Jarvis March

Description

This algorithm is also called the gift-wrapping algorithm. The most intuitive algorithm which comes to our mind when we think of wrapping a string around some nails hammered into a piece of wood. This algorithm was discovered by R.A. Jarvis in 1973 and has been one of the most used algorithms for calculating convex hulls.

Methodology

The Jarvis March algorithm has a principled very similar to selection sort. In selection sort, we select the least number in the array and add it to the sorted array. Similarly, in the Jarvis-March algorithm, we find the point that is the rightmost point from where we stand and add this to the array containing the points for the convex hull.

To start off, we select the vertex that has the least x-coordinate (the leftmost point). This point is guaranteed to be in the convex hull array. If there are 2 such points having the same x-coordinate, select the one which has the least y-coordinate.

Basically, at every iteration, imagine yourself standing on the vertex and looking at all the other vertices in the group. You will select the the rightmost point, add it to your collection called “convex-hull” and then move onto that point. Repeat the same process till the right most point is the first point you stood on.

The orientation checker function is based on comparing the slopes of 2 lines. Since the algorithm is all about finding the rightmost point. If the slope of the line joining point1 from the vertex you are “standing on” is more than the slope of the line joining point2 to that vertex, then point2 is more to the right than point1. If there is a case when both the points, point1 and point2, are collinear from the vertex, then the point selected is that whose distance from the vertex is most. Thus there is also an inbuilt distance method which calculates and compares the distance in the case as mentioned earlier.

Pseudocode

Algorithm Jarvis\_March(N): // *N points in (x,y) form*

Lowest :=1

For i := 2 to N:

If i[x] < Lowest[x]:

Lowest := i

p := Lowest

convex\_hull.add(p)

repeat

cursor := p + 1 // *p and cursor points cannot be the same*

for j := 2 to N :

if orientation\_checker (p,j,cursor): // *checks if the point j lies to the right of the cursor point*

cursor := j

convex\_hull.add(cursor) // *cursor is the new rightmost point, so add to the convex\_hull and move onto it*

p := cursor

until p := Lowest // *stop process when you return to starting point*

Time complexity

Average Case: For each point that is added to the convex\_hull boundary, the algorithm runs operations that have a time complexity of O(N), where N is the total number of points in the cluster. If there are going to be H points in the convex hull, the time complexity for the Jarvis-March algorithm is O(H\*N).

Worst Case: This happens when H=N i.e all the N points lie in a circle

Best Case: Jarvis March is a very effective algorithm if we know that the number of points in the convex hull are going to be very small. Therefore if we know that H<<N, the algorithm almost has a linear time complexity of O(N).

Jarvis-March algorithm is therefore output-sensitive. The smaller the output, the faster the algorithm runs.

Quickhull

Description

Quickhull is an improvement on the naïve gift-wrapping algorithm. It was invented by C. Bradford Barber, David P. Dobkin, and Hannu Huhdanpaa in 1996. Quickhull is a divide and conquer algorithm that works very similarly to quicksort by partitioning the dataset into 2 sets that individually go through the algorithm and the final results are clubbed into one forming the convex hull. The special property of quickhull is that the time complexity of the algorithm is actually determined by how the algorithm runs. Unlike the quicksort algorithm, this is not a randomized algorithm and each dataset runs in a way that the big-Oh notation cannot be easily calculated. This will be better described later after explaining the functionalities of the algorithm.

Methodology

As mentioned earlier, quickhull is a divide and conquer algorithm.

Assume you have a set of N points in the form of (x,y)

The following are the basic steps of the Quickhull algorithm:

1. Find 2 starter points. These 2 points are those with the minimum (called *lowest*) and the maximum x-coordinates (called *highest*). These points are the rightmost and the leftmost points of the collection and so have to be a part of the convex hull. In case there are 2 or more points that have the same minimum x-coordinate, choose the one with the least y-coordinate. Similarly, if there are 2 or more points with the same maximum x-coordinate choose the one with the biggest y-coordinate.
2. Imagine a line joining these 2 extreme points, dividing the entire dataset into sets, one lying to the left of the line and the. other lying on the right of this line. Divide and conquer algorithm works simultaneously on these 2 sets and concatenates the result into 1.

