// mgraph.h文件

#include <vector>

#include <iostream>

#ifndef \_POINT\_H

#define \_POINT\_H

class Point

{

public:

    int x; int y;

    Point(int x, int y)

    {

        this->x = x; this->y = y;

    }

    Point()

    {

        this->x = this->y = 0;

    }

    friend std::ostream &operator<<(std::ostream& out, Point& onepoint)

    {

        out << "[" << onepoint.x << ", " << onepoint.y << "]";

        return out;

    }

    Point& operator=(const Point& p)

    {

        x = p.x; y = p.y;

        return \*this;

    }

    bool operator==(const Point& p)

    {

        if (x == p.x && y == p.y)

            return true;

        return false;

    }

    Point(const Point &p)

    {

        this->x = p.x; this->y = p.y;

    }

};

#endif

#ifndef \_EDGE\_H

#define \_EDGE\_H

struct Edge

{

    Point adjvex;

    int lowcost;

};

#endif

#ifndef \_MGRAPH\_H

#define \_MGRAPH\_H

class MGraph

{

private:

    int vexnum;

    std::vector<Point> vexs;

    std::vector<std::vector<int>> edges;

    int MiniNum(Edge \*miniegdes);

public:

    MGraph(std::vector<Point> &v);

    Point GetVexValue(int v);

    int GetEdgeValue(int v, int i);

    ~MGraph();

    int Prim(int v, std::vector<std::pair<Point, Point>>&);

    int Prim(int v);

};

#endif

// mgraph.cpp文件

#include <vector>

#include <cstdlib>

#include "mgraph.h"

MGraph::MGraph(std::vector<Point>& v)

{

    this->vexnum = v.size();

*// 初始化节点数量*

    this->vexs.resize(this->vexnum);

*// 初始化边矩阵大小*

    this->edges.resize(this->vexnum);

    for (int i = 0; i < this->vexnum; i++)

    {

        this->vexs.at(i) = v[i];

        this->edges.at(i).resize(this->vexnum);

    }

    for (int i = 0; i < this->vexnum; i++)

    {

        for (int j = 0; j < this->vexnum; j++)

        {

            this->edges.at(i).at(j) = abs(v[i].x - v[j].x) + abs(v[i].y - v[j].y);

        }

    }

}

MGraph::~MGraph()

{

    for (int i = 0; i < this->vexnum; i++)

    {

        this->edges[i].clear();

    }

    this->edges.clear();

    this->vexs.clear();

}

int MGraph::MiniNum(Edge \*miniedges)

{

    int k, i;

    for (i = 0; i < vexnum; ++i)

        if (miniedges[i].lowcost != 0 && miniedges[i].lowcost != 0x80000000)

        {

            k = i;

            break;

        }

    for (i = 0; i < vexnum; ++i)

    {

        if (miniedges[i].lowcost == 0 || miniedges[i].lowcost == 0x80000000)

            continue;

        if (miniedges[i].lowcost < miniedges[k].lowcost)

            k = i;

    }

    return k;

}

Point MGraph::GetVexValue(int v)

{

    return this->vexs.at(v);

}

int MGraph::GetEdgeValue(int v, int i)

{

    return this->edges.at(v).at(i);

}

int MGraph::Prim(int v, std::vector<std::pair<Point, Point>>& vec)

{

*// std::vector<std::pair<char, char>> vec;*

    int length = 0;

    Edge\* miniedges = new Edge[this->vexnum];

    for (int i = 0; i < this->vexnum; i++)

    {

        miniedges[i].adjvex = GetVexValue(v);

        miniedges[i].lowcost = GetEdgeValue(v, i);

    }

    miniedges[v].lowcost = 0;

    for (int i = 1; i < this->vexnum; i++)

    {

        int k = MiniNum(miniedges);

        Point p1 = miniedges[k].adjvex;

        Point p2 = GetVexValue(k);

        vec.push\_back(std::pair<Point, Point>(p1, p2));

        length += abs(p1.x - p2.x) + abs(p1.y - p2.y);

        miniedges[k].lowcost = 0;

        for (int j = 0; j < this->vexnum; j++)

