Algorithms

Example Sheet 6

Petar Veličković

Easter Term 2017

Warm-up

1. Given two flows f_1 and f_2 in a flow network $G = (V, E)$, let $f_3 : V \times V \to \mathbb{R}$ be defined as the sum of the two flows, i.e.:

   $$f_3(x, y) = f_1(x, y) + f_2(x, y)$$

   Is f_3 a flow in G or not? Why?

2. Provide an example of a small flow network for which the Ford-Fulkerson algorithm may take at least n iterations (where n is an arbitrarily chosen integer), if unlucky choices are made. How would you fix this problem?

3. For a convex hull problem on $n = 10^6$ vertices, with the hull being expected to consist of $h \approx 1000$ vertices, would you use Graham's scan or Jarvis' march? Which factors influenced your decision?

4. Graham's scan considers points sorted by an angle. Carefully explain, by analysing each case separately, how the algorithm will behave when two points have the same angle. Does this modify your answer to Exercise 3?

Exercises

1. Consider an undirected graph $G = (V, E)$. You are tasked with finding the smallest number of edges that you need to *remove* from E to disconnect the graph (i.e. such that there exist two vertices x and y that are not connected). Describe an algorithm to compute this number, analyse its time complexity, and prove its correctness.

2. A city (consisting of n places, with m different roads connecting pairs of places) has been contaminated by a deadly gas! Each of the n places initially has some (known) number of survivors, and they may freely move along the roads; each road requires a specific (known) number of hours to traverse. In order to perform a rescue operation, the mayor will send

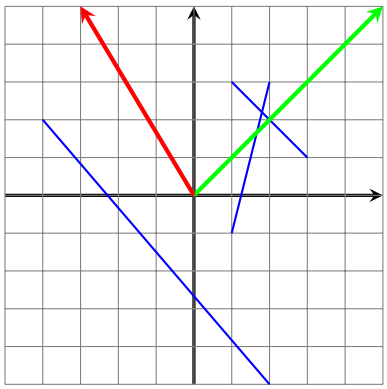
h helicopters in. For each helicopter, we know the time (expressed in number of hours from the start) and place when it will land, as well as the number of people it can carry. Provide algorithms to compute:

(a) The maximal number of people that can be rescued;

(b) **(OPTIONAL)** If everyone can be rescued, the minimal number of hours required to do this.

3. You are given n segments in the plane, and want to fire a ray from the origin. Provide an algorithm to determine the minimal and maximal number of segments you can "hit" in this way, and analyse its time complexity. You may assume that no segment will cross the origin.

   Below is an example, where the maximal and minimal numbers of segments are two and zero, respectively (highlighted by green and red rays):



4. Consider the *on-line convex hull problem*; rather than knowing the n points in advance, you're given one point at a time and need to be able to iteratively update the convex hull as new points are added. Naïvely recomputing the hull each time would require $O(n^2 \log n)$ overall time (if e.g. Graham's scan was used)—provide an $O(n^2)$ algorithm.

5. Solve *Exercise 8* from the pre-sheet.

Implementation

This week's implementation exercise is a "computational geometry playground" of sorts. You will use standard subroutines to progressively build up solutions to two very important problems in this space:

- Determining whether a polygon is *simple* (does not self-intersect).
- Determining whether a convex polygon *contains a given point.*

In this context, polygons are represented by a list P of n vertices, such that P_i is connected to P_{i+1}. Specially, P_n is connected to P_1.

Each step of this implementation exercise should be briefly tested on a few "edge" inputs of your choosing. You may find it useful to visualise these inputs somehow, but this is not a requirement.

For starters, a few trivial subroutines:

1. Implement a method to compute $\overrightarrow{AB} \times \overrightarrow{AC}$ for points A, B and C in the plane (for simplicity, you may assume that these points will always have integer coordinates).
2. Implement a method to determine whether point C is *to the left* (counterclockwise/CCW) of the vector $\overrightarrow{AB}$ (for given points A and B).
3. Implement a method to determine whether points A, B and C are collinear.

Now, to determine whether a polygon is simple:

1. Implement a method to determine whether points C and D are on *opposite sides* of the vector $\overrightarrow{AB}$ (for given points A and B).
2. Implement a method to determine whether point C is contained in the rectangle spanned by points A and B.
3. **(IMPORTANT)** Implement a method to determine whether segments AB and CD intersect. Carefully analyse the edge cases!
4. Determine whether a given polygon is simple, by considering whether any of its two segments intersect. Note the special case with adjacent segments, $P_{i-1}P_i$ and P_iP_{i+1}, which are supposed to intersect in P_i only!

Finally, to determine whether a convex polygon contains a given point:

1. **(IMPORTANT)** Implement a method to determine whether point X is contained within a triangle ABC. Carefully analyse the edge cases!
2. Reduce the problem to testing whether a point is contained within n separate triangles (you may omit implementing this step).
3. Devise and implement an algorithm which will always test *exactly one* triangle in this manner, after performing $O(\log n)$ additional work.
   [*Hint:* Binary search.]