**四、模型评估和改进**

**4.1模型优点**

（1）对于问题一本文选择了以岩石轮廓的质心作为图论节点，在一定程度上简化了计算，使得问题聚焦于路线的规划。

（2）对于问题一本文通过枚举每次攀爬者最大上升高度，找到了合适的常数，并以此构建网络图，保证了图的连通性。

（3）对于问题一本文通过分析节点数和边数，选择了以邻接表建图，使得算法运行更高效。

（4）对于问题二本文通过使用拆点算法，将点权转变为了边权，使得可以使用更高效的图论算法，加快了程序的运行效率。

（5）对于问题三本文使用了聚类分析，并通过肘部原则，选取了聚类数为2，以此更好地从整体方面评价路径。

**4.2模型缺点**

（1）本文忽略了岩石轮廓的大小和岩石之间的角度方向考虑，缺少了力学方面的计算分析。

（2）Dijkstra算法会有其对应的局限性，如不能处理负权边，此时可以使用SPFA或Floyd算法进行优化。

（3）由于本题数据过少，聚类分析的结果不能很好的统计分析，不能很好地应用在大批量数据。

**参考文献**

[1] 王伯宇. 攀岩運動型態之分析與比較[J]. 大專體育, 2003 (66): 62-68.

[2] 朱松梅. 攀岩运动力量训练研究[J]. 河南师范大学学报: 自然科学版, 2010, 38(2): 169-171.

[3] 张福浩, 刘纪平, 李青元. 基于 Dijkstra 算法的一种最短路径优化算法[J]. 遥感信息, 2004, 2(4).

[4] 乐阳, 龚健雅. Dijkstra 最短路径算法的一种高效率实现[D]. , 1999.

[5] 侯宾, 张文志, 戴源成, 等. 基于 OpenCV 的目标物体颜色及轮廓的识别方法[J]. 现代电子技术, 2014, 37(24): 76-79.

[6] 王千, 王成, 冯振元, 等. K-means 聚类算法研究综述[J]. 电子设计工程, 2012, 20(7): 21-24.

**附录**

**问题一**

**OpenCV轮廓识别代码：**

import cv2

import numpy as np

import heapq

def euclidean\_distance(point1, point2):

    return np.sqrt((point1[0] - point2[0])\*\*2 + (point1[1] - point2[1])\*\*2)

def dijkstra(graph, start):

    distances = {vertex: float('infinity') for vertex in graph}

    distances[start] = 0

    priority\_queue = [(0, start)]

    while priority\_queue:

        current\_distance, current\_vertex = heapq.heappop(priority\_queue)

        if current\_distance > distances[current\_vertex]:

            continue

        for neighbor, weight in graph[current\_vertex].items():

            distance = current\_distance + weight

            if distance < distances[neighbor]:

                distances[neighbor] = distance

                heapq.heappush(priority\_queue, (distance, neighbor))

    return distances

title = "using 70"

cv2.putText(output, title, (30, 50), cv2.FONT\_HERSHEY\_SIMPLEX, 5, (0, 0, 255), 5)

image = cv2.imread("test.jpg")

gray = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)

edges = cv2.Canny(gray, threshold1=20, threshold2=60)

adaptive\_thresh = cv2.adaptiveThreshold(gray, 255, cv2.ADAPTIVE\_THRESH\_GAUSSIAN\_C,

                                       cv2.THRESH\_BINARY, 11, 2)

kernel = np.ones((5, 5), np.uint8)

dilated = cv2.dilate(adaptive\_thresh, kernel, iterations=1)

contours, \_ = cv2.findContours(edges, cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)

output = np.copy(image)

center\_points = []

index = 1

for idx, contour in enumerate(contours):

    area = cv2.contourArea(contour)

    if area > 5:

        M = cv2.moments(contour)

        # if M["m00"] != 0:

        cX = int(M["m10"] / M["m00"])

        cY = int(M["m01"] / M["m00"])

        cv2.circle(output, (cX, cY), 5, (0, 0, 255), -1)

        cv2.putText(output, f"({cX},{cY})", (cX - 50, cY - 10),

                    cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (255, 255, 255), 2)

        cv2.putText(output, f"({cX},{cY})", (cX + 10, cY + 10),

                    cv2.FONT\_HERSHEY\_SIMPLEX, 0.3, (0, 0, 0), 1)

        cv2.putText(output, f"{index}", (cX - 10, cY + 30),

                    cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (0, 0, 255), 2)

        center\_points.append((cX, cY))

        index = index + 1

graph = {}

for i in range(len(center\_points)):

    graph[i] = {}

    for j in range(i + 1, len(center\_points)):

        distance = euclidean\_distance(center\_points[i], center\_points[j])

        if distance <= 100:

            graph[i][j] = distance

            # cv2.line(output, center\_points[i], center\_points[j], (255, 0, 0), 1)  # 修改颜色为蓝色

shortest\_paths = dijkstra(graph, start=1)

target\_points = [181, 174, 182]

for target in target\_points:

    if target in shortest\_paths:

        print(f"Shortest distance from point 1 to point {target}: {shortest\_paths[target]:.2f}")

    else:

        print(f"No path from point 1 to point {target}")

cv2.imwrite("label.jpg", output)

cv2.imshow("Rock Contours, Centers, and Blue Edges", output)

cv2.waitKey(0)

cv2.destroyAllWindows()

