2-D Convex Hull

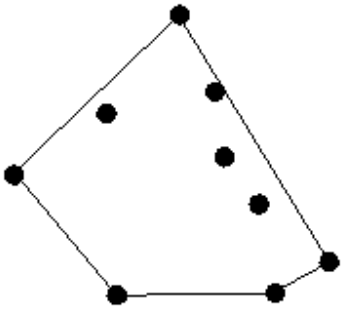
Prerequisite

- Geometry

The Abstraction

Given: A collection of points in the plane, find the convex polygon with smallest area such that each point is contained within (or on the boundary of) the polygon.

Observe that the vertices of such a polygon will be points from the given collection.



Sample Problem: Cow Herding

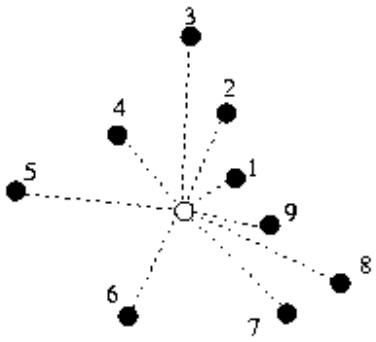
Farmer John wants a fence to keep the pesky local college students from tipping his cows over as they sleep. Each of his cows has a favorite spot for grazing grass, and Farmer John would like the fenced enclosure to include all of these favorite locations. Farmer John would like to enclose a convex figure, as it makes it a lot easier for the cows to make it to their grazing locations.

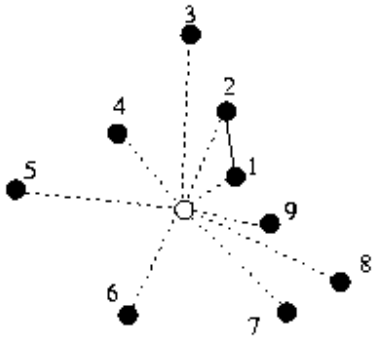
Help Farmer John by calculating the fence which encloses the minimum amount of area while still including all of the cows' favorite dining locations.

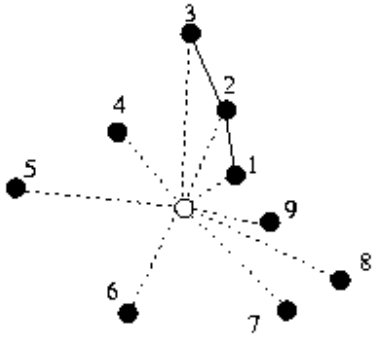
The Algorithm

Several algorithms solve the two dimensional convex hull algorithm. Here, we'll present only the ``gift-wrapping'' algorithm, which is probably the easiest to code and the easiest to remember.

The basic idea is to add the points in clockwise or counterclockwise order around some point within the final answer, checking to see if any angles greater than 180 degrees are created, which would make the final figure concave. Every time three points in a row appear such that the angle created by those three points is greater than 180 degrees, delete the middle point. Check the angle is done by calculating the cross product of vectors along two consecutive edges.







Find a point which will be within the convex hull:

- Calculate the angle that each point makes with the x axis (within the range 0 to 360 degrees)
- Sort the points based on this angle
- Add the first two points
- For every other point except the last point
- Make it the next point in the convex hull
- Check to see if the angle it forms with the previous two points is greater than 180 degrees
    - As long as the angle formed with the last two points is greater than 180 degrees, remove the previous point
- To add the last point
    - Perform the deletion above,
    - Check to see if the angle the last point forms with the previous point and the first point is greater than 180 degrees or if the angle formed with the last point and the first two points is greater than 180 degrees.

    - If the first case is true, remove the last point, and continue checking with the next-to-last point.
    - If the second case is true, remove the first point and continue checking.
    - Stop when neither case is true.
- The adding of the points is linear time, as is the calculation of the angles. Thus, the run-time is dominated by the time of the sort, so gift-wrapping runs in O(n log n) time, which is provably optimal.

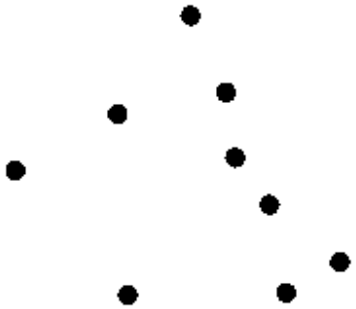
Pseudocode

Here is the pseudocode for this convex hull algorithm:

