

# 15-750 — Graduate Algorithms — Spring 2006

Miller and Derryberry

Assignment 2 Due date: Friday, February 17, 2006.

## Some Reminders:

- Read the Policies section on the course website before you start working on this assignment.
- You may work in **groups of size up to 3** for this problem set if you wish. However, **you should write up your solutions separately**. That is, collaboration should be limited to talking about the problems, so that your writeup is written entirely by you and not copied from your partner. In addition, state whether you worked alone and **list all collaborators**.
- Please refrain from consulting external materials when solving these problems.
- Please **submit both an electronic version, as well as a hard copy of your solutions** at the beginning of class on the due date.
- In all problems, it is implicit that you should show that your answer is correct, even when this is not explicitly stated.
- If you have questions, contact the course staff.

## 1 Splay Trees

- (a) (5 points) When we studied self-adjusting binary search trees, we saw that the rotate-to-root algorithm can be viewed as a treap in which each access moves the accessed element to the front of the priority list.

Show how to translate the splay algorithm to this “treap view” of BSTs. That is, give a priority list update scheme that is consistent with splaying, but does not have separate cases for left-rotations and right-rotations or zig-zig splay steps versus zig-zag steps.

---

**SOLUTION:** First, take the subset of the priority list corresponding to the access path and move it to the front of the list. Note that this reordering does not change the BST because the access path retains its original ordering and no node on the access path moves in front of an ancestor.

Next, repeatedly swap the accessed element  $x$  with its grandparent until  $x$  has no grandparent. It is easily verified that each of these swaps corresponds to a splay step. Finally, if  $x$  is a child of the root, swap  $x$  with the root in the priority list.

Note that the first step we took, moving the access path to the front of the list, was necessary to ensure that the access path remained connected in the BST when the grandparent swaps were executed.

- (b) (10 points) The Access Lemma for splay trees can be used to prove many interesting theorems about splay trees. In this problem, the goal is to come up with a weight assignment  $w : \{1, \dots, n\} \mapsto \mathbb{R}_{>0}$ , that proves the following statement: for a fixed target  $t \in \{1, \dots, n\}$ , the cost of a sequence of accesses  $\sigma = \sigma_1 \sigma_2 \cdots \sigma_m$  in a splay tree storing the keys  $\{1, \dots, n\}$  is  $O(m + n \log n + \sum_{i=1}^m \log(1 + |t - \sigma_i|))$ . Notice that if many of the accesses are very close to  $t$ , this statement proves that the average cost per access could be much less than  $\log n$ . Also, notice that the choice of  $t$  is independent of the splay algorithm, so we can choose any  $t$  we want in hindsight.

---

**SOLUTION:** Note that this theorem is actually called the *Static Finger Theorem*.

The access lemma states that, using a potential of  $\log(s(x))$  for each node, the amortized cost of each access  $\sigma_i$  is  $1 + O(\log(\frac{s(r)}{s(\sigma_i)}))$ , where  $r$  is the root of the tree,  $s(x) = \sum_{y \in T_x} w(y)$ , and  $T_x$  is the subtree rooted at  $x$ . Hence, the actual cost of the sequence of accesses  $\sigma$  is

$$O\left(m + \Phi_0 - \Phi_m + \sum_{i=1}^m \log\left(\frac{s(r)}{s(\sigma_i)}\right)\right),$$

where  $\Phi_0$  is the initial potential of the tree and  $\Phi_m$  is the final potential.

Using a weight function of  $w(x) = (1 + |t - x|)^{-2}$ , we have  $W = \sum_{i=1}^n w(x) \leq 2 \sum_{i=1}^{\infty} i^{-2} \leq 4$  so that

$$-2n \log n \leq \sum_{i=1}^n \log(n^{-2}) \leq \sum_{i=1}^n \log(w(i)) \leq \Phi_0, \Phi_m \leq \sum_{i=1}^n \log W \leq 2n.$$

Thus,  $\Phi_0 - \Phi_m$  is at most  $O(n \log n)$ .

Finally,

$$\log\left(\frac{s(r)}{s(\sigma_i)}\right) \leq \log\left(\frac{W}{w(\sigma_i)}\right) \leq \log\left(\frac{2}{w(\sigma_i)}\right) = \log 4 + 2 \log(1 + |t - \sigma_i|),$$

as suffices to prove the theorem.

