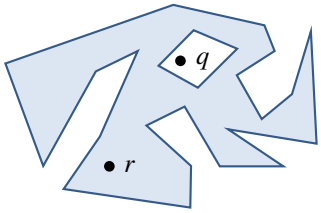
Example sheet 8

Space.

Algorithms—DJW*—2018/2019

For the exam you will not be required to remember the details of the geometric algorithms from Section 8. Instead, use these questions to practice your bug-spotting rigour—you don't want to be the programmer who let a nuclear reactor explode because you missed a corner case. Questions labelled ∘ are warmup questions. Questions labelled ∗ involve more work or more thinking.

Question 1°. Consider a horizontal line starting from q and going infinitely to the right. How many times does it cross an edge? What about from r? Devise an algorithm to detect whether a given point is outside a given shape (like q) or inside (like r), where the shape is specified as a list of non-intersecting polygons and each polygon is specified as a list of points.



Question 2*. Discuss the handling of corner cases in your answer to question 1.

Question 3. Two line segments are moving. The first line segment has endpoints $(-0.2+0.1t, -0.1+0.1t)$ and $(0.8+0.1t, -0.4+0.2t)$, the second has endpoints $(-0.36+0.52t, 1.1-0.3t)$ and $(2.1-0.3t, 0.1-0.3t)$, and $t \geq 0$. Do they collide? If so, at what time t? Write the most concise pseudocode you can, to solve this problem for arbitrary coefficients.

Question 4 (FS64). Imagine a complex CAD model consisting of millions of points. Which of the two algorithms we've studied, Jarvis's march and Graham's scan, would you use to compute the convex hull if you expected it to have about 1,000 vertices? What is the rough threshold at which you'd switch to using the other algorithm? (And is this an answerable question?)

Question 5°. To compute the convex hull of a set of points P, we started by picking a point $q_0 \in P$ with the lowest y-coordinate, breaking ties by choosing the larger x-coordinate. Give an example to show that, without this tie breaking rule, q_0 might not be a corner point. (A corner point is a point $p \in P$ such that p is not in the convex hull of $P \setminus \{p\}$.)

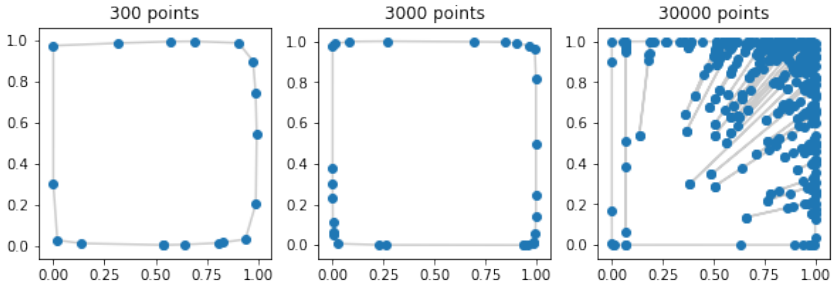
Question 6. Graham's algorithm scans points in order, according to a certain angle. Explain carefully what happens when two points have exactly the same angle. Does the code in lecture notes still work? *[Hint. The code can crash! Make sure you identify all the cases where the code crashes or gives an incorrect answer.]*

Question 7. I generated `n` points uniformly in the unit square $[0,1] \times [0,1]$, using 16-bit floating point numbers for speed.

```
import numpy as np
points = np.random.uniform(size=(n,2)).astype(np.float16)
```

*Question 4 is from Prof Stajano.

I then ran Graham's scan and plotted the convex hull. Here are my results, for a range of values of n. What's going on? *[Hint. You may find it useful to implement the algorithm yourself, to help you track down the bug.]*



Question 8*. Generate n points uniformly in the unit square $[0, 1] \times [0, 1]$, and find the number of corner points in the convex hull. How does your answer grow with n? *You should easily be able to work with a million points on a low-end laptop.*

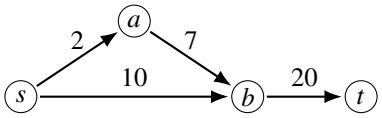
Question 9°. The A^* routing algorithm is for finding routes from one vertex to another, on a directed graph with edge weights that is embedded in 2D space. It assumes we know a heuristic distance function $h()$, and it uses this to prioritize which paths to explore. For example, for a Roomba robot vacuum cleaner trying to plan a route to the next room, $h(v \text{ to } t)$ might be the straight line distance from v to t, whereas the true distance $d(v \text{ to } t)$ is the length of the shortest path taking obstacles into account. Read the description at `http://www.redblobgames.com/pathfinding/a-star/introduction.html`.

Given a graph, and a start vertex s, and an end vertex t, A^* proceeds exactly like Dijkstra's algorithm, except that it uses a different key to sort the priority queue:

$$\texttt{key}(v) = v.\texttt{distance} + h(v \text{ to } t)$$

where v.`distance` is its best guess so far for the distance from s to v, and $h(v \text{ to } t)$ is the heuristic distance function.

Use the A^* algorithm, by hand, to find a route from s to t in the following graph. Use the heuristic $h(a \text{ to } t) = 20$ and $h(b \text{ to } t) = 3$. *[Use the* `dijkstra` *algorithm as given in lecture notes, which allows vertices to re-enter* `toexplore` *after they've been popped. Some textbooks present a different version of Dijsktra's algorithm, which does not allow vertices to re-enter* `toexplore`*. In the conventional setting, as described in section 5.4, both versions have identical behaviours.]*



Question 10*. Assuming that the heuristic distance is always less than or equal to the true distance, i.e. that

$$h(v \text{ to } t) \leq d(v \text{ to } t) \qquad \text{for all } v,$$

prove that A^* is correct, i.e. that when it terminates t.`distance` $= d(s \text{ to } t)$. *[Hint. Look at the proof for case (i) of Bellman's algorithm. It's an induction, but not the same type of induction as used in Dijsktra's algorithm]*