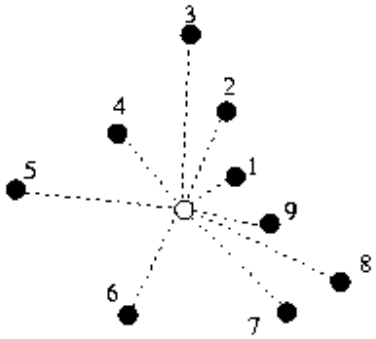
```
 # x(i), y(i) is the x,y position
     #    of the i-th point
     # zcrossprod(v1,v2) -> z component
     #             of the vectors v1, v2
     #  if zcrossprod(v1,v2) < 0,
     #     then v2 is "right" of v1
     #   since we add counter-clockwise
     #     <0 ->  angle > 180 deg
 1   (midx, midy) = (0, 0)
 2   For all points i
 3     (midx, midy) = (midx, midy) +
           (x(i)/npoints, y(i)/npoints)
 4   For all points i
 5     angle(i) = atan2(y(i) - midy,
                        x(i) - midx)
 6     perm(i) = i

 7   sort perm based on the angle() values
     # i.e., angle(perm(0)) <=
              angle(perm(i)) for all i

     # start making hull
 8   hull(0) = perm(0)
 9   hull(1) = perm(1)
10   hullpos = 2
11   for all points p, perm() order,
                except perm(npoints - 1)
12     while (hullpos > 1 and
           zcrossprod(hull(hullpos-2) -
13         hull(hullpos-1),
           hull(hullpos-1) - p) < 0)
14           hullpos = hullpos - 1
15       hull(hullpos) = p
16       hullpos = hullpos + 1

     # add last point
17   p = perm(npoints - 1)
18   while (hullpos > 1 and
          zcrossprod(hull(hullpos-2) -
19        hull(hullpos-1),
          hull(hullpos-1) - p) < 0)
20     hullpos = hullpos - 1

21   hullstart = 0
22   do
23     flag = false
24     if (hullpos - hullstart >= 2 and
            zcrossprod(p -
25          hull(hullpos-1),
            hull(hullstart) - p) < 0)
26       p = hull(hullpos-1)
27       hullpos = hullpos - 1
28       flag = true
29     if (hullpos - hullstart >= 2 and
           zcrossprod(hull(hullstart) - p,
           hull(hullstart+1) -
           hull(hullstart)) < 0)
30        hullstart = hullstart + 1
31        flag = true
32   while flag
33   hull(hullpos) = p
34   hullpos = hullpos + 1
```

Sample Run

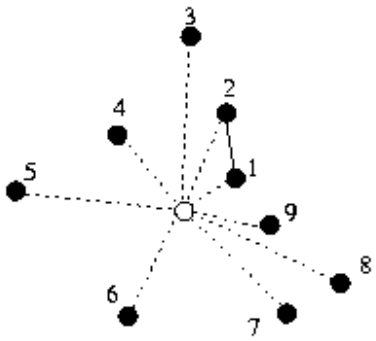
For the sample run, use these points:



Select a center, calculate angles, and sort.

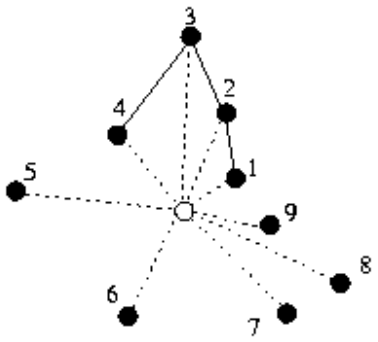


Now, start by adding the first two points.

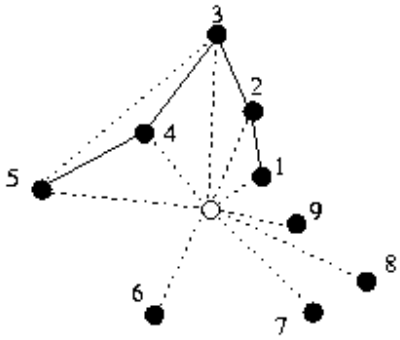


Now, add the third point. Since this does not create an angle of greater than 180 degrees with the first two points, we just have to add the point.

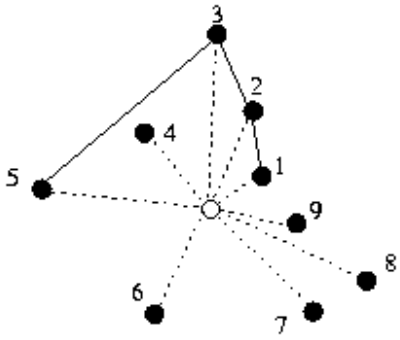
Add the fourth point. Again, no angle greater than 180 degrees was created, so no further work is necessary.



Add the fifth point.

