安装栅栏

＋ 关注 TA

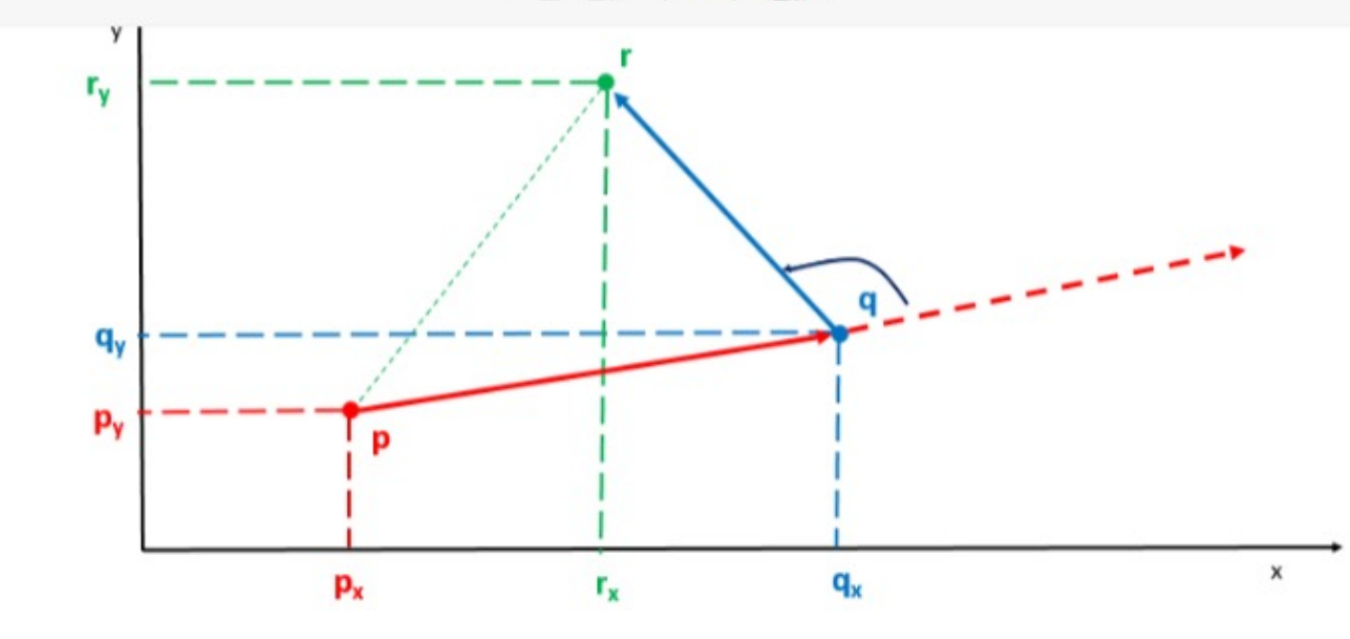
力扣官方题解 L6 发布于 21 小时前 7.7k 官方 几何 数组 数学 C C++ C# Go Java Python

方法一：Jarvis 算法

思路与算法

此题为经典的求凸包的算法，详细的算法原理可以参考「凸包」。常见的凸包算法有多种，在此只描述 Jarvis 算法、Graham 算法、 Andrew 算法。

Jarvis 算法背后的想法非常简单。首先必须要从凸包上的某一点开始，比如从给定点集中最左边的点开始，例如最左的一点 A_1。然后选择 A_2 点使得所有点都在向量 $\vec{A_1A_2}$的左方或者右方，我们每次选择左方，需要比较所有点以 A_1 为原点的极坐标角度。然后以 A_2 为原点，重复这个步骤，依次找到 $A_3, A_4, \ldots, A_k$。
给定原点 p，如何找到点 q，满足其余的点 r 均在向量 $\vec{pq}$ 的左边，我们使用「向量叉积」来进行判别。我们可以知道两个向量 $\vec{pq}, \vec{qr}$ 的叉积大于 0 时，则两个向量之间的夹角小于 180°，两个向量之间构成的旋转方向为逆时针，此时可以知道 r 一定在 $\vec{pq}$ 的左边；叉积等于 0 时，则表示两个向量之间平行，p, q, r 在同一条直线上；叉积小于 0 时，则表示两个向量之间的夹角大于 180°，两个向量之间构成的旋转方向为顺时针，此时可以知道 r 一定在 $\vec{pq}$ 的右边。为了找到点 q，我们使用函数 $\texttt{cross}()$，这个函数有 3 个参数，分别是当前凸包上的点 p，下一个会加到凸包里的点 q，其他点空间内的任何一个点 r，通过计算向量 $\vec{pq}, \vec{qr}$ 的叉积来判断旋转方向，如果剩余所有的点 r 均满足在向量 $\vec{pq}$ 的左边，则此时我们将 q 加入凸包中。

