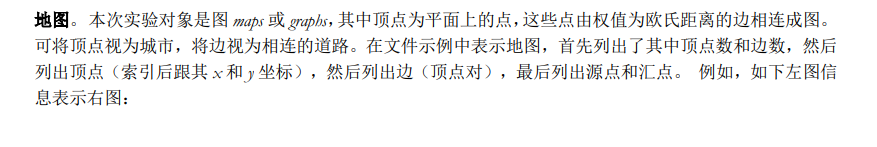
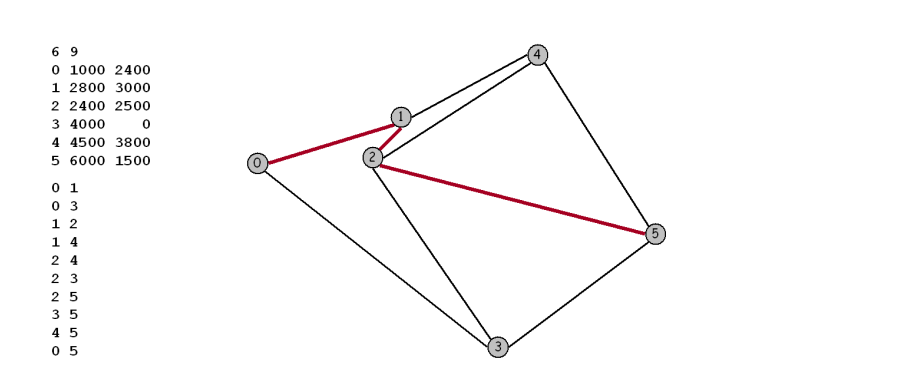
**实验三 地图路由**

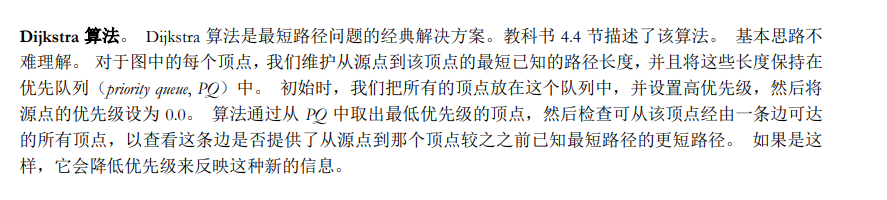
1. 实验目的

实现经典的 Dijkstra 最短路径算法，并对其进行优化。 这种算法广泛应用于地理信息系统（GIS），包括 MapQuest 和基于 GPS 的汽车导航系统。

1. 实验内容







1. 代码实现

代码如下：

#include <bits/stdc++.h>

#include <fstream>

#include <time.h>

using namespace std;

*//改善思路为想法2，使用A\*算法，改善性能*

const size\_t MAX\_Point\_Number = 87575;

const size\_t MAX\_Edge\_Number = 125000;

const double eps = 1e-6;

class Point {

private:

    int x;

    int y;

    int num;

public:

    Point() {}

    Point(int xa, int ya, int numa) :x(xa), y(ya), num(numa) {

    }

    int getx() const { return x; }

    int gety() const { return y; }

    int getnum() const { return num; }

    void setx(int x) { this->x = x; }

    void sety(int y) { this->y = y; }

    void setnum(int num) { this->num = num; }

    double dist(Point p1) {

        int x2 = (p1.getx() - x) \* (p1.getx() - x);

        int y2 = (p1.gety() - y) \* (p1.gety() - y);

        return sqrt(x2 + y2);

    }

};

*//创建边的数据结构*

*/\*class Edge {*

*private:*

*int from;*

*int to;*

*public:*

*Edge(int f, int t) :from(f), to(t) {};*

*int getfrom() const { return from; }*

*int getto() const { return to; }*

*double dist(Point p1, Point p2) {*

*int x2 = (p1.getx() - p2.getx()) \* (p1.getx() - p2.getx());*

*int y2 = (p1.gety() - p2.gety()) \* (p1.gety() - p2.gety());*

*return sqrt(x2 + y2);*

*}*

*};\*/*

*//定义一些必要的数据结构*

typedef struct LI {

    Point p;

    struct LI\* next;

