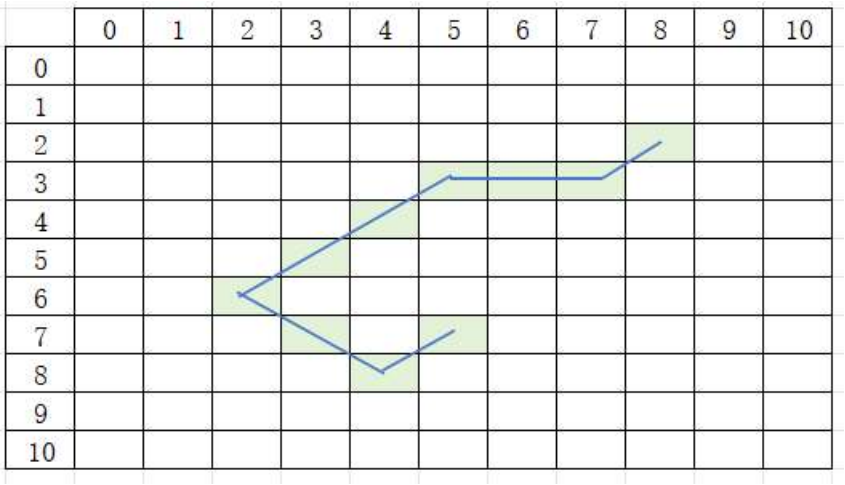
【华为OD机考 统一考试机试C卷】多段线数据压缩（C++ Java JavaScript Python C语言）

华为OD机考:统一考试 C卷 + D卷 + B卷 +A卷

目前在考C卷，经过两个月的收集整理，C卷真题已基本整理完毕
抽到原题的概率为2/3到3/3，也就是最少抽到两道原题。请注意：大家刷完C卷真题，最好要把B卷的真题刷一下，因为C卷的部分真题来自B卷。
另外订阅专栏还可以联系笔者开通在线 OJ 进行刷题，提高刷题效率。
真题目录：华为OD机考机试 真题目录（C卷 + D卷 + B卷 + A卷） + 考点说明
专栏：2023华为OD机试(B卷+C卷+D卷)（C++JavaJSPy）
华为OD面试真题精选：华为OD面试真题精选
在线OJ：点击立即刷题，模拟真实机考环境

题目描述

下图中，每个方块代表一个像素，每个像素用其行号和列号表示。为简化处理，多段线的走向只能是水平、竖直、斜向45度。



上图中的多段线可以用下面的坐标串表示：(2, 8), (3, 7), (3, 6), (3, 5), (4, 4), (5, 3), (6, 2), (7, 3), (8, 4), (7, 5)。

但可以发现，这种表示不是最简的，其实只需要存储6个蓝色的关键点即可，它们是线段的起点、拐点、终点，而剩下4个点是冗余的。

> 即可以简化为：（2,8）、（3,7）、（3,5）、（6,2）、（8,4）、（7,5）

现在，请根据输入的包含有冗余数据的多段线坐标列表，输出其最简化的结果。

输入描述

2 8 3 7 3 6 3 5 4 4 5 3 6 2 7 3 8 4 7 5

1、所有数字以空格分隔，每两个数字一组，第一个数字是行号，第二个数字是列号；

2、行号和列号范围为[0,64)，用例输入保证不会越界，考生不必检查；

3、输入数据至少包含两个坐标点。

输出描述

2 8 3 7 3 5 6 2 8 4 7 5

压缩后的最简化坐标列表，和输入数据的格式相同。

备注: 输出的坐标相对顺序不能变化。

用例

输入

```
2 8 3 7 3 6 3 5 4 4 5 3 6 2 7 3 8 4 7 5
```

输出

```
2 8 3 7 3 5 6 2 8 4 7 5
```

说明

> 如上图所示，6个蓝色像素的坐标依次是（2,8）、（3,7）、（3,5）、（6,2）、（8,4）、（7,5）。

解题思路

遍历输入的坐标点列表，对于每个当前考察的点 `curr`，检查它是否是拐点。为此，比较它与前一个点 `prev` 以及后一个点 `next` 形成的两个向量。如果这两个向量不共线（即它们的叉积不为零），则当前点是一个拐点。

