Activity 4.2

José Carlos Martínez Núñez, Tania Sayuri Guizado Hernández

November 11, 2022

Contents

Problem Description

In teams of two people and using the computational geometry programming technique, build a C++ program that implements the algorithm to determine the smallest convex polygon in a set of points by applying Graham's Scan algorithm. The program receives a set of n points located on a Cartesian plane.

The output of the program will be an array of points (vertices) in counterclockwise order defining the convex polygon. The array should be displayed in the standard output.

Algorithm Implementation

The algorithm to determine the smallest convex polygon in a set of points is based on the following steps:

1. Find the point with the lowest y-coordinate. If there is a tie, choose the point with the lowest x-coordinate.
2. Sort the points by polar angle in counterclockwise order around the lowest point.
3. Push the first three points onto a stack.
4. For each remaining point p, do the following:
    1. While the angle formed by the points on the top of the stack, the point next to the top of the stack, and p makes a non-left turn, pop the top point off the stack.
    2. Push p onto the stack.
5. The points on the stack define the convex hull.

In order to display the points in counterclockwise order, we need to reverse the order of the points in the stack. That is why in the previous algorithm instead of using a stack, we use a vector.

```
int main(int argc, char const *argv[])
{
    int n;
    cin >> n;

    if (n < 3)
    {
        cout << "Convex Hull not possible" << endl;
        return 0;
```

```
    }

    vector<Point> points(n);
    for (int i = 0; i < n; i++)
    {
        cin >> points[i].x >> points[i].y;
    }

    int minY = points[0].y;

    int minIndex = 0;

    for (int i = 1; i < n; i++)
    {
        int y = points[i].y;
        if (y < minY || (y == minY && points[i].x < points[minIndex].x))
        {
            minY = y;
            minIndex = i;
        }
    }

    Point temp = points[0];
    points[0] = points[minIndex];
    points[minIndex] = temp;

    initialPoint = points[0];

    sort(points.begin() + 1, points.end(), [](Point p1, Point p2)
         {
             Orientation orientation = getOrientation(initialPoint, p1, p2);

             if (orientation == COLINEAR)
                 return distance(initialPoint, p1) < distance(initialPoint, p2);
             return orientation == COUNTERCLOCKWISE; });

    vector<Point> hull;
```

```
    hull.push_back(points[0]);
    hull.push_back(points[1]);
    hull.push_back(points[2]);

    for (int i = 3; i < n; i++)
    {
        while (getOrientation(hull[hull.size() - 2], hull[hull.size() - 1], points[i]) !=
        ↪ COUNTERCLOCKWISE)
        {
            hull.pop_back();
        }
        hull.push_back(points[i]);
    }

    for (int i = 0; i < hull.size(); i++)
    {
        cout << "(" << hull[i].x << ", " << hull[i].y << ")" << endl;
    }

    return 0;
}
```

The time complexity of the algorithm is $O(n \log n)$.

Test Cases

Each test case consists of a set of points located on a Cartesian plane. The first line of each test case contains an integer n that represents the number of points. The next n lines contain two integers x and y that represent the coordinates of the points. The coordinates of the points are integers between -100 and 100. All of them were generated randomly using the following website: https://onlineintegertools.com/generate-integer-pairs.

- `2_points.txt`: 2 points, the convex hull is not possible.
- `10_points.txt`: 10 points. The convex hull is made up of 6 points.
- `50_points.txt`: 50 points. The convex hull is made up of 10 points.
- `100_points.txt`: 100 points. The convex hull is made up of 13 points.

The test cases and their results are located in the `test_cases` and `results` folders respectively.