CMSC 754: Fall 2023 Dave Mount

Homework 3: Voronoi/Delaunay, Arrangements, and Search

Handed out Thu, Nov 2. Due at the start of class on **Tue, Nov 28**. Late homeworks are not accepted (unless an extension has been prearranged) so please turn in whatever you have completed by the due date.

Unless otherwise specified, you may assume that all inputs are in *general position*. Whenever asked to give an algorithm running in $O(f(n))$ time, you may give a *randomized algorithm* whose expected running time is $O(f(n))$. If your algorithm involves a plane sweep of a line arrangement and runs in time $O(n^2 \log n)$, you may assume that there exists a topological plane sweep variant that runs in time $O(n^2)$.

Problem 1. (10 points) This problem demonstrates an important application of computational geometry to the field of amphibian navigation. A frog who is a afraid of the water wants to cross a stream that is defined by two horizontal lines $y = y^-$ and $y = y^+$. On the surface there are n circular lily pads whose centers are at the points $P = \{p_1, \ldots, p_n\}$. The frog crosses by hopping across the circular lily pads. The lily pads grow at the same rate, so that at time t they all have radius t (see Fig. 1(a)). Help the frog out by presenting an algorithm that in $O(n \log n)$ time determines the earliest time t^* such that there exists a path from one side of the stream to the other by traveling along the lily pads (see Fig. 1(b)).

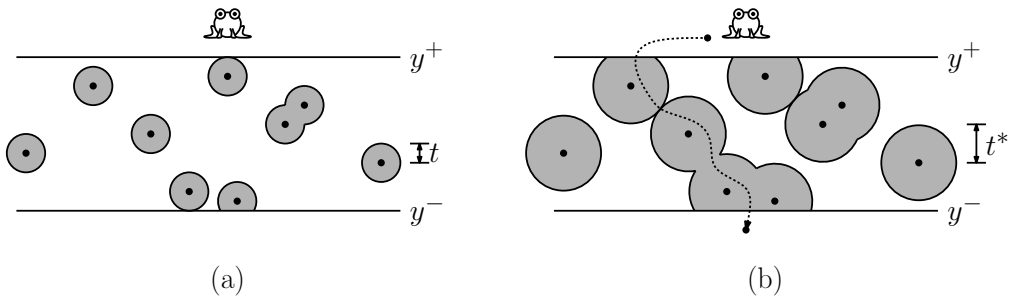

Figure 1: Frog crossing.

Problem 2. (15 points) Define a *slab* to be the region bounded by two parallel (nonvertical) lines, and define its *height* to be the vertical distance between these lines. You are given three sets of points R, W, and B (red, white, and blue) in $\mathbb{R}^2$. A *3-color slab* is a pair of parallel lines such that the closed region bounded between these two lines contains exactly one point from each of R, W, and B in its interior (see Fig. 2). (We do not care what are the colors of the points that lie on the two lines that bound the slab.)

Update (11/8): It was pointed out that with the above definition, an optimal solution may fail to exist. (This is illustrated in Fig. 2(c), where the blue slab is a valid 3-color slab, but it can be made progressively higher by rotating the two bounding lines. When the slab reaches maximum height, shown in yellow, the blue point now lies on the slab's upper boundary, and hence it is not a valid 3-color slab!) To circumvent this issue, we allow solutions like the one shown above, where the slab itself may not be valid, but an infinitesimal perturbation would result in a valid 3-color slab.

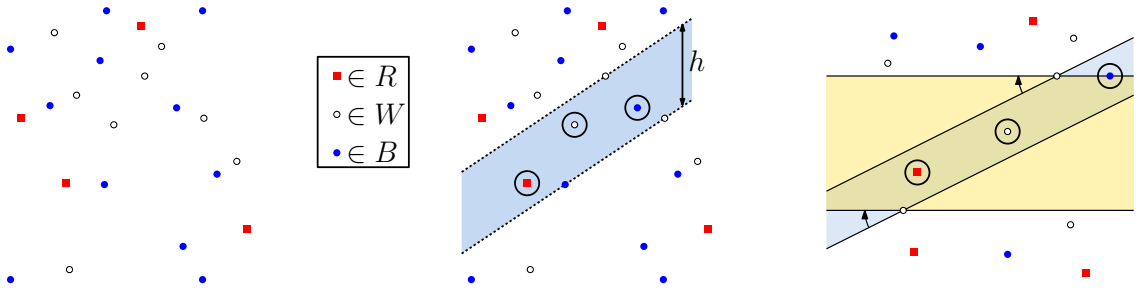

Figure 2: A 3-color slab of height h.

