**第四部分 第2题： 最小生成树**

**实验报告**

题目：构建最小生成树并进行输出。

班级：F1703407 姓名：赵伟基 学号：517021910883 完成日期：2018-12-27

1. **需求分析**

若要在 n 个城市之间建设通信网络，只需要架设 n-1 条线路即可。如何以最低的经济代 价建设这个通信网，是一个网的最小生成树问题。本问题要求：

1. 利用Prim算法求网的最小生成树，并要求起始顶点由用户输入
2. 以文本形式输出生成树中各条边及其权值。
3. **概要设计**

本实验设计图的中最小生成树的建立，故需要定义类模板图（graph），并选择以邻接表的形式存储图，故在基类图的基础上派生邻接表类（adjListGraph）

1. 主程序模块

int main()

{

//手动输入模块

输入顶点；

输入图形各边；

完成最小生出树边的打印；

/\*测试数据模块\*/ 如果要查看测试数据直接将注释符号删除即可

}

1. 图的抽象类class graph
2. 数据成员：numOfVer 点数 ；numOfEdge 边数；
3. 成员函数：
   1. insert：在已有两点间插入边及其权值；
   2. remove：移除某边；
   3. exist：判断某边是否存在
4. 邻接表类的定义
   * 1. 数据成员：Vers 点数，Edges 边数；
     2. edgeNode 邻接表储存边的结点类

{ end终点编号； weight 边的权值； edgeNode \*next;}

* + 1. verNode 保存定点的数据元素类型

{ ver 顶点值； head 对应单链表的头指针}

* + 1. 成员函数
       1. adjListGraph（）：构造函数

邻接表初始化，输入字符顶点；

* + - 1. ~adjListGraph（）：析构函数

析构边表和结点空间；

* + - 1. insert（）:插入函数

初始条件：给定两定点和权值w

操作结果：构建两定点间权值为w的边

* + - 1. remove（）：删除函数

初始条件：给定两点

操作结果：清除两点间连线

* + - 1. exist（）：

初始条件：给定两点

返回结果：判断两点间是否有连线。

* + - 1. prim（）

初始条件：给定权值上限

返回结果; 输出最小生成树对应的边及其权值的文件。

1. **详细设计**
2. Head.h

#ifndef HEAD\_H

#define HEAD\_H

#include <iostream>

#include <cstring>

#include <fstream>

using namespace std;

//图的抽象类

template <class TypeOfVer,class TypeOfEdge>

class graph

{

public:

virtual void insert(TypeOfVer x,TypeOfVer y,TypeOfEdge w)=0;

virtual void remove(TypeOfVer x,TypeOfVer y)=0;

virtual bool exist(TypeOfVer x,TypeOfVer y)const =0;

int numOfVer()const {return Vers;}

int numOfEdge()const {return Edges;}

protected:

int Vers,Edges;

};

//邻接表类的定义

template <class TypeOfVer,class TypeOfEdge>

class adjListGraph:public graph<TypeOfVer,TypeOfEdge>

{

public:

adjListGraph(int vSize,const TypeOfVer d[]);

void insert(TypeOfVer x,TypeOfVer y,TypeOfEdge w);

void remove(TypeOfVer x,TypeOfVer y);

bool exist(TypeOfVer x,TypeOfVer y)const;

~adjListGraph();

void prim(TypeOfEdge noEdge)const;

//void dfs()const;

//void bfs()const;

protected:

int Vers,Edges;

private:

struct edgeNode //邻接表中存储边的结点类

{

int end; //终点编号

TypeOfEdge weight;//边的权值

edgeNode \*next;

edgeNode(int e,TypeOfEdge w,edgeNode \*n=NULL)

{

end=e;//边的终点

weight=w;//权值

next=n;

}

};

struct verNode //保存顶点的数据元素类型

{

TypeOfVer ver;//顶点值

edgeNode \*head;//对应的单链表的头指针

verNode(edgeNode \*h=NULL){head=h;}

};

verNode \*verList;//指向保存顶点的数组的首地址

int find(TypeOfVer v)const

{

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

if(verList[i].ver==v) return i;

}

// void dfs(int start,bool visited[])const;

};

//adjListGraph类的构造函数和析构函数

template <class TypeOfVer,class TypeOfEdge>

adjListGraph<TypeOfVer,TypeOfEdge>::adjListGraph(int vSize,const TypeOfVer d[])

{

Vers=vSize;