A picture containing map, ball, white, table

Description automatically generated

1. On the left side (considering one side for explanation purposes), we find the point that is farthest from the line lowest and highest. This point (called *furthest*) if the furthest point perpendicular to the line and by reason is also present in the convex hull.
2. We imagine a triangle between lowest, highest and furthest. All points in the left side, which lie inside this triangle are by default removed from consideration for the convex hull. Only the points that remain outside the triangle are possible candidates.

A picture containing ball

Description automatically generated

1. We now repeat the steps 2,3,4 recursively for the lines made by joining lowest and furthest and the line made by joining furthest and highest. Consider the points the lie on the left side of these lines only. When carrying out this recursion on the right side of the main division, consider only the right side points of any recursive division formed.

A close up of a map

Description automatically generated

Pseudocode

Input = a set S of n points

Assume that there are at least 2 points in the input set S of points

**function** QuickHull(*S*) **is**

*// Find convex hull from the set S of n points*

Convex Hull := {}

Find left and right most points, say A & B, and add A & B to convex hull

Segment AB divides the remaining (*n* − 2) points into 2 groups *S1* and *S2*

where *S1* are points in *S* that are on the right side of the oriented line from *A* to *B*,

and *S2* are points in *S* that are on the right side of the oriented line from *B* to *A*

FindHull(*S1*, *A*, *B*)

FindHull(*S2*, *B*, *A*)

Output := Convex Hull

**end function**

**function** FindHull(*Sk*, *P*, *Q*) **is**

*// Find points on convex hull from the set Sk of points*

*// that are on the right side of the oriented line from P to Q*

**if** *Sk* has no point **then**

**return**

From the given set of points in *Sk*, find farthest point, say *C*, from segment *PQ*

Add point *C* to convex hull at the location between *P* and *Q*

Three points *P*, *Q*, and *C* partition the remaining points of *Sk* into 3 subsets: *S0*, *S1*, and *S2*

where *S0* are points inside triangle PCQ, *S1* are points on the right side of the oriented

line from *P* to *C*, and *S2* are points on the right side of the oriented line from *C* to *Q*.

FindHull(*S1*, *P*, *C*)

FindHull(*S2*, *C*, *Q*)

**end function**

Time complexity

Average Case: The time complexity for quickhull is O(N\*log(r)). Here, N is the total number of points in the dataset and r is the total number of “*furthest*” points that are found by recursion of the algorithm. Because we do not know the exact number of points that will be considered for recursion we cannot exactly calculate the time complexity.

Best case: The best case for the quickhull algorithm will be the case where the data is in such a format that the number of points that can be the extremes are very small in number. As compared to the total number of points. Imagine a dense cluster of points and a few outliers on the boundaries.

Worst Case: The worst case situation is of complexity O(N2). This situation happens when the points are arranged in a circle. This will call the recursion N times and will consider every last point as the furthest till the recursion runs.

References

1. <http://jeffe.cs.illinois.edu/teaching/compgeom/notes/01-convexhull.pdf>
2. <https://www.cs.princeton.edu/courses/archive/fall08/cos226/demo/ah/JarvisMarch.html>
3. <http://www.personal.kent.edu/~rmuhamma/Compgeometry/MyCG/ConvexHull/jarvisMarch.htm>
4. <https://en.wikipedia.org/wiki/Gift_wrapping_algorithm>
5. <https://algorithmtutor.com/Computational-Geometry/Convex-Hull-Algorithms-Jarvis-s-March/>
6. <https://iq.opengenus.org/quick-hull-convex-hull/>
7. <https://www.cise.ufl.edu/~ungor/courses/fall06/papers/QuickHull.pdf>
8. <https://en.wikipedia.org/wiki/Quickhull>