向量和向量的叉积（cross product）。

1. 向量：在二维平面中，向量可以用来表示两点之间的方向和距离。在这段代码中，向量是由两个点确定的，例如，向量 `(dx1, dy1)` 是由点 `prev` 指向点 `curr` 的向量，而向量 `(dx2, dy2)` 是由点 `curr` 指向点 `next` 的向量。
2. 向量的叉积：在二维空间中，两个向量 `a(x1, y1)` 和 `b(x2, y2)` 的叉积定义为 `a.x * b.y - a.y * b.x`，即 `x1 * y2 - y1 * x2`。叉积的结果是一个标量，它的绝对值等于以这两个向量为边的平行四边形的面积。叉积的符号表示这两个向量的相对方向，如果叉积为正，则 `b` 向量在 `a` 向量的逆时针方向；如果叉积为负，则 `b` 向量在 `a` 向量的顺时针方向；如果叉积为零，则两个向量共线。

下面的代码中，叉积用于判断三个连续的点 `prev`、`curr`、`next` 是否构成一个拐点。如果 `prev` 到 `curr` 的向量 `(dx1, dy1)` 与 `curr` 到 `next` 的向量 `(dx2, dy2)` 的叉积不为零，即 `dx1 * dy2 != dy1 * dx2`，则说明这两个向量不共线，即 `curr` 点是一个拐点。如果叉积为零，则表示这三个点共线，`curr` 点不是拐点。

C++

```cpp
#include <iostream>
#include <vector>

using namespace std;
// 判断当前点是否为拐点
bool isTurningPoint(const pair<int, int>& prev, const pair<int, int>& curr, const pair<int, int>& next) {
    // 计算向量
    int dx1 = curr.first - prev.first;
    int dy1 = curr.second - prev.second;
    int dx2 = next.first - curr.first;
    int dy2 = next.second - curr.second;

    // 如果两个向量不在同一直线上，则为拐点
    return dx1 * dy2 != dy1 * dx2;
}

// 简化路径
vector<pair<int, int>> simplifyPath(const vector<pair<int, int>>& points) {

```

```
    vector<pair<int, int>> result;
    // 至少需要两个点才能形成路径
    if (points.size() < 2) return points;

    // 添加起点
    result.push_back(points[0]);

    // 遍历所有点，找出拐点
    for (size_t i = 1; i < points.size() - 1; ++i) {
        if (isTurningPoint(points[i - 1], points[i], points[i + 1])) {
            result.push_back(points[i]);
        }
    }

    // 添加终点
    result.push_back(points.back());

    return result;
}

int main() {
    vector<pair<int, int>> points;
    int x, y;

    // 读取输入的坐标点
    while (cin >> x >> y) {
        points.emplace_back(x, y);
    }

    // 简化路径
    vector<pair<int, int>> simplifiedPoints = simplifyPath(points);

    // 输出简化后的路径
    for (size_t i = 0; i < simplifiedPoints.size(); ++i) {
        cout << simplifiedPoints[i].first << " " << simplifiedPoints[i].second;
        if (i < simplifiedPoints.size() - 1) {
            cout << " ";
        }
    }
```

```
    return 0;
}
```

Java

```java
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

public class SimplifyPath {

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        // 读取输入的坐标点
        List<int[]> points = new ArrayList<>();
        while (scanner.hasNextInt()) {
            int x = scanner.nextInt();
            int y = scanner.nextInt();
            points.add(new int[]{x, y});
        }
        scanner.close();

        // 简化路径
        List<int[]> simplifiedPoints = simplifyPath(points);

        // 输出简化后的路径
        for (int i = 0; i < simplifiedPoints.size(); i++) {
            System.out.print(simplifiedPoints.get(i)[0] + " " + simplifiedPoints.get(i)[1]);
            if (i < simplifiedPoints.size() - 1) {
                System.out.print(" ");
            }
        }
    }

    private static List<int[]> simplifyPath(List<int[]> points) {
        List<int[]> result = new ArrayList<>();
        // 至少需要两个点才能形成路径
        if (points.size() < 2) return points;

        // 添加起点
```

```
        result.add(points.get(0));

        // 遍历所有点，找出拐点
        for (int i = 1; i < points.size() - 1; i++) {
            int[] prev = points.get(i - 1);
            int[] curr = points.get(i);
            int[] next = points.get(i + 1);

            // 如果当前点是拐点，则加入结果列表
            if (isTurningPoint(prev, curr, next)) {
                result.add(curr);
            }
        }

        // 添加终点
        result.add(points.get(points.size() - 1));

        return result;
    }

    // 判断当前点是否为拐点
    private static boolean isTurningPoint(int[] prev, int[] curr, int[] next) {
        // 计算向量
        int dx1 = curr[0] - prev[0];
        int dy1 = curr[1] - prev[1];
        int dx2 = next[0] - curr[0];
        int dy2 = next[1] - curr[1];

        // 如果两个向量不在同一直线上，则为拐点
        return dx1 * dy2 != dy1 * dx2;
    }
}
```