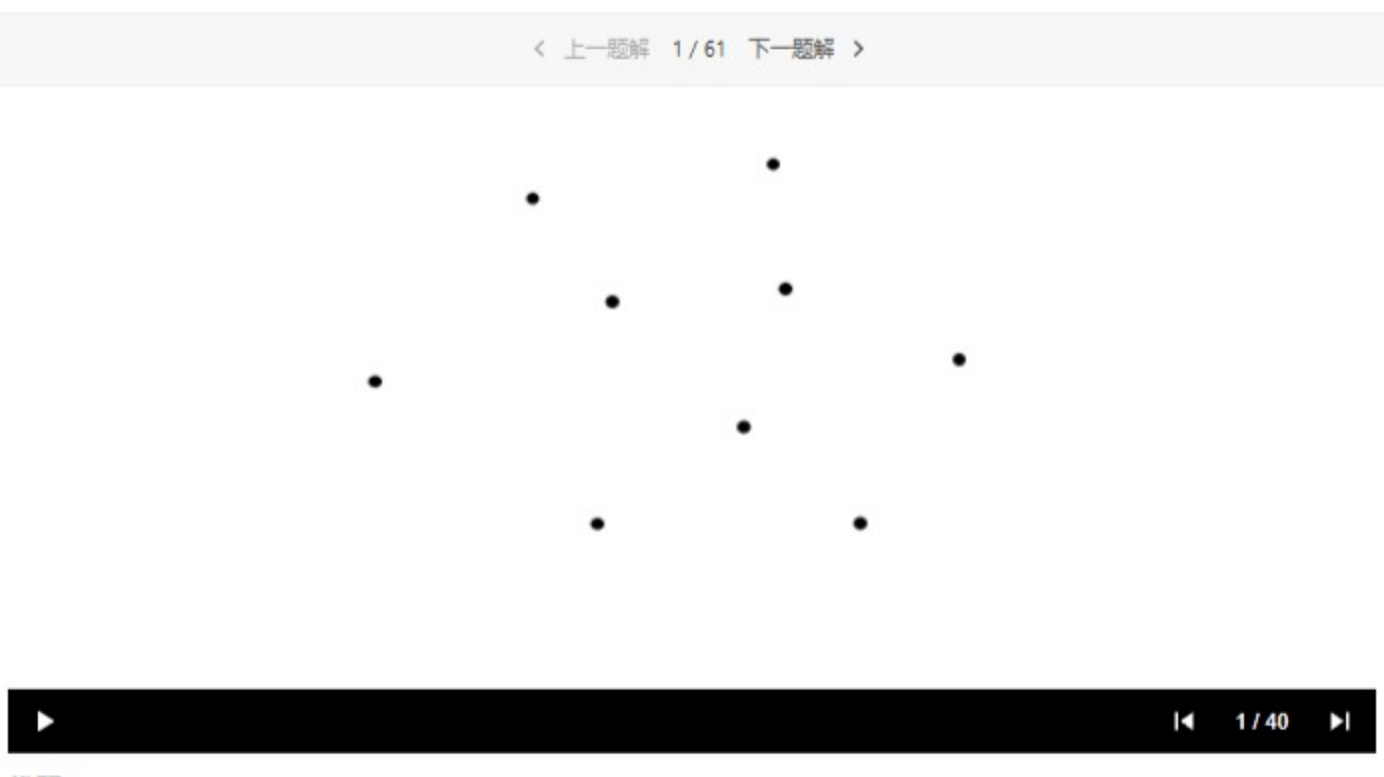


从上图中，我们可以观察到点 p，q 和 r 形成的向量相应地都是逆时针方向，向量 $\vec{pq}$ 和 $\vec{qr}$ 旋转方向为逆时针，函数 $\texttt{cross}(p,q,r)$ 返回值大于 0。

$$\begin{aligned} cross(p,q,r) &= \vec{pq} \times \vec{qr} \\ &= \begin{vmatrix} (q_x-p_x) & (q_y-p_y) \\ (r_x-q_x) & (r_y-p_y) \end{vmatrix} \\ &= (q_x-p_x) \times (r_y-p_y) - (q_y-p_y) \times (r_x-q_x) \end{aligned}$$

