

CS 473G: Graduate Algorithms, Spring 2007

Homework 5

Due Thursday, April 17, 2007

Please remember to submit **separate, individually stapled** solutions to each problem.

Unless a problem specifically states otherwise, you can assume the function $\text{RANDOM}(k)$, which returns an integer chosen independently and uniformly at random from the set $\{1, 2, \dots, k\}$, in $O(1)$ time. For example, to perform a fair coin flip, you would call $\text{RANDOM}(2)$.

1. Suppose we want to write an efficient function $\text{RANDOMPERMUTATION}(n)$ that returns a permutation of the integers $\langle 1, \dots, n \rangle$ chosen uniformly at random.

- (a) What is the expected running time of the following RANDOMPERMUTATION algorithm?

```
RANDOMPERMUTATION(n):
  for i ← 1 to n
    π[i] ← EMPTY
  for i ← 1 to n
    j ← RANDOM(n)
    while (π[j] ≠ EMPTY)
      j ← RANDOM(n)
    π[j] ← i
  return π
```

- (b) Consider the following partial implementation of RANDOMPERMUTATION .

```
RANDOMPERMUTATION(n):
  for i ← 1 to n
    A[i] ← RANDOM(n)
  π ← SOMEFUNCTION(A)
  return π
```

Prove that if the subroutine SOMEFUNCTION is deterministic, then this algorithm cannot be correct. [Hint: There is a one-line proof.]

- (c) Describe and analyze an RANDOMPERMUTATION algorithm whose expected worst-case running time is $O(n)$.
- * (d) **[Extra Credit]** Describe and analyze an RANDOMPERMUTATION algorithm that uses only fair coin flips; that is, your algorithm can't call $\text{RANDOM}(k)$ with $k > 2$. Your algorithm should run in $O(n \log n)$ time with high probability.

2. A *meldable priority queue* stores a set of keys from some totally-ordered universe (such as the integers) and supports the following operations:

- MAKEQUEUE: Return a new priority queue containing the empty set.
- FINDMIN(Q): Return the smallest element of Q (if any).
- DELETEMIN(Q): Remove the smallest element in Q (if any).
- INSERT(Q, x): Insert element x into Q , if it is not already there.
- DECREASEKEY(Q, x, y): Replace an element $x \in Q$ with a smaller element y . (If $y > x$, the operation fails.) The input is a pointer directly to the node in Q that contains x .
- DELETE(Q, x): Delete the element $x \in Q$. The input is a pointer directly to the node in Q that contains x .
- MELD(Q_1, Q_2): Return a new priority queue containing all the elements of Q_1 and Q_2 ; this operation destroys Q_1 and Q_2 .

A simple way to implement such a data structure is to use a heap-ordered binary tree, where each node stores a key, along with pointers to its parent and two children. MELD can be implemented using the following randomized algorithm:

```

MELD( $Q_1, Q_2$ ):
  if  $Q_1$  is empty, return  $Q_2$ 
  if  $Q_2$  is empty, return  $Q_1$ 
  if  $key(Q_1) > key(Q_2)$ 
    swap  $Q_1 \leftrightarrow Q_2$ 
  with probability 1/2
     $left(Q_1) \leftarrow MELD(left(Q_1), Q_2)$ 
  else
     $right(Q_1) \leftarrow MELD(right(Q_1), Q_2)$ 
  return  $Q_1$ 

```

- (a) Prove that for *any* heap-ordered binary trees Q_1 and Q_2 (not just those constructed by the operations listed above), the expected running time of MELD(Q_1, Q_2) is $O(\log n)$, where $n = |Q_1| + |Q_2|$. [Hint: How long is a random root-to-leaf path in an n -node binary tree if each left/right choice is made with equal probability?]
- (b) Prove that MELD(Q_1, Q_2) runs in $O(\log n)$ time with high probability.
- (c) Show that each of the other meldable priority queue operations can be implemented with at most one call to MELD and $O(1)$ additional time. (This implies that every operation takes $O(\log n)$ time with high probability.)

3. Prove that GUESSMINCUT returns the *second* smallest cut in its input graph with probability $\Omega(1/n^2)$. (The second smallest cut could be significantly larger than the minimum cut.)

4. A *heater* is a sort of dual treap, in which the priorities of the nodes are given by the user, but their search keys are random (specifically, independently and uniformly distributed in the unit interval $[0, 1]$).
- Prove that for any r , the node with the r th smallest *priority* has expected depth $O(\log r)$.
 - Prove that an n -node heater has depth $O(\log n)$ with high probability.
 - Describe algorithms to perform the operations INSERT and DELETEMIN in a heater. What are the expected worst-case running times of your algorithms?

You may assume all priorities and keys are distinct. [Hint: Cite the relevant parts (but only the relevant parts!) of the treap analysis instead of repeating them.]

5. Let n be an arbitrary positive integer. Describe a set \mathcal{T} of binary search trees with the following properties:
- Every tree in \mathcal{T} has n nodes, which store the search keys $1, 2, 3, \dots, n$.
 - For any integer k , if we choose a tree uniformly at random from \mathcal{T} , the expected depth of node k in that tree is $O(\log n)$.
 - Every tree in \mathcal{T} has depth $\Omega(\sqrt{n})$.

(This is why we had to prove via Chernoff bounds that the maximum depth of an n -node treap is $O(\log n)$ with high probability.)

- ★6. [Extra Credit] Recall that F_k denotes the k th Fibonacci number: $F_0 = 0$, $F_1 = 1$, and $F_k = F_{k-1} + F_{k-2}$ for all $k \geq 2$. Suppose we are building a hash table of size $m = F_k$ using the hash function

$$h(x) = (F_{k-1} \cdot x) \bmod F_k$$

Prove that if the consecutive integers $0, 1, 2, \dots, F_k - 1$ are inserted in order into an initially empty table, each integer is hashed into one of the largest contiguous empty intervals in the table. Among other things, this implies that there are no collisions.

For example, when $m = 13$, the hash table is filled as follows.

0												
0								1				
0			2					1				
0			2					1			3	
0			2			4		1			3	
0	5		2			4		1			3	
0	5		2			4		1	6		3	
0	5		2	7		4		1	6		3	
0	5		2	7		4		1	6		3	8
0	5		2	7		4	9	1	6		3	8
0	5	10	2	7		4	9	1	6		3	8
0	5	10	2	7		4	9	1	6	11	3	8
0	5	10	2	7	12	4	9	1	6	11	3	8