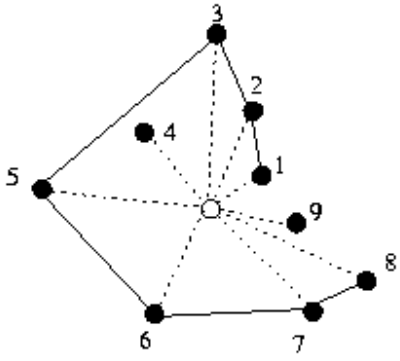


Since the third, fourth, and fifth points together create an angle of greater than 180 degrees (a "right" turn), we remove the fourth point.

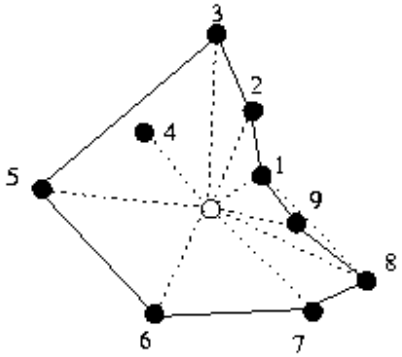


The second, third, and fifth points do not create an angle of greater than 180 degrees, so we are done with adding the fifth point.

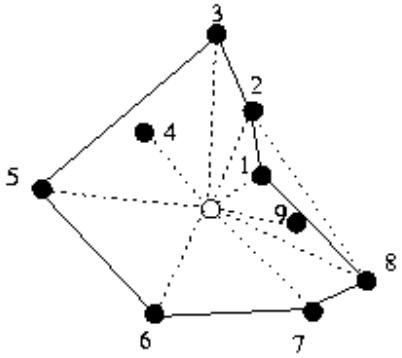
Add the sixth, seventh, and eighth points. None of these require additional work.



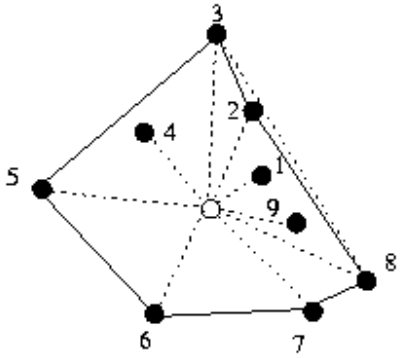
Next, add the last point.



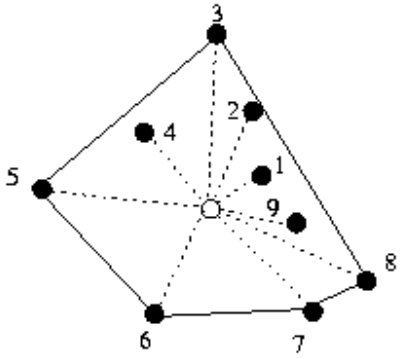
The eighth, ninth, and first point create a "right" turn; remove the ninth point.



The seventh, eighth, and first points are fine, but the eighth, first and second points have a "right" term, so we must remove the first point.



Now the eighth, second and thirst points have a "right" term, so we must remove the second point.



No more violations exist, so we are done, and we have the convex hull of the given points.

Problem Cues

Problems which ask for enclosing points within a polygon are usually convex hull problems. If the problem asks for a minimum area convex polygon or a polygon of minimum circumference, it's almost certainly asking for the convex hull.

Extensions

Unfortunately, this algorithm does not extend in an obvious manner to three dimensions. Fortunately, the three dimensional algorithms are all fairly complicated (four and higher+ dimensions are just plain ugly), so it's unlikely you'll get asked to do it there.

This algorithm no longer works if you limit the created polygon in any way (e.g., no more than n points or must be a rectangle).

Sample Problems

Trees Problem [IOI 1991, problem 2]

Given: a collection of trees, surround it with wire such that you use the minimal amount of wire. Calculate the trees that will be the vertices of the polygon, the length of wire required, and whether the farmer's house, which is at a given location, lies inside of, outside of, or across the polygon.

Analysis: The vertices of the polygon and the length of wire required follow fairly directly above. The farmer's house is specified as an axis-aligned rectangle, so it takes a bit of geometry to determine if all the points are within the convex hull, without the convex hull, or some are within and some without, which gives you the answer you wanted. See the Geometry pamphlet for clues on these kinds of intersections.

Cheapskate Moat Building

Given: A collection of polygon houses, calculate the minimum length moat that encloses all but at most one of them.

Analysis: To enclose a given polygon in convex polygon is equivalent to enclosing all of the vertices of the given polygon. This is very slightly a combination problem, which requires both a loop and a convex hull builder. For each house, delete the house and enclose the remaining vertices in a convex hull. Pick the house whose resulting convex hull is smallest. Note that only deleting houses which share a vertex with the convex hull of the entire set would help if some of the houses don't share a vertex with that convex hull.

Back to USACO Training Gateway | Comment or Question