我们遍历所有点 r，找到对于点 p 来说逆时针方向最靠外的点 q，把它加入凸包。如果存在 2 个点相对点 p 在同一条线上，我们应当将 q 和 p 同一线段上的边界点都考虑进来，此时需要进行标记，防止重复添加。

通过这样，我们不断将凸包上的点加入，直到回到了开始的点，下面的动图描述了该过程。



代码

Python3 | C++ | Java | C# | C | JavaScript | Golang

```
        n = len(trees)
        if n < 4:
            return trees

        leftMost = 0
        for i, tree in enumerate(trees):
            if tree[0] < trees[leftMost][0]:
                leftMost = i

        ans = []
```

```
                # // 如果 r 在 pq 的右侧，则 q = r
                if cross(trees[p], trees[q], tree) < 0:
                    q = r
            # 是否存在点 i, 使得 p q i 在同一条直线上
            for i, b in enumerate(vis):
                if not b and i != p and i != q and cross(trees[p], trees[q], trees[i]) == 0:
                    ans.append(trees[i])
                    vis[i] = True
            if not vis[q]:
                ans.append(trees[q])
                vis[q] = True
            p = q
            if p == leftMost:
                break
        return ans
```

复杂度分析

- 时间复杂度：$O(n^2)$，其中 n 为数组的长度。每次判定一个点 p，同时需要遍历数组所有点，一共最多需要取出 n 个点，因此时间复杂度为 $O(n^2)$。
- 空间复杂度：$O(n)$。需要对每个点进行标记，需要的空间复杂度为 $O(n)$。

方法二：Graham 算法

思路与算法

这个方法的具体实现为：首先选择一个凸包上的初始点 $bottom$。我们选择 y 坐标最小的点为起始点，我们可以肯定 $bottom$ 一定在凸包上，将给定点集按照相对的以 $bottom$ 为原点的极角大小进行排序。

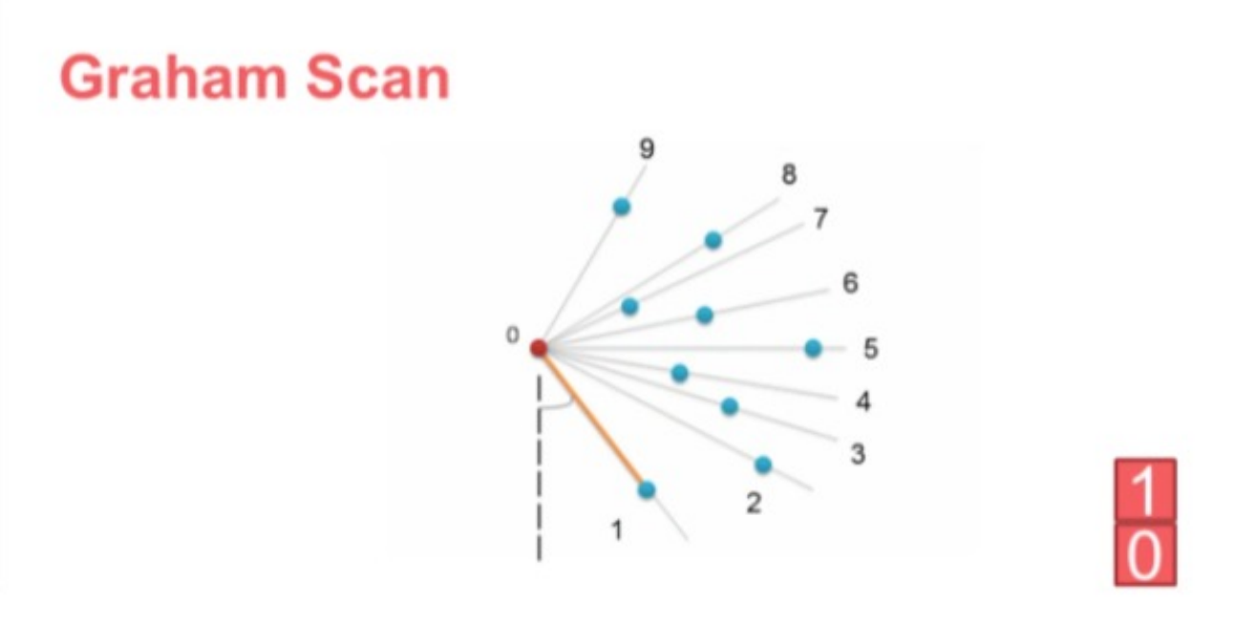
这一排序过程大致给了我们在逆时针顺序选点时候的思路。为了将点排序，我们使用上一方法使用过的函数 cross 。极角顺序更小的点排在数组的前面。如果有两个点相对于点 $bottom$ 的极角大小相同，则按照与点 $bottom$ 的距离排序。