        {

            if (GetEdgeValue(k, j) < miniedges[j].lowcost)

            {

                miniedges[j].adjvex = GetVexValue(k);

                miniedges[j].lowcost = GetEdgeValue(k, j);

            }

        }

    }

    delete[] miniedges;

    return length;

}

int MGraph::Prim(int v)

{

*// std::vector<std::pair<char, char>> vec;*

    int length = 0;

    Edge\* miniedges = new Edge[this->vexnum];

    for (int i = 0; i < this->vexnum; i++)

    {

        miniedges[i].adjvex = GetVexValue(v);

        miniedges[i].lowcost = GetEdgeValue(v, i);

    }

    miniedges[v].lowcost = 0;

    for (int i = 1; i < this->vexnum; i++)

    {

        int k = MiniNum(miniedges);

        Point p1 = miniedges[k].adjvex;

        Point p2 = GetVexValue(k);

        length += abs(p1.x - p2.x) + abs(p1.y - p2.y);

        miniedges[k].lowcost = 0;

        for (int j = 0; j < this->vexnum; j++)

        {

            if (GetEdgeValue(k, j) < miniedges[j].lowcost)

            {

                miniedges[j].adjvex = GetVexValue(k);

                miniedges[j].lowcost = GetEdgeValue(k, j);

            }

        }

    }

    delete[] miniedges;

    return length;

}

// test.cpp文件

#include <iostream>

#include <fstream>

#include <vector>

#include "mgraph.cpp"

#include "mgraph.h"

*// 原本的 9 个点*

void get\_bases(std::vector<Point>& bases)

{

    bases.clear();

    bases.push\_back(Point(36, 35));

    bases.push\_back(Point(50, 31));

    bases.push\_back(Point(68, 16));

    bases.push\_back(Point(65, 5));

    bases.push\_back(Point(86, -12));

    bases.push\_back(Point(52, -9));

    bases.push\_back(Point(57, -34));

    bases.push\_back(Point(35, -23));

    bases.push\_back(Point(25, 6));

}

*// 对 9 个基站建立最小生成树，结果为 229*

void test()

{

std::vector<Point> arr;

get\_bases(arr);

    MGraph m = MGraph(arr);

    std::vector<std::pair<Point, Point>> vec;

    int length = m.Prim(0, vec);

    std::cout << length << std::endl;

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

    {

        std::cout << vec.at(i).first << "-->"

                 << vec.at(i).second << std::endl;

    }

}

*// 在一个矩形区域内逐个取点，将这 10 个点逐一建立最小生成树，路径长度存入文件*

void test2()

{

std::vector<Point> arr;

get\_bases(arr);

    std::ofstream fout;

    int min = 0x7fffffff;

    fout.open("result.csv");

    for (int i = 20; i < 90; i++)

    {

        for (int j = -40; j < 40; j++)

        {

            arr.push\_back(Point(i, j));

            if (!pointInPoints(arr, 9, arr[9]))

            {

                MGraph m = MGraph(arr);

                std::vector<std::pair<Point, Point>> vec;

                int length = m.Prim(0, vec);

                if (min > length) min = length;

                fout << i << "," << j << ","

                    << length << "," << std::endl;

                vec.clear();

            }

            else

            {

                fout << i << "," << j << ","

                    << 260 << "," << std::endl;

            }

            arr.pop\_back();

        }

    }

    fout.close();

}

int main()

{

    test2();

    return 0;

}

# showplt.py文件

import numpy as np

import matplotlib.pyplot as plt

import csv

x\_min, x\_max = 20, 90

y\_min, y\_max = -40, 40

# 将每一个最小生成树的总路径长度整理成一个矩阵

def read\_arr():

    arr = np.zeros((0, 1))

    with open("result.csv", "r") as fin:

        csv\_reader = csv.reader(fin)

        for row in csv\_reader:

            brr = np.array([int(row[2])])

            arr = np.vstack((arr, brr))

    return arr.reshape(x\_max - x\_min, y\_max - y\_min).T

# 根据上面的矩阵画出热成像图

def show\_plt(arr: np.ndarray):

    plt.figure('Imshow', facecolor='lightgray')

    plt.imshow(arr, cmap="jet", origin="lower")

    plt.colorbar()

    plt.show()

if \_\_name\_\_ == "\_\_main\_\_":

    show\_plt(read\_arr())

