CS 466/666, Spring 2006
Timothy Chan

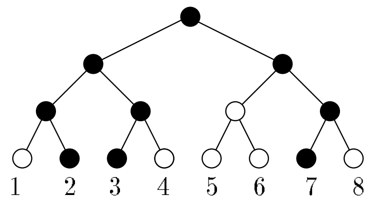
Assignment 2 (due June 8 Thursday noon)

Please read `http://www.student.cs.uwaterloo.ca/~cs466/policies.html` first for general instructions.

1. [*18 marks*] Consider the problem of maintaining a set of elements to support two operations: (i) insert a new element to S, and (ii) search for the successor and the predecessor of an element $x \in S$ (i.e., the next largest and the next smallest element in the set S). Ordinarily, the problem can be solved in $O(\log n)$ time per operation by balanced search trees. However, in this question, we will investigate a special case, where the elements are all small *integers*, in the range $\{1, 2, \dots, n\}$.

   In this special case, give a method with $O(1)$ worst-case search time and $O(1)$ amortized insertion time, i.e., a sequence of n insertions to $S \subseteq \{1, 2, \dots, n\}$ (starting with $S = \emptyset$) should take $O(n)$ total time. Write pseudocode and provide an amortized analysis.

   For your data structure, use a (static) complete binary tree with n leaves corresponding to the integers $1, 2, \dots, n$. Mark a leaf if it corresponds to an element in S. Mark an internal node z if there exists a leaf underneath z that is in S. In addition, maintain a doubly linked list containing the elements in S in order (to allow successors and predecessors later be found in constant time). To insert x, describe how to update the marks and how to find its successor or predecessor, by traversing up from the leaf x until a marked node is reached, and then traversing down to the successor or predecessor. For the analysis, use a potential method or a charging argument.

   $S = \{2, 3, 7\}$

   (marked nodes are colored black)

2. [*6 marks*] Consider the following data structure problem: maintain a set S of points in 2D under insertions and deletions, so that we can decide whether a given point $p \in S$ is *maximal*. Here, a point p is said to be maximal if every point $q \in S$ satisfies $q.x \leq p.x$ *or* $q.y \leq p.y$. Show how to obtain query and amortized update time bounds that are polylogarithmic (i.e., $O(\log^c n)$ for some constant c). (You may quote results from class.)

3. [*18 marks*] The objective of this question is to obtain a data structure for the union-find problem with good worst-case bounds (instead of amortized). The *tree method* from class with the weighted union heuristic (without path compression) already achieves $O(1)$ worst-case time for union and $O(\log n)$ worst-case time for find. We will consider a modification of this method that speeds up find.

Fix a parameter k. Let $p[x]$ denote the parent of x and size(x) denote the size of the subtree rooted at x. The idea is to do the linkings so that we always have size$(p[x]) \geq k\,$size(x) (instead of merely doubling). To accomplish this, we need to represent each set as a list of up to k^2 trees (instead of a single tree). For each root, we store the label of the set explicitly. Here is the pseudocode for union and find:

find(x):
1. if x is a root then return the label stored at x
2. else return find$(p[x])$

union(A, B):
1. move B's trees to A's list and change the label of B's roots to A
3. if A now has more than k^2 trees {
4. let $v_1, \ldots, v_\ell$ be the roots of these trees ($\ell > k^2$)
5. let $n_i = $ size(v_i) and sort so that $n_1 \leq \cdots \leq n_\ell$
6. find an i with $\ell - k^2 \leq i < \ell$ so that $n_1 + \cdots + n_i + n_\ell \geq kn_i$
7. link $v_1, \ldots, v_i$ to v_ℓ, i.e., set $p[v_1], \ldots, p[v_i] = v_\ell$

}

(a) [*5 marks*] Math problem: prove that i indeed exists in line 6 (assuming $\ell > k^2$). [Hint: first show that there exists a j with $\ell - k(k-1) \leq j \leq \ell$ such that $n_j - n_{j-k} \leq n_\ell/k$, e.g., via a proof by contradiction. Then show that $n_j - n_{j-k} \leq n_\ell/k$ implies $n_1 + \cdots + n_{j-1} + n_\ell \geq kn_{j-1}$.]

(b) [*5 marks*] Show that union takes $O(k^2 \log k)$ worst-case time (or better).

(c) [*6 marks*] Show that find takes $O(\log_k n)$ worst-case time. [Hint: confirm that size$(p[x]) \geq k\,$size(x) for all x.]

(d) [*2 marks*] More math: choose k so that both union and find take at most $O(\log n/\log\log n)$ time.

4. [*8 marks*] In this question, we revisit the problem of computing an approximate median (the static version this time, without updates). For our purposes, an *approximate median* is defined as an element of rank between $0.25n$ and $0.75n$.

There is a trivial Monte-Carlo algorithm that runs in $O(1)$ (constant!) time: just return a random element. Unfortunately, the error probability is 1/2.

Analyze the following similar $O(1)$-time Monte-Carlo algorithm: pick 3 random elements r_1, r_2, r_3 (uniformly and independently) and return the median of r_1, r_2, r_3. Show that the error probability of this algorithm is 0.3125.

[Bonus (3 marks): if we modify the strategy to return the median of r random elements, what is the smallest (odd) value of r to guarantee an error probability below 0.01?]