**将中心点坐标存入文本文件中：**

# 将center\_points写入文本文件中

file\_name = "center\_points.txt"

# 打开文件以写入模式

with open(file\_name, "w") as file:

    for point in center\_points:

        # 将每个元组的数据转换为字符串，然后写入文件

        file.write(f"{point[0]} {point[1]}\n")

print("列表已成功写入文本文件。")

**使用Dijkstra算法求解最短路：**

#include <iostream>

#include <algorithm>

#include <vector>

#include <fstream>

#include <cmath>

#include <queue>

#include <cstring>

#include <map>

using namespace std;

struct Points

{

    int id;

    int x, y;

};

struct Edges

{

    int id1, id2;

    double distance;

};

const int N = 150;

double g[N][N];

double dist[N];

bool st[N];

int s = 1;

void dijkstra()

{

    int prev[N]; // 用于记录每个节点的前驱节点

    for (int i = 1; i <= N; i ++)

    {

        dist[i] = 10000000000.0;

        prev[i] = -1;

    }

    memset(st, 0, sizeof st);

    dist[s] = 0;

    int n = 115;

    for (int i = 1; i <= n; i ++ )

    {

        int t = -1;

        for (int j = 1; j <= n; j ++ )

            if (!st[j] && (t == -1 || dist[t] > dist[j]))

                t = j;

        st[t] = true;

        for (int j = 1; j <= n; j ++ )

        {

            if (dist[j] > dist[t] + g[t][j])

            {

                dist[j] = dist[t] + g[t][j];

                prev[j] = t; // 记录节点j的前驱节点是t

            }

        }

    }

    // 打印从节点1到节点111的路径

    int current = 114;

    vector<int> path;

    while (current != -1)

    {

        path.push\_back(current);

        current = prev[current];

    }

    reverse(path.begin(), path.end());

    cout << "从1到113的路径经过的节点编号: ";

    for (int node : path)

    {

        cout << node << " ";

    }

    cout << endl;

    double maxEdgeDistance = 0;

    int maxEdgeId1 = -1, maxEdgeId2 = -1;

    for (int i = 1; i < path.size(); i++)

    {

        int u = path[i - 1];

        int v = path[i];

        double edgeDistance = g[u][v];

        if (edgeDistance > maxEdgeDistance)

        {

            maxEdgeDistance = edgeDistance;

            maxEdgeId1 = u;

            maxEdgeId2 = v;

        }

    }

    cout << "最大距离的边信息：" << endl;

    cout << "起点：" << maxEdgeId1 << " 终点：" << maxEdgeId2 << " 距离：" << maxEdgeDistance << endl;

}

int main()

{

    Points center\_points[N];

    ifstream file("center\_points.txt");

    int point\_id = 1;

    while (!file.eof())

    {

        int point\_x, point\_y;

        file >> point\_x >> point\_y;

        center\_points[point\_id] = {point\_id, point\_x, point\_y};

        point\_id ++;

    }

    int n = point\_id - 1;

    Edges edges[N \* N];

    int edge\_id = 0;

    for (int i = 1; i <= n; i ++)

    {

        for (int j = i + 1; j <= n; j ++)

        {

            double distance = sqrt(pow(center\_points[i].x - center\_points[j].x, 2) + pow(center\_points[i].y - center\_points[j].y, 2));

            if (distance <= 100)

            {

                edges[++edge\_id] = {i, j, distance};

            }

        }

    }

    for (int i = 1; i <= N; i ++)

    {

        for (int j = 1; j <= N; j ++)

        {

            g[i][j] = 10000000000.0;

        }

    }

    for (int i = 1; i <= edge\_id; i ++)

    {

        g[edges[i].id1][edges[i].id2] = edges[i].distance;

        g[edges[i].id2][edges[i].id1] = edges[i].distance;

    }

    dijkstra();

    return 0;

}

**问题二**

**拆点并使用最短路：**

#include <iostream>

#include <algorithm>

#include <vector>

#include <fstream>

#include <cmath>

#include <queue>

#include <cstring>

#include <map>

using namespace std;

const int N = 300, M = N << 1;

int h[N], e[M], ne[M], w[M], idx;

int dist[N];

bool st[N];

struct Points

{

    int id;

    int x, y;

    int color\_id;

};

struct Edges

{

    int id1, id2;

    double distance;