}ListNode;

class MyDist {

public:

    int number;

    double dist;

};

*//用于优先队列的比价函数*

struct cmp

{

    bool operator()(MyDist a, MyDist b) {

        return a.dist > b.dist;

    }

};

void ChangeDist(priority\_queue<MyDist, vector<MyDist>, cmp>& pq, MyDist md) {

    vector<MyDist> vec;

    bool flag = false;

    while (!pq.empty()) {

        if (md.number != pq.top().number) {

            vec.push\_back(pq.top());

            pq.pop();

        }

        else {

            flag = true;

            pq.pop();

            vec.push\_back(md);

            break;

        }

    }

    if (!flag) {

        vec.push\_back(md);

    }

    for (int i = 0; i < vec.size(); i++) {

        pq.push(vec[i]);

    }

}

ListNode\* L[MAX\_Point\_Number];

double dist[MAX\_Point\_Number];

bool used[MAX\_Point\_Number];

void calculate(int start, int x) {

    set<int> myset; *//用于结果路径的输出*

    double sum = 0;

    int i = start;

    int cnt = 0;

*//创建优队列*

    priority\_queue<MyDist, vector<MyDist>, cmp> pq;

    MyDist md;

    md.dist = 0.0;

    md.number = start;

    pq.push(md);

    while (!pq.empty()) {

        MyDist disttmp = pq.top();

        pq.pop();

        i = disttmp.number;

        used[i] = true;

        if (i == x) break;

        ListNode\* tmp;

        for (tmp = L[i]->next; tmp != NULL; tmp = tmp->next) {

            int n = tmp->p.getnum();

            if (used[n]) continue;

*//下面使用A\*算法来进行优化*

            MyDist disttmp1;

            dist[n] = min(dist[i] + L[i]->p.dist(tmp->p), dist[n]);

            if (dist[n] > dist[i] + L[i]->p.dist(tmp->p))

*//如果需要更新距离，就将优先队列中的距离数据更新为如下内容*

                disttmp1.dist = dist[i] + L[i]->p.dist(tmp->p);

*//disttmp1.dist = dist[i] + L[i]->p.dist(tmp->p) + L[n]->p.dist(L[x]->p) - L[i]->p.dist(L[i]->p);*

            else

                disttmp1.dist = dist[n];

            disttmp1.number = n;

            ChangeDist(pq, disttmp1);

        }

    }

    cout << "最短路径为：" << dist[x] << endl;

}

int main()

{

    int ex;

    cout << "请输入：" << endl;

    while (1) {

        memset(dist, 0x42, sizeof(dist));

        memset(used, false, sizeof(used));

        int n, v, e, x, y, num, m;

*/\**

*输入文件中的内容*

*\*/*

        ifstream in("C:\\Users\\lenovo\\Desktop\\usa.txt", ios::in);

        if (!in.is\_open())

        {

            cerr << "open error!" << endl;

            exit(0);

        }

        in >> v >> e;

        for (int i = 0; i < v; i++) {

            in >> num >> x >> y;

            L[i] = new ListNode;

            L[i]->next = NULL;

            L[i]->p.setnum(num);

            L[i]->p.setx(x);

            L[i]->p.sety(y);

        }

        for (int i = 0; i < e; i++) {

            in >> x >> y;

            ListNode\* tmp1 = new ListNode;

            ListNode\* tmp2;

            tmp1->next = NULL;

            tmp1->p = L[y]->p;

            for (tmp2 = L[x]; tmp2->next != NULL; tmp2 = tmp2->next) {

            }

            tmp2->next = tmp1;

            ListNode\* tmp3 = new ListNode;

            ListNode\* tmp4;

            tmp3->next = NULL;

            tmp3->p = L[x]->p;

            for (tmp4 = L[y]; tmp4->next != NULL; tmp4 = tmp4->next) {

            }

            tmp4->next = tmp3;

        }

        vector<int> a;

        cout << "请输入起点和终点：" << endl;

        cin >> m >> n;

        dist[m] = 0.0;

        clock\_t time\_start, time\_end;

        time\_start = clock();

        calculate(m, n);

        time\_end = clock();

