问  
 {  
 visited[v]=true;  
 //标记为已访问  
 VisitFuc(G,v);  
 //访问该点  
 q.push(v); //v进队  
 //此处用队列的含义：每次访问一个点，把该点入队  
 // 当对这个点进行了广度优先遍历，也就是所有邻接点都被访问了  
 // 该点就出队  
 //所以，当对列不为空时，说明还有顶点没有被进行广度优先遍历。需要继续  
 while(q.empty()!=true)  
 {  
 v = q.front();  
 q.pop();  
 //出队  
 //w为v的尚未访问的邻接点  
 for(w=FirstAdjVex(G,v);w>=0;w=NextAdjVex(G,v))  
 { if(!visited[w])  
 {  
 visited[w]=true;  
 VisitFuc(G,w);  
  
 q.push(w);  
 }  
 }  
 }  
 }  
 }  
}  
  
void menu()  
{  
 cout<<'\n';  
 cout<<" The basic operation of the graph "<<endl;//图的基本操作  
 cout<<" \*\* 1 The construction of a figure\*\*"<<endl;//图的构建  
 cout<<" \*\* 2 depth \*\*"<<endl;  
 cout<<" \*\* 3 breadth \*\*"<<endl;  
 cout<<" --------------------------------------------"<<endl;  
 cout<<"Please enter a number to select:"<<endl;  
}  
int main()  
{  
 ALGraph G;  
 int i;  
 menu();  
 cin>>i;  
 while(i<4)//选择调用  
 {  
 switch(i)  
 {  
 case 1:CreateDG(G);break;  
 case 2:DFSTraverse(G);cout<<endl;break;  
 case 3:BFSTraverse(G);cout<<endl;break;  
 default:return ERROR;  
 }  
 menu();  
 cin>>i;  
 }  
 return 0;  
}  
//在构建图时，需要两个结构体  
// 一个存储图中节点的信息，便是上面介绍的头节点  
// 一个是两个节点之间边的信息，便是上面的表（边）节点。  
// 采用一个一维数组存储头结点信息，然后为每个头结点建立一个链表  
// 让头结点作为这个链表的表头节点  
// 链表中存储的正是和这个头结点相关联的表节点（边的信息）  
// 这些边都是尾部和头结点相连。表节点中的adjvex存储的是与这条边关联的另一个头结点在数组中的索引（边的箭头所关联的头节点）  
// 表节点中的nextarc则是存储的该链表表头相关联的另一条边的信息。

1. 实验心得

对于运用邻接表储存无向非连通图进行了简要的学习，并且运用和掌握了图中深度遍历和广度遍历，并做到打印简单路径不含有回路。

1. 进一步的改进和设想

后续会优化简单路径不含有回路的算法。