- (c) (Extra Credit) Prove (or give a counter-example) that the cost of a sequence of accesses  $\sigma = \sigma_1 \sigma_2 \cdots \sigma_m$  in a splay tree is  $O(m + n \log n + \sum_{i=1}^m \min_{1 \leq j \leq n} \log(j + |t_j - \sigma_i|))$  where  $(t_1, \dots, t_n)$  is an arbitrary permutation of the keys  $\{1, \dots, n\}$ . Partial extra credit will be given for proving  $O(m + n \log n + \sum_{i=1}^m \min_{1 \leq j \leq n} (j + \log(1 + |t_j - \sigma_i|)))$ .
-

**SOLUTION:** Let  $w(x) = \sum_{j=1}^n (j + |t_j - x|)^{-3}$ . Notice that

$$\begin{aligned}
 W &= \sum_{i=1}^n w(i) \\
 &= \sum_{i=1}^n \sum_{j=1}^n (j + |t_j - i|)^{-3} \\
 &= \sum_{j=1}^n \sum_{i=1}^n (j + |t_j - i|)^{-3} \\
 &\leq \sum_{j=1}^n \sum_{k=0}^{\infty} 2(j+k)^{-3} \\
 &\leq \sum_{j=1}^{\infty} 2(j^{-3})j = \sum_{j=1}^{\infty} 2j^{-2} \leq 4.
 \end{aligned}$$

Moreover, the minimum potential at a node  $x$  is  $w(x) \geq -3 \log n$  and the maximum potential at a node is at most  $\log W$ . Thus,  $\Phi_0 - \Phi_m$  is  $O(n \log n)$ , and

$$\begin{aligned}
 \log \left( \frac{W}{w(\sigma_i)} \right) &\leq \log \left( \frac{4}{\sum_{j=1}^n (j + |t_j - \sigma_i|)^{-3}} \right) \\
 &\leq \log \left( \frac{4}{\max_{1 \leq j \leq n} (j + |t_j - \sigma_i|)^{-3}} \right) \\
 &\leq \log 4 + 3 \log \left( \min_{1 \leq j \leq n} (j + |t_j - \sigma_i|) \right),
 \end{aligned}$$

as suffices to prove the theorem.

## 2 Constructing a triangulation

(15 points) Give an  $O(n \log n)$  time algorithm that constructs a triangulation of  $n$  points in the plane. The triangulation should include all segments of the convex hull.

**SOLUTION:** We can use the following sweepline algorithm, which uses a doubly-linked circular list to store the convex hull of points  $p_1, \dots, p_i$ , for  $i = 1, \dots, n$ , in counter-clockwise order.

TRIANGULATE( $p = p_1, \dots, p_n$ )

- 1 sort the  $p$  by  $x$ -coordinate
- 2 let  $E$  be an empty list
- 3 let  $H$  be an empty doubly-linked circular list
- 4 insert  $p_1, p_2$ , and  $p_3$  into  $H$  so they are ordered counter-clockwise
- 5 add the 3 edges of this triangle into  $E$
- 6 **for**  $i \leftarrow 4$  **to**  $n$
- 7 **do**  $p_u \leftarrow p_{i-1}$
- 8 **while** the successor of  $p_u$  in  $H$  is right of the ray  $(p_i, p_u)$

```

9   do if  $p_u \neq p_{i-1}$ 
10      then add the edge  $(p_i, p_u)$  to  $E$ 
11         delete  $p_u$  from  $H$ 
12       $p_u \leftarrow$  the successor of  $p_u$ 
13       $p_d \leftarrow p_{i-1}$ 
14      while the predecessor of  $p_d$  in  $H$  is left of the ray  $(p_i, p_d)$ 
15      do if  $p_u \neq p_{i-1} \vee p_d \neq p_{i-1}$ 
16         then add the edge  $(p_i, p_d)$  to  $E$ 
17            delete  $p_d$  from  $H$ 
18             $p_d \leftarrow$  the predecessor of  $p_d$ 
19      insert  $p_i$  into  $H$  between  $p_u$  and  $p_d$ 
20 insert  $(p_n, p_u)$  and  $(p_n, p_d)$  into  $H$ 
21 return  $E$ 

```

Essentially, this algorithm just adds the points one at a time to the triangulation, each time “zipping” the new vertex  $p_i$  into the triangulation by adding edges between it and every visible member of the convex hull, and it deletes the newly covered vertices from the current convex hull  $H$ . After performing an  $O(n \log n)$  time sorting step, the remaining work is only  $O(n)$  because each during each iteration all but 2 of the nodes of the convex hull that are touched are covered up, never to be seen again (so we just add a credit to each node when we insert it into  $H$  to pay for when it is covered up later). Thus, this algorithm runs in  $O(n \log n)$  time.

### 3 Bridging a Set of Points

(15 points) Let  $P$  be a set of  $n$  points in the plane. The Upper Hull of  $P$  is the simple path of edges on the convex hull of  $P$  bounding  $P$  from above.

We say that the edge  $E$  is the **bridge above**  $x = c$  if  $E$  is the edge on the upper hull which intersects the line  $x = c$ . You may assume that no point in  $P$  is on the line  $x = c$ .