};

typedef pair<int, int> PII;

int t = 114;

int s = 1, s\_n = s + t; // 起点的出点和起点的入点

void add(int a, int b, int c)

{

    e[idx] = b, w[idx] = c, ne[idx] = h[a], h[a] = idx ++ ;

}

void dijkstra()

{

    memset(dist, 0x3f, sizeof dist);

    memset(st, 0, sizeof st);

    priority\_queue<PII, vector<PII>, greater<>> heap;

    heap.push({0, s});

    dist[s] = 0;

    int prev[N];  // 用于记录路径

    while (!heap.empty())

    {

        PII t = heap.top();

        heap.pop();

        if (st[t.second])  // 跳过已处理的点

            continue;

        st[t.second] = true;

        for (int i = h[t.second]; ~i; i = ne[i])

        {

            int j = e[i];

            if (dist[j] > dist[t.second] + w[i])

            {

                dist[j] = dist[t.second] + w[i];

                prev[j] = t.second;  // 记录路径

                heap.push({dist[j], j});

            }

        }

    }

    // 打印经过的点的编号

    int current = t + 114;  // 终点编号

    vector<int> path;

    while (current != s)

    {

        path.push\_back(current);

        current = prev[current];

    }

    path.push\_back(s);

    cout << "114号经过的点的编号：";

    for (int i = path.size() - 1; i >= 0; i-=2)

    {

        cout << path[i] << " ";

    }

    cout << endl;

}

int main()

{

    Points center\_points[N];

    ifstream file("center\_points\_label.txt");

    int point\_id = 0;

    while (!file.eof())

    {

        int point\_x, point\_y, color\_id;

        file >> point\_x >> point\_y >> color\_id;

        if (color\_id == 1) color\_id = 3;

        else if (color\_id == 2) color\_id = 4;

        else if (color\_id == 3) color\_id = 2;

        else if (color\_id == 4) color\_id = 1;

        center\_points[++ point\_id] = {point\_id, point\_x, point\_y, color\_id};

        // cout << point\_id << " " << point\_x << " " << point\_y << " " << color\_id << endl;

        if (point\_id == 114) break;

    }

    // cout << point\_id << endl;

    int n = point\_id;

    Edges edges[N \* N];

    int edge\_id = 0;

    for (int i = 1; i <= n; i ++)

    {

        for (int j = i + 1; j <= n; j ++)

        {

            double distance = sqrt(pow(center\_points[i].x - center\_points[j].x, 2) + pow(center\_points[i].y - center\_points[j].y, 2));

            if (distance <= 100)

            {

                edges[++edge\_id] = {i, j, distance};

            }

        }

    }

    // 建图

    memset(h, -1, sizeof h);

    for (int i = 1; i <= t; i ++)

    {

        add(i, i + t, center\_points[i].color\_id); // 起点的出点和起点的入点 权值是颜色值

    }

    for (int i = 1; i <= edge\_id; i ++)

    {

        add(edges[i].id1 + t, edges[i].id2, 0); // 该点的出点到另外一个点的入点 权值是0

    }

    dijkstra();

    cout << dist[114 + t] << endl;

    return 0;

}

**文本文件center\_points\_label.txt**

268 1176 3

427 1134 2

175 1134 2

395 1123 3

298 1120 3

69 1103 2

226 1097 3

473 1086 4

98 1074 4

384 1069 3

420 1051 3

286 1052 1

143 1034 1

346 1030 2

214 1000 4

88 996 2

425 978 2

334 964 1

120 969 1

490 941 1

281 946 1

123 914 1

390 909 3

181 910 1

245 891 1

79 872 2

433 862 4

352 870 2

165 848 3

513 832 3

262 822 2

499 805 3

413 792 3

175 801 1

317 792 1

91 766 2

253 741 1

452 737 3

294 724 4

214 713 4

511 703 3

377 700 3

172 694 1

210 663 4

256 660 3

342 659 1

102 648 4

251 627 3

200 623 3

376 613 3

173 598 3

108 594 4

438 592 1

321 582 3

526 566 1

56 555 3

231 553 4

340 531 2

398 532 1

317 522 1

103 522 4

221 512 1

545 498 1

384 479 1

132 477 1

438 471 1

47 467 4

494 448 1

260 450 4

142 435 2

379 418 2

50 410 1

493 407 2

278 393 3

226 377 4

89 369 2

135 365 3

273 363 3

313 360 4

401 367 1

454 353 2

230 345 3

109 321 3

252 314 3

457 304 1

542 295 2

209 293 2

278 281 4

366 265 1

174 243 3

302 230 2

455 227 3

228 210 1

436 208 4

57 202 3

531 189 4

325 192 2

163 193 3

135 189 3

324 159 4

210 151 3

424 148 3

368 120 4

144 116 3

503 111 4

430 93 3

152 87 3

289 98 4

471 80 4

230 67 4

329 53 4

69 48 3

183 41 1

496 35 4