*/\*cout << "最短路径顺序为：" << endl;*

*for (int i = a.size() - 1; i >= 0; i--) {*

*cout << a[i];*

*if (i > 0)*

*cout << "-->";*

*else*

*cout << endl;*

*}\*/*

        cout << "本次最短路径计算耗时为：" << (time\_end - time\_start) \* 1000 / CLOCKS\_PER\_SEC <<"ms" << endl;

        cout << "输入0以结束输入，输入1继续" << endl;

        cin >> ex;

        if (ex == 0) {

            cout << "结束输入！" << endl;

            return 0;

        }

        else {

            cout << "请继续输入:" << endl;

        }

    }

    return 0;

}

首先，过大的数据量以及过于稀疏的路径信息导致我们很难用邻接矩阵来表示点与点之间的路径，因为这样不但会非常浪费空间，同时也会给我们的算法设计带来很大的困难，这也同样会影响最后的性能。

关于对迪杰斯特拉算法的改进，我最终选择顺着想法2来进行改良：

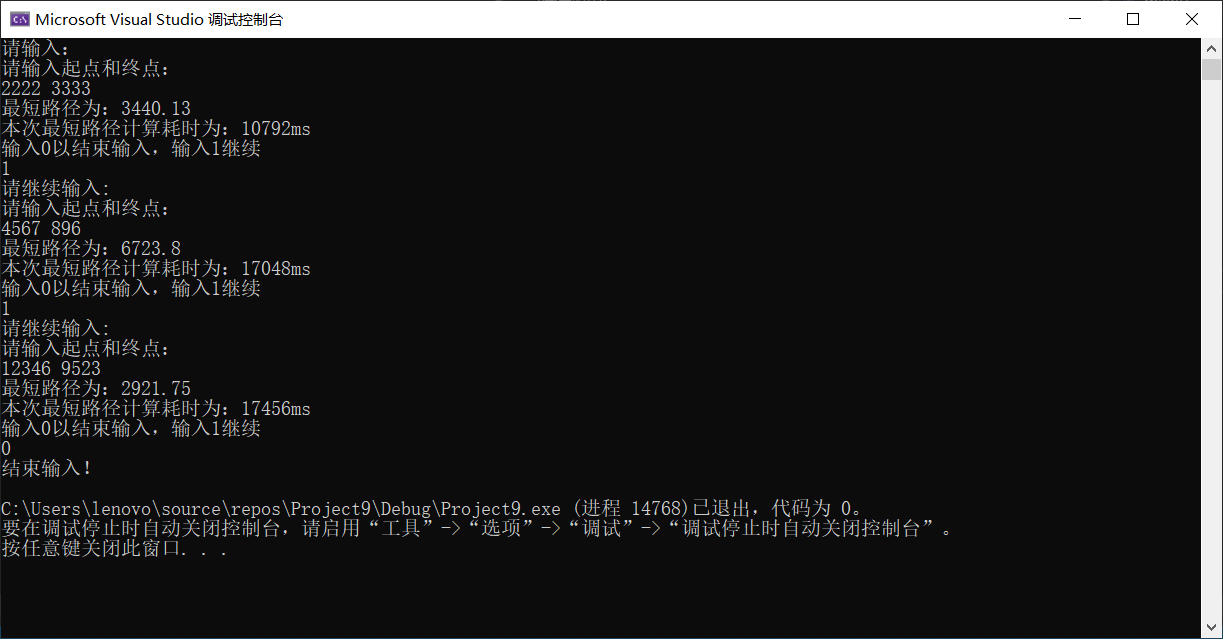
想法 2. 你可以利用问题的欧式几何来进一步降低搜索时间，这在算法书的第 4.4 节描述过。对于一般图， Dijkstra 通过将 d[w]更新为 d[v] + 从 v 到 w 的距离来松弛边 v-w。 对于地图，则将 d[w]更新为 d[v] + 从 v 到 w 的距离 + 从 w 到 d 的欧式距离 − 从 v 到 d 的欧式距离。 这种方法称之为 A\*算法。这种启发式方法会有性能上的变化，但不会影响正确性。