(a) (5 points) Assuming the standard dual transformation (mapping point (a, b) to line $y = ax - b$), explain what a 3-color slab of height h corresponds to in the dual setting.

(b) (5 points) There are generally (uncountably) many 3-color slabs. What local conditions must a 3-color slab satisfy in order to have maximum height? (You may assume that the points are in general position. You may also assume that the given point set has no 3-color slab of infinite height.)

(c) (5 points) Present an algorithm, which given inputs R, W, and B, determines whether there exists a 3-color slab. If not, indicate this. If so, compute the 3-color slab of maximum height. Derive your algorithm's running time and justify its correctness. (**Hint:** For full credit, aim for a running time of $O(n^2 \log n)$ time with $O(n)$ space, where $n = |R| + |W| + |B|$. You may make the same assumption as in part (b).)

Problem 3. (10 points) While kd-trees and range trees are good for axis-aligned queries, they do not perform very well when the query shape is more general. In a *halfplane range counting query*, we are given a point set $P = \{p_1, \ldots, p_n\} \in \mathbb{R}^2$ to be preprocessed into a data structure. Given any halfplane, expressed say by a linear inequality, $H = \{(x, y) : ax + by \leq c\}$, the objective is to count how many points of P lie within H, that is, $|P \cap H|$ (see Fig. 3(a)).

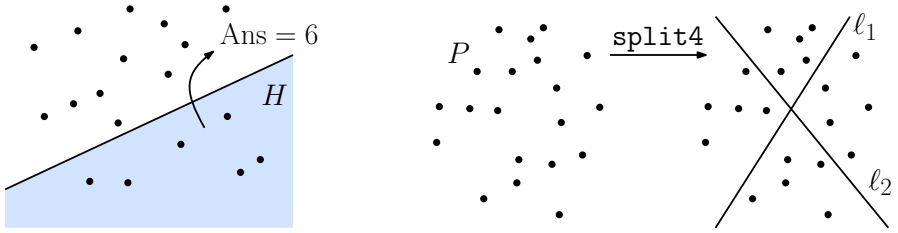

Figure 3: Halfplane range counting.

(a) (5 points) Show that kd-trees do not provide an efficient worst-case solution for halfplane range queries. Give an example of a point set P and a halfplane H such that any (reasonable) query algorithm will require $O(n)$ time to answer this query. (**Hint:** Design

the point set along with a hyperplane so that the line bounding H intersects a constant fraction of the leaves of the kd-tree. We will accept a drawing as a solution, but make it large enough that it is clear how to generalize it to arbitrarily large values of n.)

(b) (5 points) Let's explore a simple (but not optimal) approach for answering halfplane range counting queries. Let us suppose that we have as a "black-box" subroutine, called `split4`, which given any n-element point set P in $\mathbb{R}^2$, generates two lines ℓ_1 and ℓ_2, that partitions P into four subsets (depending on which sides of these lines the points lie) such that each set has at most $\lceil n/4 \rceil$ points (see Fig. 3(b)). (Such a subroutine exists. See the challenge problem.)

Given a point set P, explain how to use the `split4` function to generate a partition tree for P. Show that tree has space $O(n)$ and height $O(\log n)$. Prove that the tree supports halfplane range counting queries in time $O(n^{\log_4 3}) \approx O(n^{0.792})$. (**Hint**: Each node of the tree will have four children. The Master Theorem may be helpful in analyzing the query time.)

Problem 4. (15 points) In this problem, we will consider how to use/modify range trees to answer a number of queries. In each case, the input is an n-element point set P in $\mathbb{R}^2$, and for each explain what points are stored in the range tree, what the various layers of the range tree are, and how queries are answered. Finally, justify your algorithm's correctness and derive its storage and running time as a function of n. You may make the general-position assumption that no point of P lies on the boundary of any range.

(a) (5 points) A *sheared rectangle* is defined by two points $q^- = (x^-, y^-)$ and $q^+ = (x^+, y^+)$. The range shape is a parallelogram that has two horizontal sides and two sides with a slope of -1 (see Fig. 3(a)). The upper left corner is q^- and the lower right corner is q^+. The answer to a *sheared rectangle query* is the number of points of P that lie within the parallelogram.

