CSCI1950-J: Final Exam (125 points; 1 point = 1 point on previous exams)

Out: Tuesday, May 3, 2011
Problem 5 updated to make Report() possible only for the root interval
Due: Tuesday, May 10, 2011

Please return your writeup for this exam to Saara Moskowitz in CIT 546 by 4pm on the due date.

David will hold hours Sunday, May 8 from 2pm to 4 in CIT 227.

Readings: Section 8.8 of the textbook

This is a strictly non-collaborative assignment. You may only discuss the questions and answers with the course staff. You are permitted to use the textbook, but no external resources.

All work should be typed, preferably in LaTeX.

"Analyzing" an algorithm means proving it correct and bounding its running time.

Problem 1 (25 points)

Consider the following problem. The input consists of an n-edge convex polygon P and a line ℓ, both in the Euclidean plane. If ℓ does not intersect P, then the output is the vertex $q \in P$ closest to ℓ. If ℓ does intersect P, then the output is $\perp$.

Design and analyze algorithms that preprocess P and then use the results of preprocessing so as to support repetitive queries with different ℓ. Preprocessing should take time $O(n)$. Each query should take time $O(\log n)$.

Problem 2 (25 points)

The known linear-time algorithms for computing the convex hull of a simple polygon are somewhat subtle. In this problem, we will explore a simpler but incorrect algorithm.

Let's say that a polygon (not necessarily simple) on vertices $p_1, \ldots, p_n$ is *of Graham type* in case for all $1 \leq i \leq n$, the walk $p_{i-1}p_ip_{i+1}$ is a "left turn", where we take $p_0 = p_n$ and $p_{n+1} = p_1$. The Graham scan algorithm in essence starts with a star-shaped polygon on the given points and then iteratively removes vertices p_i such that $p_{i-1}p_ip_{i+1}$ is not a left turn. The final polygon, namely the convex hull, is both simple and of Graham type.

Suppose now that the initial polygon is simple but not star-shaped. The final polygon is still necessarily of Graham type, but it is not necessarily simple. Give an example of a simple polygon and a sequence of vertex removals satisfying Graham's condition resulting in a non-simple polygon of Graham type. (Hint: work backward.)

Problem 3 (25 points)

Design and analyze an algorithm that given $3n$ points $a_1, b_1, c_1, \ldots, a_n, b_n, c_n$ in the Euclidean plane, finds i, j, k so as to minimize the radius of the circle passing through a_i, b_j, c_k. Get the best running time you can.

Problem 4 (25 points)

Design and analyze an efficient preprocessing/query algorithm pair for the following problem. Given are n fixed line segments in the Euclidean plane. After preprocessing the segments, answer queries of the form "Does line ℓ intersect any segment?"

Problem 5 (25 points)

Given n axis-aligned rectangles $R_1, \ldots, R_n$, determine the area of the set

$$\left\{p : p \in \mathbb{R}^2,\ \left|\{i : p \in R_i\}\right| \equiv 1 \pmod 2\right\}.$$

In other words, what area of the plane is covered by exactly an odd number of rectangles?

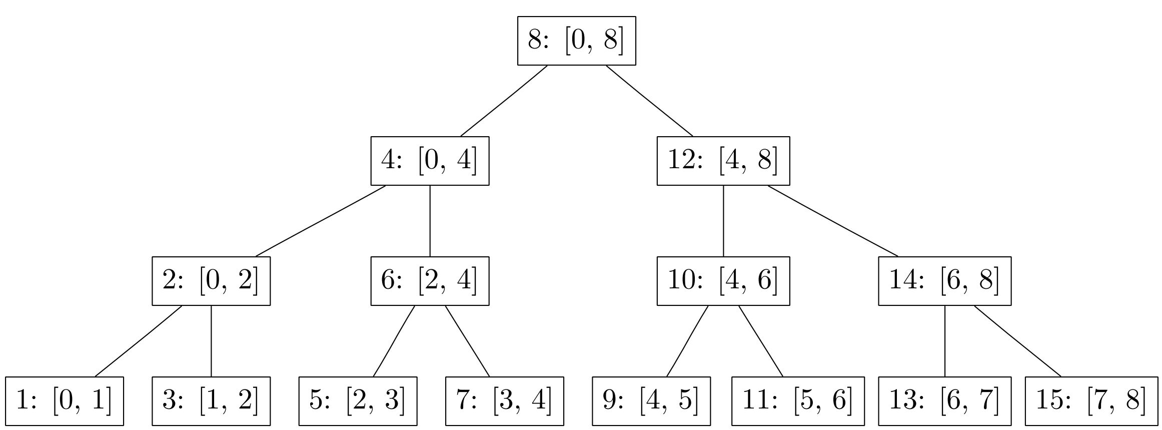
There is an efficient sweep-line algorithm based on a data structure for the dynamic version of the problem in one dimension. This data structure in turn can be implemented efficiently by means of a segment tree. Your task is to write **pseudocode** for the following methods, which span the non-generic aspects of this data structure.

- `Initialize`(k). The nodes of the segment tree are numbered $1, \ldots, k$. This method is called once, before any of the other methods. Its running time should be $O(k)$.
- `Report`$()$. This method returns the total length covered by exactly an odd number of intervals. Its running time should be $O(1)$.
- `Toggle`(i)
- `Update`(j, ℓ, r). Observe first that insertion and removal are indistinguishable. When inserting an interval $[a, b]$, code that you don't have to write finds a list of nodes $i_1, \ldots, i_m$ such that $[a, b]$ is a pairwise almost disjoint union of the corresponding segments. It invokes `Toggle`$(i_1), \ldots,$ `Toggle`(i_m). Then, for all ancestors j of $i_1, \ldots, i_m$ in the order those ancestors would be visited by a post-order traversal, the other code invokes `Update`(j, ℓ, r), where ℓ is the left child of j and r is the right child.

  The running time of both `Toggle` and `Update` should be $O(1)$.

One method is provided for you: `Length`(i), which returns the length of the segment corresponding to node i.

Let's consider a possible execution. Code that you don't have to write constructs the following segment tree and calls `Initialize`(15).



Suppose that the interval $[1, 6]$ is to be inserted. Code that you don't have to write decomposes $[1, 6] = [1, 2] \cup [2, 4] \cup [4, 6]$ and makes the following calls.

```
Toggle(3)
Toggle(6)
Toggle(10)
Update(2, 1, 3)
Update(4, 2, 6)
Update(12, 10, 14)
Update(8, 4, 12)
```

At this point, `Report`() is 5. Suppose now that the interval $[3, 8]$ is to be inserted. The other code decomposes $[3, 8] = [3, 4] \cup [4, 8]$ and makes the following calls.

```
Toggle(7)
Toggle(12)
Update(6, 5, 7)
Update(4, 2, 6)
Update(8, 4, 12)
```

At this point, `Report`() is 4, since the intervals covered an odd number of times are $[1, 3]$ and $[6, 8]$. We finish by removing $[1, 6]$ with the same sequence of calls that inserted it. Now, `Report`() $= 4$.