Give an  $O(n)$  expected-time algorithm for finding the bridge above  $x = c$ . (Hint: think about how to phrase this problem in terms of a 2-d linear program.)

---

**SOLUTION:** Recall that we can solve 2-dimensional linear programs optimally in expected time linear in the size of the linear program. Thus, it suffices to give a linear time (and, hence, linear size) reduction from the bridge problem to linear programming with 2 variables. Let  $P = \{p_1, \dots, p_n\}$  where  $p_i = (x_i, y_i)$ . Our strategy will be to find the infinite line that rests on the bridge. This can be accomplished by observing that if the line is of the form  $y = ax + b$ , then it must be the case that  $ax_i + b \geq y_i$  for all  $i$ . Further, to force the line to actually rest on the bridge, we minimize the height of the line at  $x = c$ , subject to the previous constraints. That is, we minimize  $ac + b$ . It is straightforward that this linear program is a valid reduction from the bridge problem, and note that its size is linear in  $n$  and it uses only two variables,  $a$  and  $b$ .

### 4 The Diameter of a Set of Points

(15 points) Suppose we are given a set  $P = \{p_1, \dots, p_n\}$  of points in the plane. Further assume that the convex hull of  $P$  is the points  $(p_1, \dots, p_n)$ . Give a linear time algorithm for finding the

pair of points in  $P$  with largest Euclidean distance between them.

Hint: First show how to compute for each pair of points  $(p_i, p_{i+1})$  the furthest point in  $P$  to the infinite line through  $p_i$  and  $p_{i+1}$ . You should include the pair  $(p_n, p_1)$ .

Make sure to include a proof of correctness.

**SOLUTION:** In the following solution, all subscript arithmetic is done modulo  $n$ , shifted one off from 0 (i.e.,  $p_{1-1} = p_n$  and  $p_{n+1} = p_1$ ). Assume  $p_1, \dots, p_n$  occur in clockwise order around the convex hull. First, to compute for each pair of points  $(p_i, p_{i+1})$  the furthest point in  $P$  to the infinite line through  $p_i$  and  $p_{i+1}$ , we start with  $i = 1$  and use brute force to find the farthest point from the line through  $p_1$  and  $p_2$ . Then, for each successive value of  $i$  we scan clockwise while the distance is increasing until we reach a locally maximal point  $p_{i^*}$  in terms of distance from the infinite line through  $p_i$  and  $p_{i+1}$ . Each locally maximal  $p_{i^*}$  is also the globally maximal point in terms of distance from the infinite line through  $p_i$  and  $p_{i+1}$  because of the convexity of the hull (this is obvious, but a little painful to properly prove, so it is left as an exercise). To see that this part of the algorithm runs in  $O(n)$  time, note that  $p_{i^*}$  will never “lap”  $p_i$  because the point  $p_i$  is closer (at distance 0) to the line through  $p_i$  and  $p_{i+1}$  than  $p_{i-1}$ .

Next, we claim that any pair of most distant points in  $P$  are of the form  $\{p_i, p_{i^*}\}$  or  $\{p_{i+1}, p_{i^*}\}$  for some  $i$ . To see this, let  $\{p_a, p_b\}$  be such a pair of most distant points (for simplicity, you can think of this pair as being unique). If we draw the two lines  $\ell_a$  and  $\ell_b$  perpendicular to the chord  $\{p_a, p_b\}$ ,  $\ell_a$  through  $p_a$  and  $\ell_b$  through  $p_b$ , note that all points other than  $p_a$  and  $p_b$  are strictly between  $\ell_a$  and  $\ell_b$ . Let  $\theta_{c,j}$  represent the positive acute angle between  $p_j$  and  $\ell_c$  (for  $c \in \{a, b\}$ ). Let  $\theta_{c^*,j^*}$  be the minimum of  $\theta_{a,a-1}$ ,  $\theta_{a,a+1}$ ,  $\theta_{b,b-1}$ , and  $\theta_{b,b+1}$ . Note that if we rotate  $\ell_{c^*}$  so that it is flush with the edge  $\{p_{c^*}, p_{j^*}\}$  and keep the other line  $\ell_{\bar{c}^*}$  parallel to  $\ell_{c^*}$ , on the opposite side of the hull, and disjoint from the interior of the hull, it will still be intersecting  $p_{\bar{c}^*}$ , which, hence, is the farthest point from the infinite line through the points  $p_{c^*}$  and  $p_{j^*}$ . Thus, checking the distances between the  $O(n)$  pairs of points of the form  $\{p_i, p_{i^*}\}$  and  $\{p_{i+1}, p_{i^*}\}$  suffices to find the pair of most distant points.