```cpp
// Assignment 9 - Convex Hull
// You can solve it using Graham's Scan

typedef pair<int, int> Point;

class Solution {
public:
    /* Helper function to calculate the orientation of three points (p, q, r)
    0 -> p, q, r and collinear,
    1 -> clockwise,
    2 -> counterclockwise
    */
    int orientation(const Point& p, const Point& q, const Point& r) {
        int val = (q.second - p.second) * (r.first - q.first) -
                  (q.first - p.first) * (r.second - q.second);
        if (val == 0)
            return 0;
        return (val > 0) ? 1 : 2;
    }

    // Function to compute the square of the distance between two points
    int distance(const Point& p1, const Point& p2) {
        return (p1.first - p2.first) * (p1.first - p2.first) +
               (p1.second - p2.second) * (p1.second - p2.second);
    }

    vector<vector<int>> outerTrees(vector<vector<int>>& trees) {
        // Graham's scan convex hull algorithm
        vector<Point> points;
        for (const auto& tree : trees) {
            points.emplace_back(tree[0], tree[1]);
        }

        // Step 1: Find the point with the lowest y-coordinate (or leftmost in
        // case of tie)
        Point start = *min_element(
            points.begin(), points.end(), [](const Point& p1, const Point& p2) {
                return (p1.second < p2.second) ||
                       (p1.second == p2.second && p1.first < p2.first);
            });

        // Step 2: Sort points based on polar angle with 'start' point
        auto polar_angle = [&start](const Point& p) {
            return atan2(p.second - start.second, p.first - start.first);
        };

        sort(points.begin(), points.end(),
             [&start, &polar_angle, this](const Point& p1, const Point& p2) {
                 double angle1 = polar_angle(p1);
                 double angle2 = polar_angle(p2);
                 if (angle1 == angle2) {
                     return distance(start, p1) < distance(start, p2);
                 }
                 return angle1 < angle2;
             });

        // Step 3: Initialize the convex hull with the first three points
        vector<Point> hull = {start};

        for (size_t i = 1; i < points.size(); ++i) {
            while (hull.size() > 1 &&
                   orientation(hull[hull.size() - 2], hull.back(), points[i]) ==
                       1) {
                hull.pop_back(); // Pop last point if we turn clockwise or if
                                 // collinear
            }
            hull.push_back(points[i]);
        }

```

```
        for (size_t i = points.size() - 1; i >= 0; --i) {
            if (orientation(start, hull.back(), points[i]) == 0) {
                hull.push_back(points[i]);
            }
            if (i == 0)
                break;
        }

        vector<vector<int>> result;
        unordered_set<string> unique_points;

        for (const auto& p : hull) {
            string point_str = to_string(p.first) + "," + to_string(p.second);
            if (unique_points.find(point_str) == unique_points.end()) {
                result.push_back({p.first, p.second});
                unique_points.insert(point_str);
            }
        }

        return result;
    }
};
```