序。

现在我们从有序数组最开始两个点开始考虑。我们将这条线上的点放入栈中。然后我们从第三个点开始遍历有序数组 $trees$。如果当前点与栈顶的点相比前一条线是一个「左拐」或者是同一条线段上，我们都将当前点添加到栈顶，表示这个点暂时被添加到凸壳上。

检查左拐或者右拐使用的还是 cross 函数。对于向量 $\vec{pq}, \vec{qr}$，计算向量的叉积 $\texttt{cross}(p,q,r) = \vec{pq} \times \vec{qr}$，如果叉积小于 0，可以知道向量 $\vec{pq}, \vec{qr}$ 顺时针旋转，则此时向右拐；如果叉积大于 0，可以知道向量 $\vec{pq}, \vec{qr}$ 逆时针旋转，表示是左拐；如果叉积等于 0，则 p, q, r 在同一条直线上。

如果当前点与上一条线之间的关系是右拐的，说明上一个点不应该被包括在凸壳里，因为它在边界的里面（正如动画中点 4），所以我们将它从栈中弹出并考虑倒数第二条线的方向。重复这一过程，弹栈的操作会一直进行，直到我们当前点在凸壳中出现了右拐。这表示这时凸壳中只包括边界上的点而不包括边界以内的点。在所有点被遍历了一遍以后，栈中的点就是构成凸壳的点。



1 / 15

代码

Python3 | C++ | Java | C# | C | JavaScript | Golang

```
class Solution:
    def outerTrees(self, trees: List[List[int]]) -> List[List[int]]:
        def cross(p: List[int], q: List[int], r: List[int]) -> int:
            return (q[0] - p[0]) * (r[1] - q[1]) - (q[1] - p[1]) * (r[0] - q[0])

        def distance(p: List[int], q: List[int]) -> int:
            return (p[0] - q[0]) * (p[0] - q[0]) + (p[1] - q[1]) * (p[1] - q[1])

        n = len(trees)
        if n < 4:
            return trees

        # 找到 y 最小的点 bottom
        bottom = 0
        for i, tree in enumerate(trees):
            if tree[1] < trees[bottom][1]:
                bottom = i
        trees[bottom], trees[0] = trees[0], trees[bottom]

        # 以 bottom 原点，按照极坐标的角度大小进行排序
        def cmp(a: List[int], b: List[int]) -> int:
            diff = cross(trees[0], b, a) - cross(trees[0], a, b)
            return diff if diff else distance(trees[0], a) - distance(trees[0], b)
        trees[1:] = sorted(trees[1:], key=cmp_to_key(cmp))

        # 对于凸包最后且在同一条直线的元素按照距离从小到大进行排序
        r = n - 1
        while r >= 0 and cross(trees[0], trees[n - 1], trees[r]) == 0:
            r -= 1
```

- 时间复杂度：$O(n \log n)$，其中 n 为数组的长度。首先需要对数组进行排序，时间复杂度为 $O(n \log n)$，每次添加栈中添加元素后，判断新加入的元素是否在凸包上，因此每个元素都可能进行入栈与出栈一次，最多需要的时间复杂度为 $O(2n)$，因此总的时间复杂度为 $O(n \log n)$。
- 空间复杂度：$O(n)$，其中 n 为数组的长度。首先该解法需要快速排序，需要的栈空间为 $O(\log n)$，需要栈来保存当前已经判别的元素，栈中最多有 n 个元素，所需要的空间为 $O(n)$，因此总的空间复杂度为 $O(n)$。

