数据结构与算法导论

实验报告

实验题目：\_\_\_\_\_\_图 基本实验\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

姓 名：\_\_\_\_\_\_冯韵菱\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

学 号：\_\_\_\_\_\_\_2018210407\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

日 期：\_\_\_\_\_\_\_2019.5.15\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

自我评分：\_\_\_\_\_\_\_\_\_【 A 】\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

自我评分说明：A+，A，B+，B，B-，C，D，分别对应分数95、90、85、80、75、70、60

诚信声明

本人郑重承诺：本实验程序和实验报告均是本人独立学习和工作所获得的成果。尽我所知，实验报告中除特别标注的地方外，不包含其他同学已经发表或撰写过的成果；实验程序中对代码工作的任何帮助者所作的贡献均做了明确的说明，并表达了谢意。

如有抄袭，本人原因承担因此而造成的任何后果。

特此声明。

签名：\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

日期：\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

程序引用说明

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 引用项 | 来源 | 相同代码行数 |
| 1 | BFS | 课本p172 | 27 |
| 2 | 析构函数 | 课本p173 | 10 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
| 小计 | | | 37 |

总代码行数\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_515\_\_\_\_\_; 引用占比\_\_\_7.1％\_\_\_\_\_\_\_\_\_\_\_\_\_

1、实验简介

【实验内容的简要说明，具体说明实验完成的功能和性能要求】

根据图的抽象数据类型的定义，使用邻接矩阵或邻接表实现一个图。

图的基本功能：

1、图的建立

2、图的销毁

3、深度优先遍历图

4、广度优先遍历图

5、使用普里姆算法生成最小生成树

6、使用克鲁斯卡尔算法生成最小生成树

7、求指定顶点到其他各顶点的最短路径

8.编写测试main()函数测试图的正确性：

9.使用非递归方式（stack）编写新的深度优先遍历函数

10、最短路径D算法优化

2、程序框架

【实验程序共包含哪些函数，一一列举函数名和函数功能】

ALGraph(); //两种构造 一种自己输入 一种使用已输入数据测试

ALGraph (int x,int y, char a[],int arcs[][3]); //都是利用的邻接表 建立

~ALGraph();

void DFS (int v); //主调的深搜

void dfs(); //深搜主体，使用stack进行非递归实现遍历图

void BFS (int v); //主调的广搜

void bfs(); //广搜主体，利用队列遍历

void Prim(); //普利姆，不断选择最小的点并加入

int mininum(int lowcost[]); //普利姆的辅助函数，寻找Lowcost中的最小值

void Kruskal(); //克鲁斯卡尔，不断寻找最小的边 并加入

void GenSortEdge(); //利用直接插入排序 将边进行赋值并排序

void shortest(); //寻找最短路径

int GetMin(int\* Length, bool\* ifadded); //第一种 改进前

int GetMin2(int\* Length, bool\* ifadded); //改进后 速度更快

void PrintPath(int \* Length, int\*Path,int origin,int end); //递归打印最短路径

3、关键代码实现

【哪些函数是你认为最能体现自己工作成果的函数，说明函数实现基本思想（可用文字或图表示），以及具体的实验步骤（用伪代码或带注释代码）】

3.1 DFS

基本思想：利用stack 和辅助数组，遍历图

实验步骤：

template <class T>

void ALGraph<T> :: DFS(int v) //改成了非递归的 可以防止溢出

{

stack<int> a;

cout << adjlist[v].vertex; //加入第一个顶点

visited[v] = true;

a.push(v);

while (!a.empty()) //只要栈不空 就一直循环

{

ArcNode\* temp = adjlist[a.top()].first; //指向栈顶元素

bool ifad = true; //辅助判断 该栈顶元素是否 还有未加入的点

while (temp)

{

if (visited[temp->adjvex] == 0) //如果有未加入的点 则输出、入栈、并改为已经访问过

{

cout << "->" << adjlist[temp->adjvex].vertex;

a.push(temp->adjvex);

visited[temp->adjvex] = 1;

ifad = false;

break;

}

temp = temp->next;

}

if (ifad) //如果该点已经没有未访问过的节点，则栈顶元素出栈

{

if (!a.empty())

a.pop();

else

return;

}

}

}

3.2 Prim

基本思想：先建立一个辅助数组，再加入一个点，统计该点到各点的距离，再加入一个距离最近的点，并以一个集合的形式重新刷新该集合到其他点的最小距离

具体实现：

template <class T>

void ALGraph<T>:: Prim()

{

for (int i = 0;i < vNum;i++) //先为数组赋好初值

{

lowcost[i] = -1;

adjvex[i] = 0;

}

cout<< "Prim(The order of adding adjvexes):" << endl;

ArcNode\* temp = adjlist[0].first;

while(temp!=NULL) //加进来第一个点

{

lowcost[temp->adjvex] =temp->weight ;

temp = temp->next;