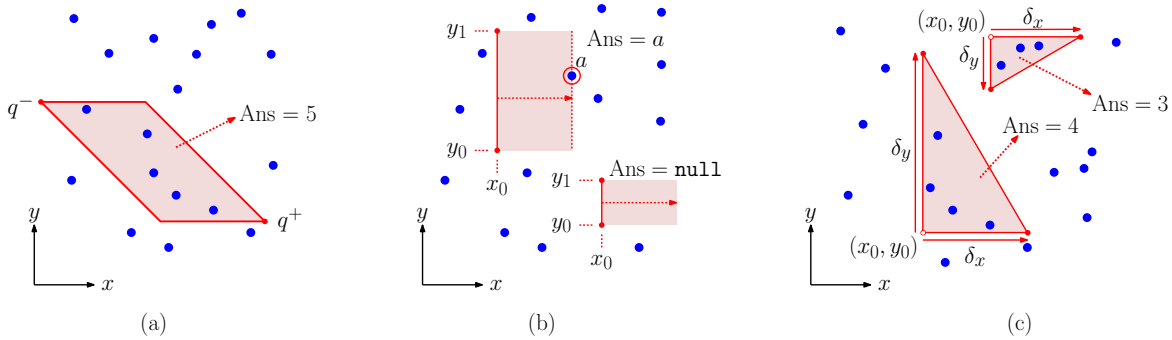

Figure 4: Using range trees to answer various queries.

(b) (5 points) In a *vertical segment-sliding query* (VSS), you are given a vertical line segment with x-coordinate x_0 and endpoints at y-coordinates y_0 and y_1, where $y_0 < y_1$. The answer to the query is the first point that is hit if the segment is translated to the right (see Fig. 4(b)). If the segment can be slid infinitely far to the right without hitting a point of P, the query returns the special value `null`.

(c) (5 points) A *30-60 right triangle* is a right triangle where vertices have angles 30, 60, and 90 degrees. A 30-60 right triangle range is such a triangle, where two of the sides are parallel to the coordinate axes. In a *30-60 triangle query*, you are given such a triangle, and the answer is the number of points of P lying within the triangle (see Fig. 4(c)). The triangle is specified by giving the coordinates (x_0, y_0) of the vertex forming the right angle, and two additional nonzero scalars δ_x and δ_y (which may be positive or negative). The other two vertices are located at $(x_0 + \delta_x, y_0)$ and $(x_0, y_0 + \delta_y)$. Note that there are eight distinct triangle shapes, depending on the signs of δ_x and δ_y and which is larger. Since we are only interested in 30-60 triangles, you may assume that $|\delta_x/\delta_y| = \sqrt{3}$ or $1/\sqrt{3}$.

Hint: In all cases, it is possible to answer queries in $O(\log^c n)$ time, for some constant c. It is sufficient to adapt the standard range-tree approach, and in particular, you do not need to use cascading search.

Challenge problems count for extra credit points. These additional points are factored in only after the final cutoffs have been set, and can only increase your final grade.

Challenge Problem 1. Present a polynomial time algorithm for the function `split4`. That is, given a set P of n points in the plane, your algorithm will output two lines ℓ_1 and ℓ_2 that partition P into four subsets, each of size at most $\lceil n/4 \rceil$.

Hint: First, translate the points so the y-axis partitions P into two subsets of equal size. The y-axis is the line ℓ_1. All you need is to show how to compute ℓ_2. For that, it may be helpful to consider the problem in the dual setting.

Challenge Problem 2. The data structure described in 3(b) can be improved upon. Show how to modify the data structure of 3(b) by replacing the 4-ary tree with a binary tree. Show that, up to constant factors, the query time $T(n)$ satisfies the following recurrence:

$$T(n) \;=\; T(n/2) + S(n/2) + 1 \qquad \text{and} \qquad S(n) \;=\; T(n/2) + 1.$$

Here $T(n)$ denotes the query time for a subtree containing n points, and $S(n)$ denotes the query time for a subtree with the added condition that for one of its two children, all the points lie within H or none of them do. Derive a solution to this recurrence. (**Hint:** Somewhat surprisingly, the answer is related to the Fibonacci sequence.)