Edges=0;

verList=new verNode [vSize];

for(int i=0;i<Vers;i++) verList[i].ver=d[i];//顶点数组赋值

}

template <class TypeOfVer,class TypeOfEdge>

adjListGraph<TypeOfVer,TypeOfEdge>::~adjListGraph()

{

int i;

edgeNode\* p;

for(i=0;i<Vers;i++)//析构边表

{

while((p=verList[i].head)!=NULL)//释放第i个结点的单链表

{

verList[i].head=p->next;

delete p;

}

}

delete []verList;//析构结点表

}

//insert函数的实现

template <class TypeOfVer,class TypeOfEdge>

void adjListGraph<TypeOfVer,TypeOfEdge>::insert(TypeOfVer x,TypeOfVer y,TypeOfEdge w)

{

int u=find(x);//例如寻找A是第几个结点

int v=find(y);

verList[u].head=new edgeNode(v,w,verList[u].head);//从头部插入，一行代码实现三个功能：1.创建2.加入边表3.改结点表中的head指针

++Edges;

}

//remove函数的实现

template <class TypeOfVer,class TypeOfEdge>

void adjListGraph<TypeOfVer,TypeOfEdge>::remove(TypeOfVer x,TypeOfVer y)

{

int u=find(x),v=find(y);

edgeNode \*p=verList[u].head,\*q;

if(p==NULL) return;//结点u没有相连的边

if(p->end==v) //单链表中的第一个结点就是被删除的边

{

verList[u].head=p->next;

delete p;

--Edges;

return;

}

while(p->next!=NULL&&p->next->end!=v) p=p->next;//查找被删除的边

if(p->next!=NULL)//删除

{

q=p->next;

p->next=q->next;

delete q;

--Edges;

}

}

//exist函数的实现

template <class TypeOfVer,class TypeOfEdge>

bool adjListGraph<TypeOfVer,TypeOfEdge>::exist(TypeOfVer x,TypeOfVer y)const

{

int u=find(x),v=find(y);

edgeNode \*p=verList[u].head;

while(p->next!=NULL&&p->end!=v) p=p->next;//查找

if(p==NULL) return false;

else return true;

}

//prim算法

template <class TypeOfVer, class TypeOfEdge>

void adjListGraph<TypeOfVer, TypeOfEdge>::prim(TypeOfEdge noEdge) const

{

ofstream fout("primtree.txt");

bool \*flag = new bool[Vers];

TypeOfEdge \*lowCost = new TypeOfEdge[Vers];

int \*startNode = new int[Vers];

edgeNode \*p;

TypeOfEdge min;

int start,i, j;

char start1;

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

{

flag[i] = false;

lowCost[i] = noEdge;

}

cout<<"请输入起始顶点"<<endl;

cin>>start1;

start=find(start1);

for ( i= 1; i < Vers; ++i) {

for (p = verList[start].head; p != NULL; p = p->next)

if (!flag[p->end] && lowCost[p->end] > p->weight) {

lowCost[p->end] = p->weight;

startNode[p->end] = start; }

flag[start] = true;

min = noEdge;

for (j = 0; j < Vers; ++j)

if (lowCost[j] < min) {min = lowCost[j]; start = j;}

cout << '(' << verList[startNode[start]].ver << ','<< verList[start].ver << ") weight:"<<lowCost[start]<<endl;

fout << '(' << verList[startNode[start]].ver << ','<< verList[start].ver << ") weight:"<<lowCost[start]<<endl;

lowCost[start] = noEdge;

}

fout.close();

delete [] flag;

delete [] startNode;

delete [] lowCost;

}

#endif // HEAD\_H

1. Main.cpp

#include <iostream>

#include <ctime>

#include <cstdlib>

#include "head.h"

using namespace std;

int main()

{

//手动输入模块

/\* char s[30];

char str1,str2;

int weight,len;

cout<<"please input the points(example:abcdefg)"<<endl;

cin>>s;

adjListGraph<char,int> g(strlen(s),s);

cout<<"please input the number of edges "<<endl;

cin>>len;

srand(time(0));

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

{

cout<<"please insert the edges :point1 point2 (example: a b )"<<endl;

cin>>str1>>str2;

weight=rand()%100;

g.insert(str1,str2,weight);

g.insert(str2,str1,weight);

