Project 4

中山大学校园导游系统

班级： 软工5班

盘学之 16340177

谭发豪 16340202

苏泽华 16340195

**【题目要求】**

1. 从中山大学东校区的平面图中选取有代表性景点（10-15个），抽象成一个无向带权图。以图中顶点表示校内各景点，存放景点名称、代号，简介等信息；以边表示路径，存放路径长度等信息；
2. 为来访客人提供图中任意景点相关信息的查询；
3. 为来访客人提供图中任意景点的问路查询，即查询任意两个景点之间的一条最短的简单路径；
4. 区分汽车线路与步行线路；

**【数据结构与算法】**

数据结构：

typedef struct vertex { //构造地图节点的结构

int number; //节点的编号

string name; //节点的名字

string info; //节点的相关信息

bool is\_scene; //是否景点

int x;

int y; //点的坐标

vertex(int n = 0, string str = "", string i = "", bool s = true) { //初始化节点

number = n;

name = str;

info = i;

is\_scene = s;

}

} vertex;

typedef struct edge { //构造道路的结构

int weight; //权重(长度)

bool is\_pavement; //是否人行道

bool is\_driveway; //是否车道

edge(int w = 0, bool p = false, bool d = false) { // 初始化道路

weight = w;

is\_pavement = p;

is\_driveway = d;

}

} edge;

class map { //构造地图的类

public:

map(): vertex\_num(0), edge\_num(0) { //构造函数

for(int i = 0; i < MAX\_VERTEX; i++) {

for(int j = 0; j < MAX\_VERTEX; j++) {

adjacency\_matrix[i][j] = nullptr;

}

vertex\_list[i] = nullptr;

}

}

~map() { //析构函数

for(int i = 0; i < vertex\_num; i++) {

for(int j = i; j < vertex\_num; j++) {

if(adjacency\_matrix[i][j] != nullptr) {

delete adjacency\_matrix[i][j];

delete adjacency\_matrix[j][i];

}

}

if(vertex\_list[i] != nullptr)

delete vertex\_list[i];

}

}

bool ReadFromFile(string url) { //文件读取函数，从文件中读入地图信息

// 打开数据文档

ifstream file;

file.open(url);

//打开文档失败，返回false

if(!file)

return false;

//处理数据并存储，最终返回true

string temp = "";

int pos1 = 0;

int pos2 = 0;

file >> temp >> vertex\_num >> temp;

for(int i = 0; i < vertex\_num; i++) {

vertex\_list[i] = new vertex;

file >> vertex\_list[i]->number;

getline(file, temp);

getline(file, vertex\_list[i]->name);

getline(file, vertex\_list[i]->info);

file >> vertex\_list[i]->is\_scene;

}

file >> temp >> edge\_num >> temp;

for(int i = 0; i < edge\_num; i++) {

file >> pos1 >> pos2;

adjacency\_matrix[pos1][pos2] = new edge;

file >> adjacency\_matrix[pos1][pos2]->weight

>> adjacency\_matrix[pos1][pos2]->is\_pavement

>> adjacency\_matrix[pos1][pos2]->is\_driveway;

adjacency\_matrix[pos2][pos1] = adjacency\_matrix[pos1][pos2];

}

file.close();

return true;

}

bool WriteToFile(string url) { //

// 打开数据文档

ofstream file;

file.open(url);

// 写入数据

file << "vertex\_num: " << vertex\_num << endl << endl

<< "vertex\_list(number,name,info,scene?):" << endl << endl;

for(int i = 0; i < vertex\_num; i++) {

file << vertex\_list[i]->number << endl

<< vertex\_list[i]->name << endl

<< vertex\_list[i]->info << endl

<< vertex\_list[i]->is\_scene << endl << endl;

}

file << "edge\_num: " << edge\_num << endl << endl

<< "edge\_list(vertex1,vertex2,weight,pavement?driveway?):" << endl;

for(int i = 0; i < vertex\_num; i++) {

for(int j = i; j < vertex\_num; j++) {

if(adjacency\_matrix[i][j] != nullptr) {

edge\* pointer = adjacency\_matrix[i][j];

file << i <<" "<< j <<" "<< pointer->weight <<" "

<< pointer->is\_pavement <<" "<< pointer->is\_driveway

<< endl;

}

}

}

file.close();

return true;

}

int getX(int num)

{

return vertex\_list[num]->x;

}

int getY(int num)

{

return vertex\_list[num]->y;

}

string getInfo(int num)

{

return vertex\_list[num]->info;

}

vertex\* getP(int num)

{

return vertex\_list[num];

}

//得到点的内部信息

int get\_number(const string& name) {

for(int i = 0; i < vertex\_num; i++) {

if(vertex\_list[i]->name == name) {

return i;

}

}

return -1;

}

// 传入点的名字，返回点的编号；若点不存在，返回-1

private:

//const int MAX\_VERTEX = 50; //最大结点数量

int vertex\_num; //结点数量

int edge\_num; //边数量

// 邻接矩阵，两点相通则有对应位置的edge指针有值，其余全为 nullptr

edge\* adjacency\_matrix[MAX\_VERTEX][MAX\_VERTEX];