方法三：Andrew 算法

思路与算法

Andrew 使用单调链算法，该算法与 Graham 扫描算分类似。它们主要的不同点在于凸壳上点的顺序。与 Graham 扫描算法按照点计较顺序排序不同，我们按照点的 x 坐标排序，如果两个点又相同的 x 坐标，那么就按照它们的 y 坐标排序。显然排序后的最大值与最小值一定在凸包上，而且因为是凸多边形，我们如果从一个点出发逆时针走，轨迹总是「左拐」的，一旦出现右拐，就说明这一段不在凸包上，因此我们可以用一个单调栈来维护上下凸壳。

仔细观察可以发现，最大值与最小值一定位于凸包的最左边与最右边，从左向右看，我们将凸壳考虑成 2 个子边界组成：上凸壳和下凸壳。下凸壳一定是从最小值一直「左拐」直到最大值，上凸壳一定是从最大值「左拐」到最小值，因此我们首先升序枚举求出下凸壳，然后降序求出上凸壳。
我们首先将最初始的两个点添加到凸壳中，然后遍历排好序的 $trees$ 数组。对于每个新的点，我们检查当前点是否在最后两个点的逆时针方向上，轨迹是否是左拐。如果是的话，当前点直接被压入凸壳 $hull$ 中，$cross$ 返回的结果为正数；如果不是的话，$cross$ 返回的结果为负数，我们可以知道栈顶的元素在凸壳里面而不是凸壳边上。我们继续从 $hull$ 中弹出元素直到当前点相对于栈顶的两个点的逆时针方向上。

这个方法中，我们不需要显式地考虑共线的点，因为这些点已经按照 x 坐标排好了序。所以如果有共线的点，它们已经被隐式地按正确顺序考虑了。通过这样，我们会一直遍历到 x 坐标最大的点为止。但是凸壳还没有完全求解出来，目前求解出来的部分只包括凸壳的下半部分。现在我们需要求出凸壳的上半部分

$hull$ 数组返回了我们需要的边界上的点。需要注意的是，由于我们需要检测上凸壳最后加入的点是否合法，此时需要再次插入最左边的点 $textithull[0]$ 进行判别。

下面的动图展示了这一过程。

代码

Python3 | C++ | Java | C# | C | JavaScript | Golang

```
class Solution:
```

```
        if n < 4:
            return trees

        # 按照 x 从小到大排序，如果 x 相同，则按照 y 从小到大排序
        trees.sort()

        hull = [0]  # hull[0] 需要入栈两次，不标记
        used = [False] * n
        # 求凸包的下半部分
        for i in range(1, n):
            while len(hull) > 1 and cross(trees[hull[-2]], trees[hull[-1]], trees[i]) < 0:
                used[hull.pop()] = False
            used[i] = True
            hull.append(i)
        # 求凸包的上半部分
        m = len(hull)
        for i in range(n - 2, -1, -1):
            if not used[i]:
                while len(hull) > m and cross(trees[hull[-2]], trees[hull[-1]], trees[i]) < 0:
                    used[hull.pop()] = False
                used[i] = True
                hull.append(i)
        # hull[0] 同时参与凸包的上半部分检测，因此需去掉重复的 hull[0]
        hull.pop()
```

复杂度分析

- 时间复杂度：$O(n \log n)$，其中 n 为数组的长度。首先需要对数组进行排序，时间复杂度为 $O(n \log n)$，每次添加栈中添加元素后，判断新加入的元素是否在凸包上，因此每个元素都可能进行入栈与出栈一次，最多需要的时间复杂度为 $O(2n)$，因此总的时间复杂度为 $O(n \log n)$。
- 空间复杂度：$O(n)$，其中 n 为数组的长度。首先该解法需要快速排序，需要的栈空间为 $O(\log n)$，用来标记元素是否存在重复访问的空间复杂度为 $O(n)$，需要栈来保存当前判别的凸包上的元素，栈中最

30 条评论 ›_ 最热

编辑 预览 评论

精选评论(3)