}

lowcost[0] = 0;

cout << "The added adjvex：" << adjlist[0].vertex << endl;

for (int i = 1;i < vNum;i++) //逐渐加入所有的点

{

int k = mininum(lowcost);

cout<<"The added adjvex："<< adjlist[k].vertex << endl;

lowcost[k] = 0; //包含入k点

temp = adjlist[k].first;

while(temp) //遍历完k点的所有邻接点

{

if ((temp->weight <lowcost[temp->adjvex]|| lowcost[temp->adjvex]==-1)&& lowcost[temp->adjvex] != 0)

{

lowcost[temp->adjvex] = temp->weight; //如果这个点到集合的值有更小值 则刷新数组

temp->adjvex = k;

}

temp = temp->next;

}

}

}

3.5 shortest

基本思路 找到此点到其他点的路径，并选一条最短的，再通过这条找到新的点的最短路径，最后通过递归打印路径即可

具体实现：

template <class T>

void ALGraph<T>:: shortest() //复杂度应该为nlogn 因为肯定得遍历一次 所以在寻找最小值上边做了优化

{

int origin, i = 0,end;

cout << '\n' << "To get the shortest way:" << endl;

cout << "Please put in the started vertex and the end vertex you wanted: " << endl;

cin >> origin>>end;

cout << "Please choose the version you wanted:" << endl;

cout << "You can compared the speed yourself!!(numeber 1 for old version 2 for improved version) :" << endl;

int way;

cin >> way;

QueryPerformanceCounter(&t3); //程序计时

bool\* ifadded = new bool[vNum];

int \*Length = new int[vNum];

int\* Path = new int[vNum];

for (i = 0;i < vNum;i++) //初始化 长度为无限 未访问过

{

Length[i] = 999999;

ifadded[i] = true;

}

ArcNode\*temp = adjlist[origin].first; //加入起始节点

while (temp)

{

Length[temp->adjvex] = temp->weight; //遍历起始节点 加入最短路径 和路线

Path[temp->adjvex] = origin;

temp = temp->next;

}

ifadded[origin] = false; //设置为访问过

int k = origin;

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

{

if(way==1)

k = GetMin(Length, ifadded); //获取最近的点

else

k = GetMin2(Length, ifadded);

if (k == -1)

break;

temp = adjlist[k].first;

while (temp) //用最近的点刷新最短路径和最近路线

{

if (temp->weight + Length[k] < Length[temp->adjvex])

{

Length[temp->adjvex] = temp->weight + Length[k];

Path[temp->adjvex] = k;

}

temp = temp->next;

}

ifadded[k] = false;

}

cout << '\n'<<'\n'<<"The shorted way is:" << endl;

PrintPath(Length, Path, origin, end);

cout << adjlist[end].vertex<<endl<<endl; //下面的递归不方便输出终点

QueryPerformanceCounter(&t4); //程序计时

dt = (t4.QuadPart - t3.QuadPart) / (long double)nFreq.QuadPart;

cout << '\n' << setprecision(12) << "The running time of your program:" << dt << "s" << endl;

}

3.5 GetMin2

基本思路 利用队列，改进了寻找最小值函数，建立优先级队列 使时间复杂度变为了logn

具体实现：

struct shor

{

int poi; //数组下标位置

int weight;

};

struct cmp //定义比较算法

{

bool operator()(shor a, shor b)

{

return a.weight > b.weight; //小根堆

}

};

template <class T>

int ALGraph<T>::GetMin2(int\* Length, bool\* ifadded)

{

std::priority\_queue<shor, vector<shor>, cmp>a; //建立优先队列 使时间复杂度变为logn

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

{

if (ifadded[i]) //如果没有访问过， 入队

{

shor temp;

temp.poi = i;

temp.weight = Length[i];

a.push(temp);

}

}

if (!a.empty()) //栈如果非空 就返回位置

return a.top().poi;

else

return -1;

}

3.5 PrintPath

基本思路 利用递归，不断寻找该点的上一个

具体实现：

void ALGraph<T>::PrintPath(int \* Length,int\*Path,int origin,int end)

{

if (end == origin) //递归画图 如果起点等于终点 回溯

return;

else

{

end = Path[end]; //如果不是，到上一个顶点

PrintPath(Length, Path, origin, end);

cout<< adjlist[end].vertex << "->";

}

}

4、不足

【实验程序哪些函数功能还有缺陷或不足，或者程序架构有不足，或者性能还有待提高、或者代码不和规范等等，所有你自己对程序不满意的地方】

本来想在输入方面加入一些优化的，就是先让用户选择输入的为有向图还是无向图，之后在输入无向图的时候就会方便不少，但奈何时间有限没能修改

5、心得体会

【所有你在实验中的感受和想和老师说的话都可以放在这里，篇幅不限】

这次实验感觉写递归的能力在前面的的锻炼中已经有所提高，较短时间就能写出一个以前要花很长时间写的递归函数了，还有就是这次比较多的使用了STL,发现STL真的是非常好用，但我也直到肯定不能依赖它，我都是先自己写了一个函数，再用stl来改进的，同样，感觉每做一次实验能力都提升了不少。