// 结点数组，有新的点则new一个新值

vertex\* vertex\_list[MAX\_VERTEX];

};

算法:

关于Dijkstra算法：

find\_all\_path() 函数用dijkstra算法来寻找某个结点去其它结点的全部最短路径，返回一个vector<vector<int> >，外层vector储存每条路径，而一条路径就是一个 vector<int>，储存路径的结点顺序，比如从0到3是：0 1 8 6 3。另外，因为人行道和车道是分开的，在找路径时会传入一个judge参数，用来判断找哪种最短路径，若找的是车道，将忽略所有非车道的边的权重。

Dijkstra算法开始前，先假设所有路径是直接从起点到终点，比如0->1; 0->2; 0->3;然后再用一个distance数组来储存每条路径的距离长度，当然起点和终点不一定一开始就能连在一起，比如如果0和2不相连，那么0->2的距离就是max。另外，再用一个visited数组来标记哪些结点已经找到最短路径。

算法开始后，进行n-1次循环（n为结点数量，因为起点相当于已经找到最短路径，因而要找路径的结点是n-1个）。每次循环，找出distance数组中最小的一个值，也就是找出起点到其它点的路径中最短的一条，并将其确定为对应终点的最短路径。

然后，用这条新找到的最短路径来“松弛”起点到其它终点的路径。比如说，在某一次循环中，我确定了起点0到2的最短路径为：0->1->2，然后我就用这条新的最短路径代入其它路径，原来0到4的路径就是0->4，现在尝试用0->1->2->4路径来松弛，发现0->1->2->4路径比0->4还短，这种情况即为“松弛”成功，那么更新0到4的路径为0->1->2->4，并更改distance数组中的距离数值。

总之，在这次循环中，就成功将结点2 visited，即找到了2的最短路径，还顺便缩短了其它最短路径的距离（不过还不能确定最终结果）。

最终得到所有路径，并存入vector<vector<int>> allPath 之后，返回容器，即得到所有最短路径。这就是Dijkstra的全过程。

关于图像处理：

原本计划是用全自动的方法从图中提取特征从而实现较为精确地描绘地图，然而较为复杂的问题令该计划落空。

示例图：

该图为采用色彩的二值化后得到的，较为精确地描述了道路网络结构。



该图为提取角点的结果，由于数量太大导致了无法准确地去处理

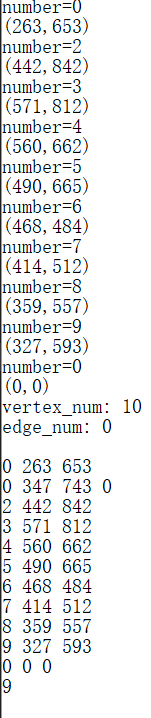


该图为二值化图像矢量化后的结果

图像的机器化处理并不能作出一张完整的图，所以python方面用来手动标注点和线，从而生成原始的图的数据

**【测试数据、结果及分析】**

（所输入的数据及相应的运行结果，运行结果要有提示信息，运行结果采用截图方式给出。）



这个测试是用来测试样例的数据是否正确和程序是否工作正常。

**【分工、贡献%、自我评分】**

分工：

盘学之：图像处理、调试、优化代码，Python方面

谭发豪：底层开发、调试、优化代码，Native C++

苏泽华：UI设计、撰写报告、优化代码,Qt

贡献：

盘学之：33.3%

谭发豪：33.3%

苏泽华：33.3%

自我评分：

盘学之：95

谭发豪：80

苏泽华：85

**【项目总结】**

（收获、体会，若实验课上未完成调试，要认真找出错误并分析原因等。）

盘学之

这次的作业就是比较艰难的一次联合开发，虽然要求比较少但是可以做拓展的地方非常多。虽然这次的代码质量不算很高，但是还是基本上做到了预期的目标。

苏泽华

一开始我原本想用API来做UI的，因为老师不断和我们说“不要造轮子”，但后来想着想给自己一些挑战，学习一些新的东西，所以还是选择了自己用QT来做。QT我们基本是从0开始学的，遇到了非常多的困难，而且与web相比，QT让人感觉非常迷，大概是因为我还没有系统地读过QT的文档吧，以后有时间一定要好好读一读。不过综合下来，本次实验还是很有收获的。快期末了，还有Project5吗？不好吧。。。。

谭发豪：

本次实验我负责底层C++程序，也就是写图的相关数据结构和功能函数。在这次实验中，我觉得我的参与度并不算高，图的数据结构代码比较简单，我也没有实现更多的功能函数，比如从一个节点出发如何最短距离走完其它所有景点，这算是个小小的遗憾。不过这次实验也让我对图的理解更加深刻了，比如总算完全弄懂了Dijkstra算法，这算是个较大的进步。

**【程序清单**】

由于word的缩进与高亮不适用于代码展示，源代码详请见附件