Algorithms CSc545 — homework #4
Due 11/11/02

November 18, 2002

1. Suggest a data structure that supports the following operations on a set of intervals on the real numbers.

   - `Insert`(x, y) — Insert the interval (x, y) (where $x < y$) into the structure.
   - `Delete`(x, y) — Delete the interval (x, y) (where $x < y$) from the structure, if it was inserted beforehand. You may assume that no two intervals share the same endpoint in the structure.
   - `Report`(x) — Report how many intervals in the structure contain x ?

     The time for each operation should be $O(\log n)$, where n is the number of intervals.

   Answer: *We create a balanced search tree T that stores all endpoints of intervals, sorted by their x-coordination. With each node μ of T we store two fields L_μ and R_μ, where L_μ (resp. R_μ) which is the number of left (resp. endpoints) of endpoints of intervals stored in the subtree whose root is μ. Then we can find for a query value x the number of left (resp. right) endpoint of intervals to the left (resp. right) of x. The different between these numbers is the number of intervals containing x.*

2. Let S be a set of numbers. An partition of S into k-subsets $S_1\dot{S}_k$ is called *ordered splitting* if each element of S appears in exactly one subset, $S = \cup_i S_i$, each S_i contains $\leq \lceil |S|/k \rceil$ elements, and each element of S_i is smaller than each element of S_{i+1}, for $i = 1 \ldots k - 1$.

   Let $\varepsilon > 0$ be a very tiny number, and let S be a set of n numbers. We wish to find an order splitting of S into n^ε subsets, (each contains $\lceil n^{1-\varepsilon} \rceil$. For example, if $n = 2^{20}$ and $\varepsilon = 1/10$ then each subset contains $2^{20\cdot(1-1/10)} = 2^{18}$, and we divide S into $2^{20\cdot 1/10} = 4$ subsets.

   (a) Suggest an algorithm for this problem whose running time is $O(n \log n)$.

Answer: *We sort the elements of S (in $O(n \log n)$ time), and pick the appropriate numbers.*

(b) Suggest an algorithm whose running time is $O(n)$, when we want an order splitting of S into 3 subsets.

Answer: *We apply twice the algorithm for picking the k'st largest element in S, where $k = \lceil n/3 \rceil$ and $k = \lceil 2k/3 \rceil$.*

(c) Suggest an algorithm whose running time is $O(n)$ when $\varepsilon = 1/\log n$, for finding an ordered splitting of S into n^{ε} subsets.

Answer: *We find the median of of S, split S into two parts, divide each region into two parts of equal size (so each part is $n/4$ and repeat until each part is of size $n/2^k \leq n^{1-\varepsilon}$. The running time of each iteration is $O(n)$, since the total number of elements moved in each iteration by the partition algorithm is $O(n)$. The number of iteration is $O(\log n^{\varepsilon} = \varepsilon \log n)$, and since $\varepsilon = 1 \log n$, this time is $O(1)$.*

Finally we need to find in each part of the array the k'th largest element, according to the values of $n^{1-\varepsilon}$.

3. We discussed in class and in a homework a method for transforming a static data structure for a set S of elements into a semi-dynamic one (when insertions are allowed). In the case of the sorted array proplem, the dynamization was obtained by storing the elements of S in $\leq \log_2 |S|$ sorted arrays of sizes $1, 2, 2^2, \ldots 2^{\lfloor \log_2 n \rfloor}$, where at most one array of each size was stored. We showed an analogy between the way we maintain the arrays as points are added to S, and the method a binary counter is updated as we increase the values it stores.

   The purpose of this question is to study what happens if we handle the arrays analogous to a decimal counter. That is, we allow at most 9 arrays of size 1, at most 9 arrays of size 10, at most 9 of size 100 etc. Once 10 arrays of size 10^k are created (for some integer k), we merge them into an array of size 10^{k+1}.

   (a) What is the amortized running time for inserting an element? What is the time for finding an element in the structure?

   (b) What is the answer to the previous question if we use base b ? What if $b = n^{\alpha}$ for some small α?

   (c) What is you answer for the first question, if the time to create an array for a set of n elements is $\Theta(n^2)$?

4. Suggest how to modify the data structure Union/Find studied in class, so that in addition to the "standard' operations it supports (`Link` of two sets, `find` the root of an element and create a new set), it would also support the operation `Historical_Find`(x, y, t), where x, y are elements in the structure and t is some time in the past. This operation should answer whether at time t

the elements x and y were in the same set. The amortized time for `Link` and `Find_set` operations is $\alpha(n)$, and the amortized time for `Historical_Find` is $O(\alpha(m) \log\log n)$. Here n is the number of elements and m is the number of operations. Hint — maintain for each element x the parent of x at each time.

Answer: *Each element x in the structure stores all the points to elements it used to point to in different stages of the algorithm, including the current one. These elements are sorted in sorted array by their time of update. Note that there are only $O(\log n)$ of them. When x is updated to a new element y (as a result to a path compression or link operation) y is added at the end of the array. This operation takes $O(1)$ time for an element, so it does not increase the asymptotic running time.*

To answer `Historical_Find`(x, y, t) *operation, we trace elements from x and y, based on the element pointed at time t. To find this pointers, we perform binary search in each array. Since the size of the array is $O(\log n)$, the time for a finding the pointer of a single element is $O(\log\log n)$.*

5. Let S be a set of vertical segments in the plane. The distance between two segments is defined as the closest distance between two points on the segments. Describe an $O(n \log n)$-time algorithm for finding the closest pair of segments. You can assume that no two endpoints of different segments have the same x or y coordination.

   Answer: *The algorithm is identical to the divide and conquer algorithm for finding the closest pair of segments. We only need to show how to find in $O(n)$ time the distance between the closest pair of segments, one to the left of a vertical line ℓ, one to the right, and both lie in a vertical strip of width 2δ (the notation is as in the algorithm for closest pair of points). This is true since a rectangle sliding along the strip can intersect $O(1)$ segments at any fixed location. Indeed, at most one segment intersect the upper or lower edge of the sliding rectangle, and the rectangle can contain at most 4 endpoints of segments.*

   assume that more than 4 segment to the left of ℓ intersect the rectangle, and let p be the distance bet

6. CLRS 15-4.5