}

g.prim(100);\*/

//测试数据

adjListGraph<char,int> g(8,"abcdefgh");

g.insert('a','b',4);

g.insert('a','c',3);

g.insert('b','c',5);

g.insert('b','d',5);

g.insert('b','e',9);

g.insert('c','d',5);

g.insert('c','h',5);

g.insert('d','e',7);

g.insert('d','f',6);

g.insert('d','g',5);

g.insert('d','h',4);

g.insert('e','f',3);

g.insert('f','g',2);

g.insert('g','h',6);

g.insert('b','a',4);

g.insert('c','a',3);

g.insert('c','b',5);

g.insert('d','b',5);

g.insert('e','b',9);

g.insert('d','c',5);

g.insert('h','c',5);

g.insert('e','d',7);

g.insert('f','d',6);

g.insert('g','d',5);

g.insert('h','d',4);

g.insert('f','e',3);

g.insert('g','f',2);

g.insert('h','g',6);

g.prim(100);

return 0;

}

1. **调试分析**
2. 该项目对邻接表类进行了很好的包装，在保障数据的安全性同时具有很高的可移植性。
3. 该算法未对用户界面进行很好的设计，只满足了实验的要求
4. 算法复杂度分析

（1）：时间复杂度

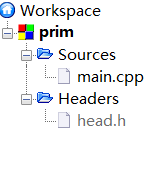
1. O(n):find(); adjListGraph();remove();exist();
2. O(n^2):~adjListGraph();prim();
3. O(1):insert();

（2）：空间复杂度

O(n): prim();remove();insert();adjListGraph();

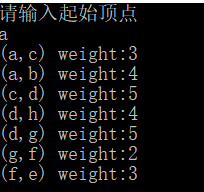
1. **用户手册**

1．本程序使用的 Code::Blocks 16.01 IDE，程序以项目（project）方式组织，如图 1 所示



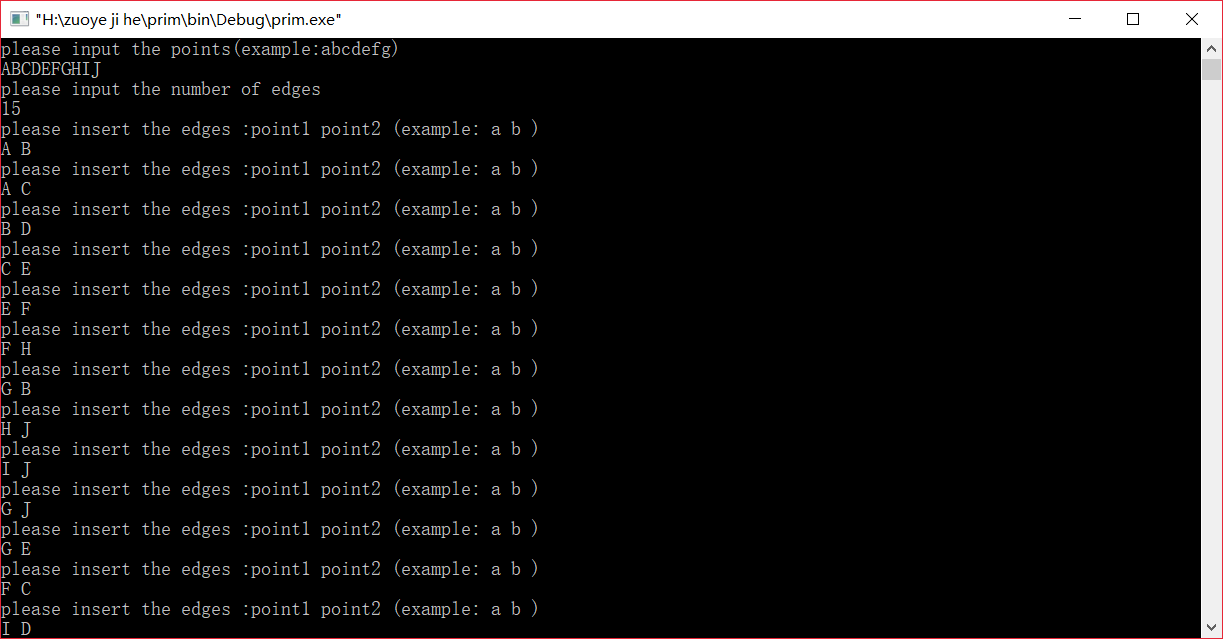
2． 用户点击main.cpp->build and run,手动按照交互提示输入数据即可。