SNORLAX L4 11 小时前

真好，一看就是我不会的

27 踩 查看 1 条回复 回复 分享 举报

为她命 L1 21 小时前

看到困难，就知道明天可以睡个懒觉了

6 踩 回复

Jam L6 (编辑过) 11 小时前

解题思路 本题为凸包问题，采用通用的思路求解，对点进行x和y轴大小排序，判断最近三点的形成的边与点的位置关系，如何才能满足包含最近的其他点，要分开讨论本题分为上凸包和下凸包两个不同情况，分别从左边的点开始和从最右边的点开始判断当前点在前面两个点连线的左边还是右边，在右边能获得更大的凸包。

代码

```
class Solution:
    def outerTrees(self, trees: List[List[int]]) -> List[List[int]]:
```

```
        # 判断添加点 r 是否在前面两点连线的右边
        def check_valid(points, r):
            points.append(r)
            while len(points) >= 3 and sign(*points[-3:]) < 0:
                points.pop(-2)
            return points

        trees.sort(key = lambda p: (p[0], p[1]))
        lower = reduce(check_valid, trees, [])
        upper = reduce(check_valid, trees[::-1], [])
        return lower + list(filter(lambda p: p not in lower, upper))
```

6 踩 回复

评论(30)

廿陆書生 L2 11 小时前

Jarvis里第一个行列式里，右下角qy写成了py

4 踩 回复 分享 举报

知心猛男 L5 14 小时前

计算几何😵，我直接狠狠学习！

3 踩 回复

coco2602002 21 小时前

4月23号的题目get!

1 踩 查看 1 条回复 回复

赞 踩 回复

kotori L4 1 小时前

方法一的C++代码是错的，没有考虑起始点在线段中点以及线段上同时存在多个点的情况。例如样例 `[[2,2],[2,0],[2,4],[4,2],[3,3]]`，别的语言就没看了。附上自己的代码，思路和方法一差不多，有很多改进空间，欢迎大佬指教

```
class Solution {
public:
    vector<vector<int>> outerTrees(vector<vector<int>>& trees) {
        int n = trees.size();
        if(n <= 3) return trees;

        vector<vector<int>> ans;

        int fir = -1;       //找最左当中最下点作为起始点
        for(int i = 0; i < n; ++i){
            if(fir == -1 || trees[fir][0] > trees[i][0] || (trees[fir][0] == trees[i][0] && trees[fir][1] > trees[
        }

        int start = fir;
        int sec = -1;

        vector<int> vis(n);

        while(sec != start){
            //find sec
            sec = (fir + 1) % n;
            for(int i = 0; i < n; ++i){
                if(get_angle(trees[fir], trees[sec], trees[i]) < 0) sec = i;
            }
            vis[sec] = 1;
```

```
                if(get_angle(trees[fir], trees[sec], trees[i]) == 0){
                    same.push_back({get_dis(trees[fir], trees[i]), i});
                    vis[i] = 1;
                }
            }

            sort(same.begin(), same.end());

            for(auto& [_, idx] : same){
                ans.push_back(trees[idx]);
                sec = idx;
            }

            fir = sec;
        }

        return ans;
    }

    int get_angle(vector<int>& a, vector<int>& b, vector<int>& c){
        return (b[0] - a[0]) * (c[1] - b[1]) - (b[1] - a[1]) * (c[0] - b[0]);
    }

    int get_dis(vector<int>& a, vector<int>& b){
        return (a[0] - b[0]) * (a[0] - b[0]) + (a[1] - b[1]) * (a[1] - b[1]);
    }
};
```

赞 踩 回复 分享 举报

赞 踩 回复

terrygmx L3 1 小时前

mark一个，以后多学学

赞 踩 回复

青清山 L3 2 小时前

坐等卷王的题解

赞 踩 回复

青清山 L3 3 小时前

困难题 虽迟但到

1 踩 回复 分享 举报

futureterryqin L5 3 小时前

jarvis官方解释中有一处错误:

```
cross(p,q,r)=(qx-px)*(ry-py)-(qy-py)*(rx-qx)    //官方原文
```

```
cross(p,q,r)=(qx-px)*(ry-qy)-(qy-py)*(rx-qx)     //正确写法
```

1 踩 回复

Carl_Czerny L3 3 小时前

这题扔置顶打卡就离谱......能死记硬背以防万一更好（虽然我一直认为面试考超纲知识点的板子题没有筛选价值），直接收藏+cv也没啥问题

2 踩 回复

inuter L2 3 小时前

我滴个亲娘诶!

2 踩 回复