根据此思路，我们在每次找到一个小于∞的距离时，我们需要将优先队列中的距离更新为d[v] + 从 v 到 w 的距离 + 从 w 到 d 的欧式距离 − 从 v 到 d 的欧式距离。在这里需要注意的是只需要修改优先队列中的d[w]，而不需要修改实际上的d[w]，即真正的d[w]依然更新为d[v] + 从 v 到 w 的距离。这个地方如果不注意会导致比较微妙的bug，不易察觉。

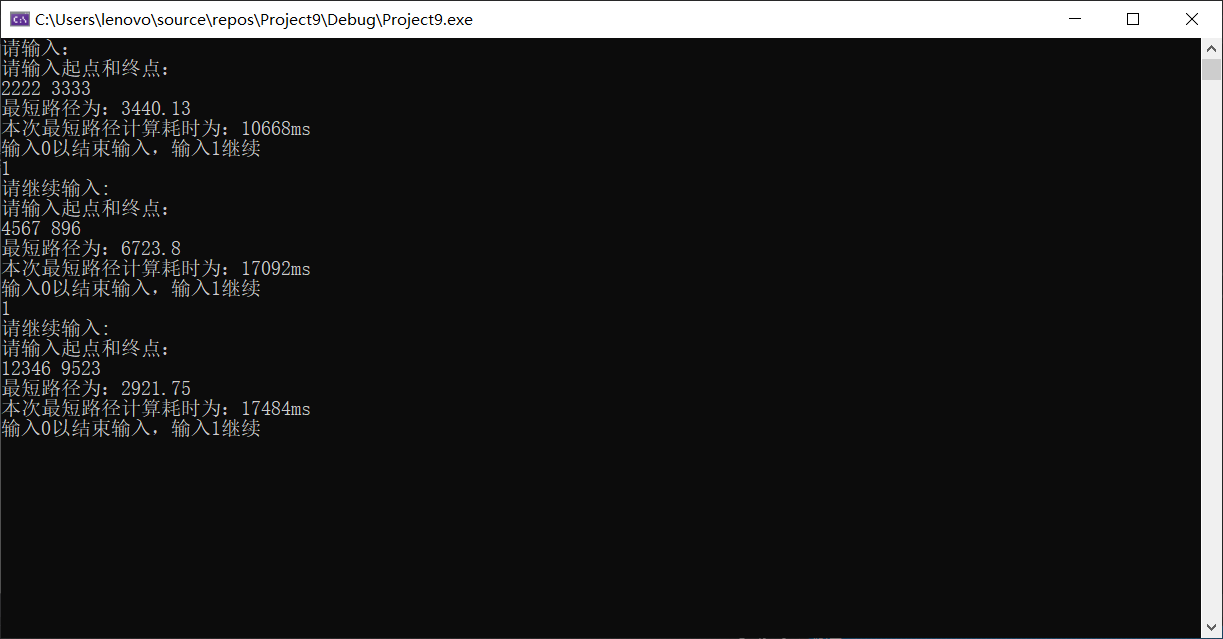
由于我是使用C++来进行编程，某些函数的具体实现与java的方法略有不同，比如优先队列的排序函数需要自己来设计。同时，我并没有采用书中例题所示的图的样例，而是自己设计了一个Point类，边的关系我通过点与点之间的邻接表来表示，所以后来我放弃了使用Edge类来表示边。另外，由于C++的优先队列不具备查找以及修改功能，我自行设计了一个ChangeDist函数，用来查找距离是否已经被加入到优先队列中了，并对其做一定的修改。

1. 实验结果

下面是改进后的结果：



下面是未改进的结果：



1. 实验结果分析

可以看到，改进的算法同样正确的计算出了最短路径的大小，但是最终的效率似乎并没有改变太多，我推测这可能与我所编写的程序有关，我的思路与书上的样例思路并不太一致，这很可能导致了我的算法具有很低的效率，当然这是需要后续优化的一点，不过算法目前已经可以做到结果的正确，这一点已经得到了验证。