javaScript

```
// Node.js

// 使用 readline 模块来处理输入
const readline = require('readline');
const rl = readline.createInterface({
```

```
  input: process.stdin,
  output: process.stdout
});

// 判断当前点是否为拐点
function isTurningPoint(prev, curr, next) {
  // 计算向量
  const dx1 = curr[0] - prev[0];
  const dy1 = curr[1] - prev[1];
  const dx2 = next[0] - curr[0];
  const dy2 = next[1] - curr[1];

  // 如果两个向量不在同一直线上，则为拐点
  return dx1 * dy2 !== dy1 * dx2;
}

// 简化路径
function simplifyPath(points) {
  // 至少需要两个点才能形成路径
  if (points.length < 2) return points;

  // 添加起点
  const result = [points[0]];

  // 遍历所有点，找出拐点
  for (let i = 1; i < points.length - 1; i++) {
    if (isTurningPoint(points[i - 1], points[i], points[i + 1])) {
      result.push(points[i]);
    }
  }

  // 添加终点
  result.push(points[points.length - 1]);

  return result;
}

const points = [];
rl.on('line', (line) => {
  const nums = line.split(' ').map(Number);

```

```
  for (let i = 0; i < nums.length; i += 2) {
    points.push([nums[i], nums[i + 1]]);
  }
  rl.close();
}).on('close', () => {
  // 简化路径
  const simplifiedPoints = simplifyPath(points);

  // 输出简化后的路径
  const output = simplifiedPoints.map(point => point.join(' ')).join(' ');
  console.log(output);

  process.exit(0);
});
```

Python

```


# 判断当前点是否为拐点
def is_turning_point(prev, curr, next):
    # 计算向量
    dx1 = curr[0] - prev[0]
    dy1 = curr[1] - prev[1]
    dx2 = next[0] - curr[0]
    dy2 = next[1] - curr[1]

    # 如果两个向量不在同一直线上，则为拐点
    return dx1 * dy2 != dy1 * dx2

# 简化路径
def simplify_path(points):
    # 至少需要两个点才能形成路径
    if len(points) < 2:
        return points

    # 添加起点
    result = [points[0]]

    # 遍历所有点，找出拐点
```

```python
    for i in range(1, len(points) - 1):
        if is_turning_point(points[i - 1], points[i], points[i + 1]):
            result.append(points[i])

    # 添加终点
    result.append(points[-1])

    return result



# 读取输入的坐标点
points = []
line = [int(i) for i in input().split()]
for i in range(0, len(line), 2):
    points.append((line[i],line[i+1]))


    pass

# 简化路径
simplified_points = simplify_path(points)

# 输出简化后的路径
print(' '.join(f"{x} {y}" for x, y in simplified_points))
```

C语言

```c
#include <stdio.h>
#include <stdlib.h>

// 定义坐标点的结构
typedef struct {
    int x;
    int y;
} Point;

// 判断当前点是否为拐点
int isTurningPoint(Point prev, Point curr, Point next) {
    // 计算向量
```

```
    int dx1 = curr.x - prev.x;
    int dy1 = curr.y - prev.y;
    int dx2 = next.x - curr.x;
    int dy2 = next.y - curr.y;

    // 如果两个向量不在同一直线上，则为拐点
    return dx1 * dy2 != dy1 * dx2;
}

int main() {
    Point points[1000]; // 假设最多有1000个点
    int n = 0; // 点的数量
    int x, y;

    // 读取输入的坐标点
    while (scanf("%d %d", &x, &y) == 2) {
        points[n].x = x;
        points[n].y = y;
        n++;
    }

    // 至少需要两个点才能形成路径
    if (n < 2) return 0;

    // 输出起点
    printf("%d %d", points[0].x, points[0].y);

    // 遍历所有点，找出并输出拐点
    for (int i = 1; i < n - 1; i++) {
        if (isTurningPoint(points[i - 1], points[i], points[i + 1])) {
            printf(" %d %d", points[i].x, points[i].y);
        }
    }

    // 输出终点
    printf(" %d %d\n", points[n - 1].x, points[n - 1].y);

    return 0;
}
```

文章目录