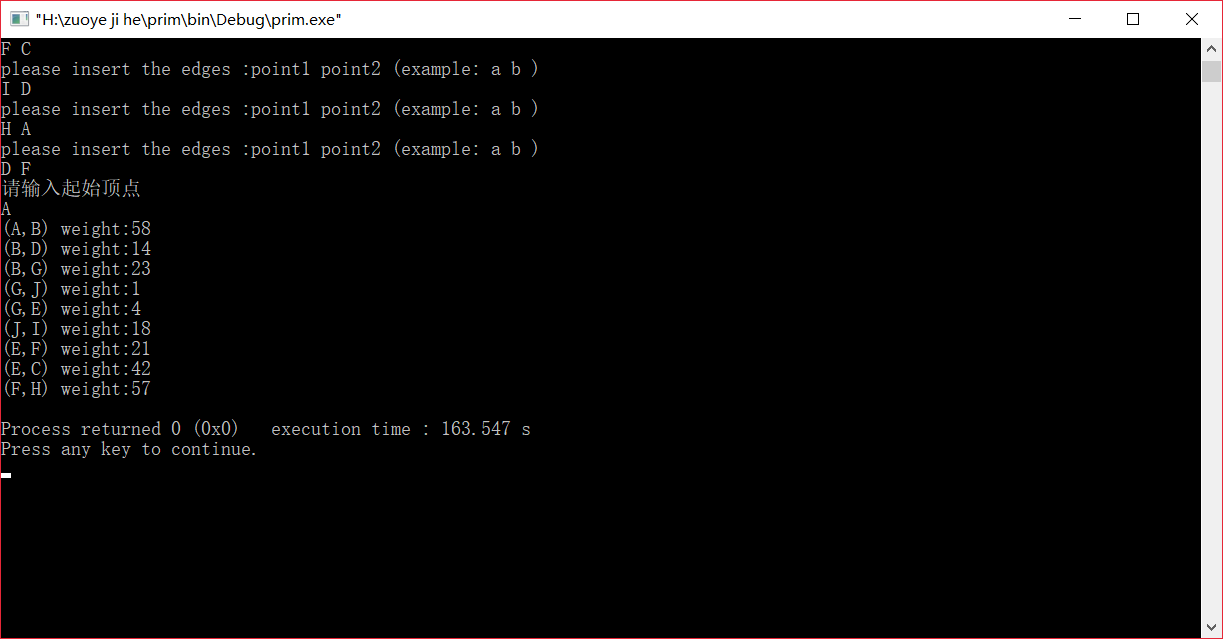
1. **测试结果**
2. 依照测试数据

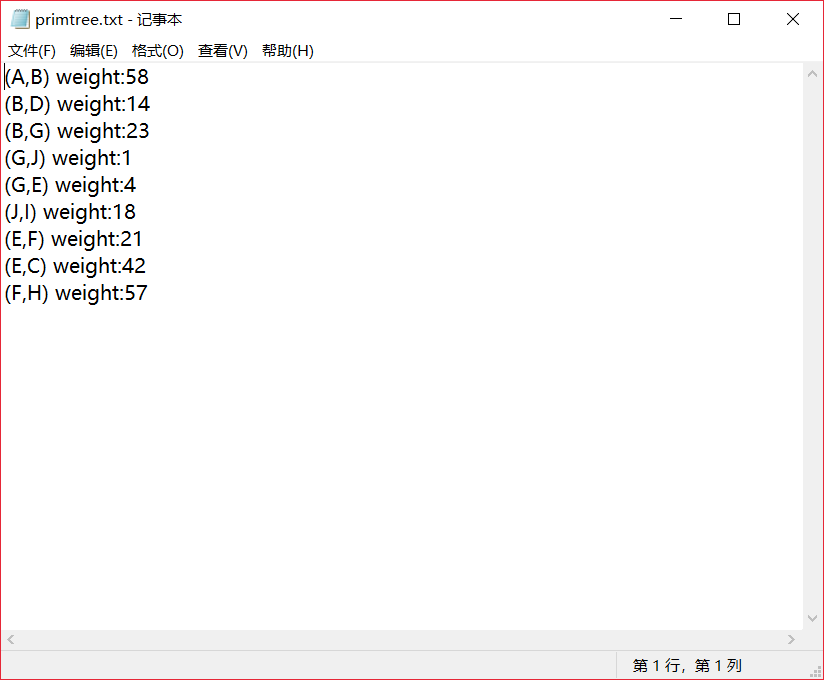


1. 手动输入数据

依照图片所示输入数据，即可即可完成随机权值构建图。并输出最小生成树至文件“primtree”中去。







1. **附录**

main.cpp //主函数 功能函数模块

head.h //头文件 定义函数