 实 验 报 告

课程名称： 数据结构（C++版）

题 目： 邻接表的实现

专业班级： 18空间

姓 名： 邓珊珊

学 号： 1812001107

指导老师： 袁莹

实验时间： 2019.12.13

第一次实验报告

班级：18空间 姓名：邓珊珊 学号：1812001107

1. 实验目的及原理

（选择公式的原理，及文字说明，所选的公式需要输入什么，计算出什么，如何计算，目的：通过实际问题学会用函数编程实现）

1. 实验目的
2. 掌握图的逻辑结构；
3. 掌握图的邻接表存储结构；
4. 验证图的邻接表存储及其遍历操作的实现。
5. 实验内容
6. 建立一个有向图的邻接表存储结构。
7. 建立一个有向图，进行深度优先遍历。
8. 对建立的有向图，进行广度优先遍历。
9. 实验提示

定义邻接表存储的有向图类ALGraph，包括题目要求的建立、深度优先遍历、广度优先遍历等基本操作。

1. 实验内容

（代码）

头文件ALGraph.h

#ifndef ALGraph\_H

//避免重复包含ALGraph.h头文件

#define ALGraph\_H

const int MaxSize=10;

//图的最大顶点数

//以下定义边表结点和顶点表结点

struct ArcNode

//定义边表结点

{

int adjvex;

//邻接点域

ArcNode \*next;

};

template<class DataType>

struct VertexNode

//定义顶点表结点

{

DataType vertex;

ArcNode \*firstedge;

};

//以下是类ALGraph的声明

template<class DataType>

class ALGraph

{

public:

ALGraph(DataType a[],int n,int e);

//构造函数

~ALGraph();

//析构函数

void DFSTraverse(int v);

//深度优先遍历图

void BFSTraverse(int v);

//广度优先遍历图

private:

VertexNode<DataType>adjlist[MaxSize];

//存放顶点表的数组

int vertexNum,arcNum;

//图的顶点数和边数

};

#endif

ALGraph.cpp:

#include<iostream>

//引入输入输出流

using namespace std;

#include"ALGraph.h"

//引入类ALGraph的声明

//以下是类ALGraph的成员函数定义

template<class DataType>

ALGraph<DataType>::ALGraph(DataType a[],int n,int e)

{

ArcNode \*s;

int i,j,k;

vertexNum=n;arcNum=e;

//存储顶点信息，初始化顶点表

for(i=0;i<vertexNum;i++)

{

adjlist[i].vertex=a[i];

adjlist[i].firstedge=NULL;

}

for(k=0;k<arcNum;k++)

//依次输入每一条边

{

cout<<"请输入边的两个顶点的序号:";

cin>>i>>j;

//输入边所依附的两个顶点的编号

s=new ArcNode;s->adjvex=j;

//生成一个边表结点s

s->next=adjlist[i].firstedge;

//将结点s插入到第i个边表的表头

adjlist[i].firstedge=s;

}

}

template<class DataType>

ALGraph<DataType>::~ALGraph()

{

ArcNode \*p=NULL;

//删除第i个边表

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

{

p=adjlist[i].firstedge;

while(p!=NULL)

{

adjlist[i].firstedge=p->next;

delete p;

//释放结点空间

p=adjlist[i].firstedge;

}

}

}

template<class DataType>

void ALGraph<DataType>::DFSTraverse(int v)

//深度优先遍历图

{

ArcNode \*p=NULL;int j;

cout<<adjlist[v].vertex;visited[v]=1;

p=adjlist[v].firstedge;

//工作指针p指向顶点v的边表

while(p!=NULL)

//依次搜索顶点v的邻接点j

{

j=p->adjvex;

if(visited[j]==0)DFSTraverse(j);

p=p->next;

}

}

template<class DataType>

void ALGraph<DataType>::BFSTraverse(int v)

//广度优先遍历图

{

int Q[MaxSize];

//假设队列采用顺序存储

int front=-1,rear=-1;

//初始化队列

ArcNode \*p=NULL;

cout<<adjlist[v].vertex;visited[v]=1;Q[++rear]=v;

//被访问顶点入队

while(front!=rear)

//当队列非空时

{

v=Q[++front];

p=adjlist[v].firstedge;

//工作指针p指向顶点v的边表

while(p!=NULL)

{

int j=p->adjvex;

//j是顶点v的邻接点

if(visited[j]==0){

cout<<adjlist[j].vertex;visited[j]=1;Q[++rear]=j;

}

p=p->next;

}

}

}

ALGraph\_main.cpp:

#include<iostream>

//引入输入输出流

using namespace std;

#include"ALGraph.cpp"

//引入类ALGraph的声明和成员函数定义

int visited[MaxSize]={0};

//标志数组visited[MaxSize]为全局变量

//以下为主函数

int main()

{

char ch[]={'A','B','C','D','E'};

//顶点信息

int i;

ALGraph<char>ALG(ch,5,6);

//图中顶点的类型为char型

for(i=0;i<MaxSize;i++)

//初始化图中所有顶点均未被访问

visited[i]=0;

cout<<"深度优先遍历序列是:";

ALG.DFSTraverse(0);

//从顶点0出发进行深度优先遍历

cout<<endl;

for(i=0;i<MaxSize;i++)

//初始化图中所有顶点均未被访问

visited[i]=0;

cout<<"广度优先遍历序列是:";

ALG.BFSTraverse(0);

//从顶点0出发进行广度有点遍历

cout<<endl;

return 0;

}