587. Erect the Fence

Add to List

Description Hints Submissions Solutions

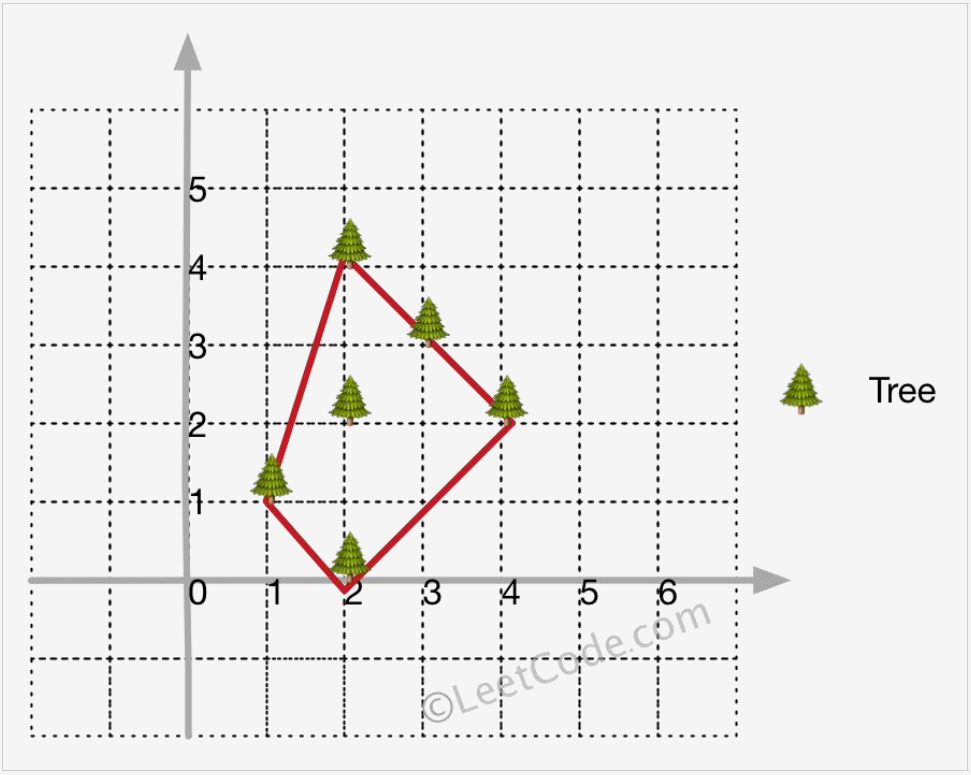
- Total Accepted: **995**
- Total Submissions: **3445**
- Difficulty: **Hard**
- Contributors:vishal51

There are some trees, where each tree is represented by (x,y) coordinate in a two-dimensional garden. Your job is to fence the entire garden using the **minimum length** of rope as it is expensive.

The garden is well fenced only if all the trees are enclosed. Your task is to help find the coordinates of trees which are exactly located on the fence perimeter.

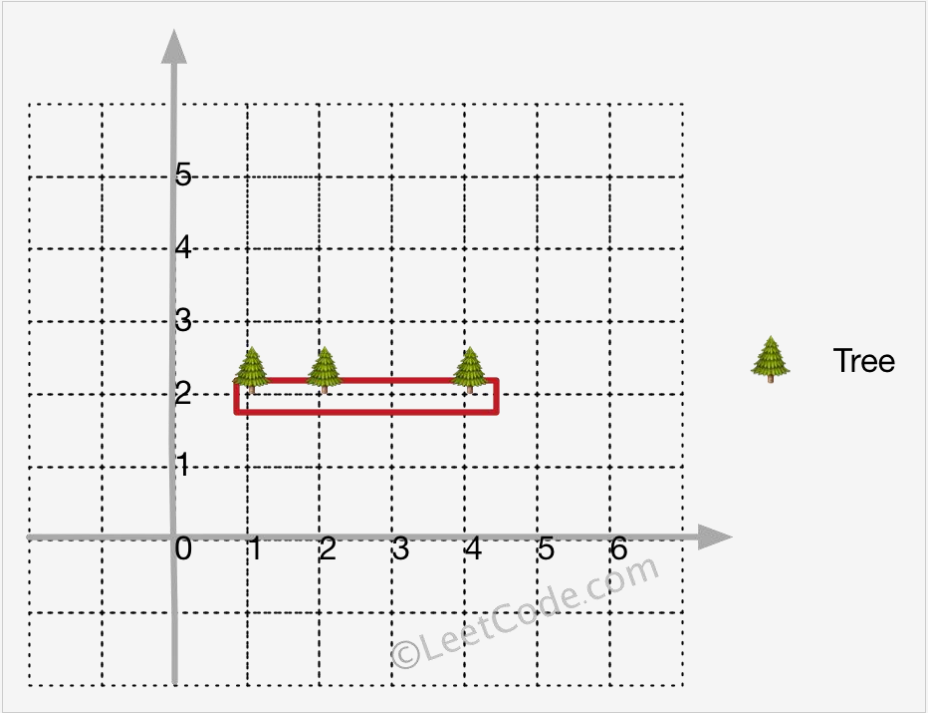
Example 1:

Input: `[[1,1],[2,2],[2,0],[2,4],[3,3],[4,2]]` **Output:** `[[1,1],[2,0],[4,2],[3,3],[2,4]]` **Explanation:**



Example 2:

Input: `[[1,2],[2,2],[4,2]]` **Output:** `[[1,2],[2,2],[4,2]]` **Explanation:**



Even you only have trees in a line, you need to use rope to enclose them.

Note:

1. All trees should be enclosed together. You cannot cut the rope to enclose trees that will separate them in more than one group.
2. All input integers will range from 0 to 100.
3. The garden has at least one tree.
4. All coordinates are distinct.
5. Input points have **NO** order. No order required for output.

Subscribe to see which companies asked this question.

```
class Solution {
public:
    vector<Point> outerTrees(vector<Point>& points) {
        if (points.size() <= 3) return points;
        vector<Point> res;
        res.push_back(left_most(points));
        Point pre = res[0], next;
        while (1) {
            next = pre;
            for (auto & p : points) {
                if (p.x == pre.x && p.y == pre.y) continue;
                double cur = direction(pre, p, next);
                if (cur == 0) {
                    if (next.x == pre.x && next.y == pre.y) next = p;
                    else if (inbetween(pre, p, next) || (visited(next, res) &&
next.x != res[0].x && next.y != res[0].y)) next = p;
                }
                else if (cur > 0) next = p;
            }
            if (visited(next, res)) return res;
            res.push_back(next);
            pre = next;
        }
    }
private:
    Point left_most (vector<Point>& points) {
        Point res;
        int l_m = INT_MAX;
        for (auto & p : points) {
            if (p.x < l_m) {
                l_m = p.x;
```

```
                res = p;
            }
        }
        return res;
    }
    bool visited(Point& l, vector<Point>& res) {
        for (auto & p : res) {
            if (p.x == l.x && p.y == l.y) return true;
        }
        return false;
    }
    double direction(Point & a, Point & b, Point & c) {
        Point ab (b.x - a.x, b.y - a.y);
        Point bc (c.x - b.x, c.y - b.y);
        return ab.x * bc.y - ab.y * bc.x;
    }
    bool inbetween(Point& a, Point & b, Point & c) {
        return (b.x >= a.x && b.x <= c.x || b.x >= c.x && b.x <= a.x) && (b.y >=
a.y && b.y <= c.y || b.y >= c.y && b.y <= a.y);
    }
};
```