Computational Geometry
Winter Term 26/27
`scale.iti.kit.edu`



Exercise Sheet 3

Submission due by 2026-12-03

Problem 1: Triangulation

$2 + 2 + 2 = 6$ points

Part (a) Let P be a simple polygon with n vertices. There may be multiple ways to triangulate P. Into how many triangles is P partitioned by any of these triangulations? *(Prove your answer!)*

Part (b) Now consider a polygon P that may contain holes. Prove that P can always be triangulated.

Part (c) Let P be a polygon that may contain holes. The boundary of the polygon is defined by n vertices. There may be multiple ways to triangulate P. Into how many triangles is P partitioned by any of these triangulations? *(Prove your answer!)*

Hint: In a planar graph with n vertices and m edges, it holds that $m \leq 3n - 6$. Equality holds if and only if the boundary of every face (including the outer one) is a triangle.

Problem 2: Completing 2D Linear Programming

4 points

Give an algorithm that, given a 2-dimensional linear program (LP) with n constraints, determines whether it is unbounded in $\mathcal{O}(n)$ time. How can your algorithm be used to complete the 2D LP algorithm presented in the lecture?

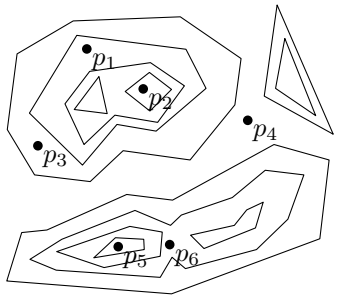
Problem 3: Ski Huts

5 points

You are planning your next skiing holiday. The most important criterion for choosing a ski resort is (for different values of x) the number of ski huts located at an altitude of x meters. To make it easier to assess how interesting a ski resort is, you are therefore looking for an efficient algorithm that can extract this information from ski resort maps.

The maps consist of a set of polygons (representing contour lines) and a set of points (representing the ski huts). The polygon edges do not intersect, but polygons may be nested inside other polygons. For a given ski resort, the algorithm should determine, for each value of x, how many ski huts are contained in exactly x polygons.

Example:



Here, we have one hut (p_4) at height 0, one hut at height 1, two huts at heigt 2, and two huts at height 4.

Problem 4: Mission Impossible **5 points**

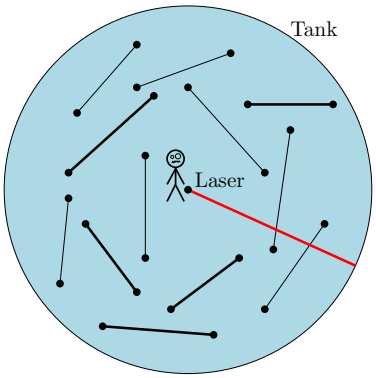
Agent Kim is in a dire situation. Her enemy, Dr. Meta, has trapped her inside a cylindrical tank filled entirely with water. Luckily, she has a laser (disguised as a whiteboard marker) that could break through the tank wall and save her.

Unfortunately, Kim is surrounded by n glass panes, each of which reduces the laser's power depending on its strength.

In a panic, Kim rotates the laser to a randomly chosen angle every $O(\log(n) + k)$ seconds, where k is the number of glass panes the laser would hit in that direction. After each reorientation, Kim must decide whether to activate the laser. Since a laser in the form of a whiteboard marker only has a very small battery, she has only one chance to fire the laser in the correct direction.

Fortunately, Agent Kim had $O(n \log n)$ preprocessing time during which she could study the positions of the glass panes — but not their strengths! For any given angle, she must now determine the total strength of the panes the laser would intersect in order to decide whether the beam could reach the tank wall.

How did Kim make use of her preprocessing time? And how can she now quickly determine which k panes the laser would intersect in a given direction before panic sets in and she rotates to the next direction?



Problem 5: Range-Tree Implementation

5 bonus points

In this exercise, you have the opportunity to implement a range tree as seen in the lecture. The points you collect in this exercise do not count towards the total points of this sheet, but can be used to compensate for missing points on this or another sheet.

Implement a 2D range tree using a programming language of your choice. The construction time should be in $\mathcal{O}(n \log^2 n)$, and the query time should be in $\mathcal{O}(\log^2(n) + k)$ per query. You should need no optimizations to solve our test cases, but you can implement optimizations (from the lecture or other sources) if you like.

Briefly explain in your submission how you implemented the range tree, e.g. which internal data structures are you using and which optimizations have you implemented. Further, put your solutions in the `.pdf` document using the format that is explained below, and submit your code as a separate file.

You can find our test cases as a `.zip` file on our website. It contains two folders: *sample* contains test cases with solutions and *test* only contains test cases. Submit your solutions for the test cases in the *test* folder.

Input format: The first line consists of two numbers, n and q, the number of points and the number of queries. In each of the next n lines, the x-coordinate and the y-coordinate of a point are given. Both are integers. Note that there might be multiple points with the same coordinates.

The next q lines consist of four integers $\min_x$, $\max_x$, $\min_y$, $\max_y$ each describing a rectangular query.

Output format: For each query, we are looking for all points (x, y) with $\min_x \leq x \leq \max_x$ and $\min_y \leq y \leq \max_y$. Since this set can be very large, we compress the solution in the following way: Let P_i be the set of points within the given rectangle of query i. Compute three numbers for query i: $|P_i|$, the size of the solution, $x_i = (\sum_{(x,y)\in P_i} x) \mod (10^9 + 7)$, the sum over all x-coordinates in P_i modulo $10^9 + 7$, and $y_i = (\sum_{(x,y)\in P_i} y) \mod (10^9 + 7)$, the sum over all y-coordinates in P_i modulo $10^9 + 7$.

The output for a test case only consists of three numbers: The sum over all $|P_i|$ (modulo $10^9 + 7$), the sum over all x_i (modulo $10^9 + 7$), and the sum over all y_i (modulo $10^9 + 7$). We use the modulo to avoid any overflows (do not apply it too late in your calculations).