Heaven's Light is Our Guide

Computer Science & Engineering
Rajshahi University of Engineering & Technology

Lab Manual

Module-05
Course Title : Sessional based on CSE 2201
Course No. : CSE 2202

Experiment No. 5

Name of the Experiment: Design and Complexity analysis of Convex Hull.

Date: 5th Cycle

Algorithms (Divide-and-Conquer method)**:**

- **Convex Hull**

Convex Hull:
Find Convex Hull from a Set of Points (Brute Force Algorithm):

There are two variants of the convex hull problem:
[1]. Obtain the vertices of the convex hull (these vertices are also called extreme points);
[2]. Obtain the vertices of the convex hull in some order (clockwise, for example).
Now a simple algorithm is to be describing to find the extreme points of a given set S of points in the plane. To check whether a particular point p ε S is extreme, look at each possible triplet of points and see whether p lies in the triangle formed by these three points. If p lies in any such triangle, it is not extreme, otherwise it is. If there are n points then there are ${}^{n}C_3$ possible triangles.

$${}^{n}C_3 = \frac{n!}{3!(n-3)!} = \frac{1}{6} \times n(n-1)(n-2) = \Theta(n^3)$$

So it takes Θ(n3) times to determine whether a given point is an extreme point or not. Since there are n points, this algorithm runs in a total of Θ(n4) times(Naive Brute force).

In the implementation of the problem, for each pair of point's p1 and p2, a brute force algorithm determines whether all other points lie to the same side of a straight line through p1 and p2. If so, the line segment connecting p1 and p2 is a part of the convex hull's boundary.

Algorithm

```
1    for each of n(n-1)/2 pairs of distinct points
2    for each of the other n – 2 points
3    find the sign of ax + by – c
```

The time efficiency of this algorithm is $\Theta(n^3)$ $^{\text{(Better Brute force)}}$.

QuickHull Algorithm:
Under average circumstances the algorithm works quite well, but processing usually becomes slow in cases of high symmetry or points lying on the circumference of a circle. The algorithm can be broken down to the following steps:

1. Find the points with minimum and maximum x coordinates, those are bound to be part of the convex hull.
2. Use the line formed by the two points to divide the set in two subsets of points, which will be processed recursively.
3. Determine the point, on one side of the line, with the maximum distance from the line. The two points found before along with this one form a triangle.

4. The points lying inside of that triangle cannot be part of the convex hull and can therefore be ignored in the next steps.
5. Repeat the previous two steps on the two lines formed by the triangle (not the initial line).
6. Keep on doing so on until no more points are left, the recursion has come to an end and the points selected constitute the convex hull.

Alternative algorithm
This is a minor change towards the initial steps of the algorithm which might help save some computation time. One can note that the points with minimum and maximum y coordinates are also bound to be part of the convex hull, therefore, there are four points (or less depending on whether these points are the same or different) which are bound to be part of the convex hull. It is possible to discard directly all points lying inside the quadrilateral found within these four points (or less). Also, a first check needs to be made to check the number of points at start, if there are only three points, then the algorithm can be solved in O(1) since the three points are part of the convex hull.
The algorithm can be broken down to the following steps:

1. If the set of point is of size 3, then the three points are part of the convex hull and the algorithm can be ended.
2. Find the points with minimum and maximum x coordinates and with minimum and maximum y coordinates, those are bound to be part of the convex hull.
3. The points lying inside of the quadrilateral formed by the previous extrema cannot be part of the convex hull and can therefore be ignored in the next steps.
4. Use the line formed by a pair of extrema from step two to divide the remaining set in two subsets of points, which will be processed recursively.
5. Determine the point, on one side of the line, with the maximum distance from the line. The two points found before along with this one form a triangle.
6. The points lying inside of that triangle cannot be part of the convex hull and can therefore be ignored in the next steps.
7. Repeat the previous two steps on the two lines formed by the triangle (not the initial line).
8. Keep on doing so on until no more points are left, the recursion has come to an end and the points selected constitute the convex hull.

Graham's Scan:

Pseudocod
First, define
```
# Three points are a counter-clockwise turn if ccw > 0, clockwise if
# ccw < 0, and collinear if ccw = 0 because ccw is a determinant that
# gives twice the signed  area of the triangle formed by p1, p2 and p3.
function ccw(p1, p2, p3):
    return (p2.x - p1.x)*(p3.y - p1.y) - (p2.y - p1.y)*(p3.x - p1.x)
```
Then let the result be stored in the array `points`.
```
let N           = number of points
let points[N+1] = the array of points
swap points[1] with the point with the lowest y-coordinate
sort points by polar angle with points[1]

# We want points[0] to be a sentinel point that will stop the loop.
let points[0] = points[N]
```

```
# M will denote the number of points on the convex hull.
let M = 1
for i = 2 to N:
    # Find next valid point on convex hull.
    while ccw(points[M-1], points[M], points[i]) <= 0:
          if M > 1:
                  M -= 1
          # All points are collinear
          else if i == N:
                  break
          else
                  i += 1

    # Update M and swap points[i] to the correct place.
    M += 1
    swap points[M] with points[i]
```

This pseudocode is adapted from Sedgewick and Wayne's *Algorithms, 4th edition*.
The check inside the while statement is necessary to avoid the case when all points in the set are collinear.

Task:

1. Find out the complexity of the above algorithms.
2. Code the above algorithm in any language(i.e. C/C++/Java)
3. Find the running time for a set of points list (let size 1000, 5000,10000, 15000 etc.
4. Write down a report on it.

Recommended Exercise:
Programming Exercises of Chapter 3: "Divide-And-Conquer" of "Fundamentals of Computer Algorithm", Ellis Horowitz, Sartaj Sahni, Sanguthevar Rajasekaran.