Convex Hull construction using Graham's Scan

Table of Contents

In this article we will discuss the problem of constructing a convex hull from a set of points.

Consider N points given on a plane, and the objective is to generate a convex hull, i.e. the smallest convex polygon that contains all the given points.

The algorithm used here is **Graham's scan** (proposed in 1972 by Graham) with improvements by Andrew (1979). The algorithm allows for the construction of a convex hull in $O(N \log N)$ using only comparison, addition and multiplication operations. The algorithm is asymptotically optimal (as it is proven that there is no algorithm asymptotically better), with the exception of a few problems where parallel or online processing is involved.

Description

The algorithm first finds the leftmost and rightmost points A and B. In the event multiple such points exist, the lowest among the left (lowest Y-coordinate) is taken as A, and the highest among the right (highest Y-coordinate) is taken as B. Clearly, A and B must both belong to the convex hull as they are the farthest away and they cannot be contained by any line formed by a pair among the given points.

Now, draw a line through AB. This divides all the other points into two sets, S1 and S2, where S1 contains all the points above the line connecting A and B, and S2 contains all the points below the line joining A and B. The points that lie on the line joining A and B may belong to either set. The points A and B belong to both sets. Now the algorithm constructs the upper set S1 and the lower set S2 and then combines them to obtain the answer.

To get the upper set, we sort all points by the x-coordinate. For each point we check if either - the current point is the last point, (which we defined as B), or if the orientation between the line between A and the current point and the line between the current point and B is clockwise. In those cases the current point belongs to the upper set S1. Checking for clockwise or anticlockwise nature can be done by checking the orientation.

If the given point belongs to the upper set, we check the angle made by the line connecting the current point and the last point in the upper convex hull, with the line connecting the last point and the second last point of the convex hull that we have been simultaneously creating. If the angle is not clockwise, we remove the most recent point added to the upper convex hull as the current point will be able to contain the previous point once it is added to the convex hull.

The same logic applies for the lower set S2. If either - the current point is B, or the orientation of the lines, formed by A and the current point and the current point and B, is counterclockwise - then it belongs to S2.

If the given point belongs to the lower set, we act similarly as for a point on the upper set except we check for a counterclockwise orientation instead of a clockwise orientation. Thus, if the angle made by the line connecting the current point and the last point in the lower convex hull, with the line connecting the last point and the second last point of the convex hull is not counterclockwise, we remove the most recent point added to the lower convex hull as the current point will be able to contain the previous point once added to the hull.

The final convex hull is obtained from the union of the upper and lower convex hull, and the implementation is as follows.

Implementation

```
struct pt {
    double x, y;
};

bool cmp(pt a, pt b) {
    return a.x < b.x || (a.x == b.x && a.y < b
}

bool cw(pt a, pt b, pt c) {
    return a.x*(b.y-c.y)+b.x*(c.y-a.y)+c.x*(a.
}

bool ccw(pt a, pt b, pt c) {
    return a.x*(b.y-c.y)+b.x*(c.y-a.y)+c.x*(a.
}

void convex_hull(vector<pt>& a) {
    if (a.size() == 1)
        return;

    sort(a.begin(), a.end(), &cmp);
    pt p1 = a[0], p2 = a.back();
    vector<pt> up, down;
    up.push_back(p1);
    down.push_back(p1);
    for (int i = 1; i < (int)a.size(); i++) {
        if (i == a.size() - 1 || cw(p1, a[i],
            while (up.size() >= 2 && !cw(up[up
                up.pop_back();
            up.push_back(a[i]);
        }
        if (i == a.size() - 1 || ccw(p1, a[i],
                while(down.size() >= 2 && !ccw(dow
                    down.pop_back();
                down.push_back(a[i]);
            }
        }

        a.clear();
        for (int i = 0; i < (int)up.size(); i++)
            a.push_back(up[i]);
        for (int i = down.size() - 2; i > 0; i--)
            a.push_back(down[i]);
    }
```

Practice Problems

- Kattis - Convex Hull
- Kattis - Keep the Parade Safe
- Timus 1185: Wall
- Usaco 2014 January Contest, Gold - Cow Curling