// t1.cpp文件

*// 用于测试 posible 中的点进行组合，加入到基站中*

*// 得到路径长度的最小值*

#include <iostream>

#include "mgraph.cpp"

#include "mgraph.h"

#include <vector>

void get\_combination(

    std::vector<Point>& vec,

    std::vector<Point>& bases,

    int begin, int len,

    std::vector<Point>& stack,

    int& min

);

*// 可选的点，可以再加亿些点*

void get\_posible\_points(std::vector<Point>& vec)

{

    vec.clear();

    vec.push\_back(Point(36, 31));

    vec.push\_back(Point(36, 16));

    vec.push\_back(Point(50, 6));

    vec.push\_back(Point(50, 16));

    vec.push\_back(Point(52, 16));

    vec.push\_back(Point(52, 15));

    vec.push\_back(Point(52, 5));

    vec.push\_back(Point(36, 6));

    vec.push\_back(Point(36, 5));

    vec.push\_back(Point(35, 5));

    vec.push\_back(Point(35, 6));

    vec.push\_back(Point(35, -9));

    vec.push\_back(Point(35, -12));

    vec.push\_back(Point(52, -12));

    vec.push\_back(Point(52, -23));

    vec.push\_back(Point(57, -23));

    vec.push\_back(Point(57, -12));

    vec.push\_back(Point(57, -9));

    vec.push\_back(Point(57, 5));

    vec.push\_back(Point(52, -34));

    vec.push\_back(Point(65, -9));

    vec.push\_back(Point(65, 16));

    vec.push\_back(Point(68, 5));

}

*// 原本的 9 个点*

void get\_bases(std::vector<Point>& bases)

{

    bases.clear();

    bases.push\_back(Point(36, 35));

    bases.push\_back(Point(50, 31));

    bases.push\_back(Point(68, 16));

    bases.push\_back(Point(65, 5));

    bases.push\_back(Point(86, -12));

    bases.push\_back(Point(52, -9));

    bases.push\_back(Point(57, -34));

    bases.push\_back(Point(35, -23));

    bases.push\_back(Point(25, 6));

}

void try\_points(std::vector<Point>& vec, std::vector<Point>& bases, int num)

{

    int min = 9999;

    std::vector<Point> stack;

    get\_combination(vec, bases, 0, num, stack, min);

    std::cout << min << std::endl;

}

void get\_combination(

    std::vector<Point>& vec,

    std::vector<Point>& bases,

    int begin, int len,

    std::vector<Point>& stack,

    int& min

) {

    if (len == 0)

    {

        for (auto it = stack.begin(); it != stack.end(); it++)

            bases.push\_back(\*it);

        MGraph m = MGraph(bases);

        int length = m.Prim(0);

        if (min > length) min = length;

*// 找目前最小的路径*

        if (length == 190)

        {

            std::cout << length << std::endl;

            std::vector<std::pair<Point, Point>> vv;

            m.Prim(0, vv);

            for (int k = 0; k < vv.size(); k++)

            {

                std::cout << vv[k].first << "-->"

                 << vv[k].second << std::endl;

            }

        }

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

            bases.pop\_back();

        return;

    }

    if (begin >= vec.size())

    {

        return;

    }

    stack.push\_back(vec.at(begin));

    get\_combination(vec, bases, begin + 1, len - 1, stack, min);

    stack.pop\_back();

    get\_combination(vec, bases, begin + 1, len, stack, min);

}

int main()

{

    std::vector<Point> vec;

    std::vector<Point> bases;

    get\_posible\_points(vec);

    get\_bases(bases);

*// 在备选通讯站中选出 5 个*

    try\_points(vec, bases, 5);

    return 0;

}