CSC C78F 1999

Assignment 4

due: Wednesday, November 24, 1999 at 1:10pm

1. Consider the following three data structures for implementing a PRIORITY QUEUE for elements from the universe $U = \{0, \ldots, 2^k - 1\}$. The priority of an element is its value.

   For any element $x \in U$, let PREDECESSOR(x) denote the largest element in the priority queue smaller than x, if such an element exists; otherwise PREDECESSOR(x) $= -1$. Note that x can be *any* element in U, and need not be in the priority queue.

   - A bit vector $A[0..2^k - 1]$ of length 2^k augmented with a variable m containing the maximum element in the priority queue.
     Here $A[i] = 1$ if and only if i is in the priority queue, for $0 \leq i < 2^k$.
   - An array $B[1..2^{k+1} - 1]$ of $2^{k+1} - 1$ bits implicitly representing (as in a heap) a tree with 2^k leaves. Here $B[2^k + i] = 1$ if and only if i is in the priority queue, for $0 \leq i < 2^k$. For $1 \leq j < 2^k$, $B[j] = 1$ if and only if the subtree rooted at node j contains a leaf with value 1.
   - An array $C[1..2^{k+1} - 1]$ of elements in $U \cup \{-1\}$ implicitly representing (as in a heap) a tree with 2^k leaves and an array $C'[1..2^k - 1]$ of elements in $U \cup \{-1\}$.
     For $0 \leq i < 2^k$,

     $$C[2^k + i] = \begin{cases} -1 & \text{if } i \text{ is not in the priority queue} \\ 2^k - 1 & \text{if } i \text{ is the minimum element in the priority queue} \\ \text{PREDECESSOR}(i) & \text{if } i \text{ is in the priority queue, but is not the minimum element} \end{cases}$$

     and, for $1 \leq i \leq 2^k - 1$,

     $$C[i] = \begin{cases} j & \text{if } j \text{ is the largest element in the priority queue such that} \\ & 2^k + j \text{ is a leaf of the subtree rooted at node } i \\ -1 & \text{if } 2^k + j \text{ is a leaf of the subtree rooted at node } i \\ & \text{for no element } j \text{ in the priority queue} \end{cases}$$

     $$\text{and } C'[i] = \begin{cases} j & \text{if } j \text{ is the smallest element in the priority queue such that} \\ & 2^k + j \text{ is a leaf of the subtree rooted at node } i \\ -1 & \text{if } 2^k + j \text{ is a leaf of the subtree rooted at node } i \\ & \text{for no element } j \text{ in the priority queue.} \end{cases}$$

   (a) For each of the data structures, explain how to perform the operations INSERT, MAXIMUM, and EXTRACT-MAX (in an efficient manner). Justify why your algorithms are correct.
   Give the worst case complexity of each algorithm.

   (b) Explain how to efficiently perform the operation PREDECESSOR(x) in each of the first two data structures. Justify why your algorithms are correct.
   Give the worst case complexity of your algorithms.

(c) Explain how to perform the operation PREDECESSOR(x) in the third data structure in $O(\log k)$ time. Justify why your algorithm is correct and runs in the required time.
(Hint: with each leaf 2^k+i, associate the k-bit string denoting the binary representation of i. Also, with each non-leaf node of depth d, associate the d-bit binary string which is the common d-bit prefix of the strings associated with all leaves in its subtree. Say a node i is *empty* if $C[i] = -1$. Perform binary search to find the longest prefix of x which is the binary representation of some non-empty node in the tree.)

(d) Describe a static data structure that stores a set of n elements from U using $O(nk)$ space and supports PREDECESSOR operations in $O(\log k)$ time. Justify why your data structure uses the required amount of space.
Explain how PREDECESSOR is performed, why your algorithm is correct, and why it runs in the required time.

(e) BONUS Describe a static data structure that stores a set of n elements from U using $O(n)$ space and supports PREDECESSOR operations in $O(\log k)$ time. Justify why your data structure uses the required amount of space.
Explain how PREDECESSOR is performed, why your algorithm is correct, and why it runs in the required time.

2. A d-ary heap is like the binary heaps we have studied so far except that each node can have up to d children.

(a) How would you represent a d-ary heap as an array? Give formulas for the *left_child*(i), *right_child*(i) and *parent*(i).

(b) What is the height of a d-ary heap of n elements?

(c) Give an efficient implementation of INSERT. What is its running time in terms of d and n? (Use Θ notation.)

(d) Give an efficient implementation of EXTRACT-MAX. What is its running time in terms of d and n? (Use Θ notation.)

3. Suppose that $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ and $M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ are two deterministic finite automata with disjoint state sets. Let $Q = Q_1 \cup Q_2$, let $F = F_1 \cup F_2$, and let $\delta : Q \times \Sigma \rightarrow Q$ denote the function

$$\delta(q,a) = \begin{cases} \delta_1(q_1,a) & \text{if } q \in Q_1 \\ \delta_2(q_2,a) & \text{if } q \in Q_2. \end{cases}$$

Two states $q, q' \in Q$ are *equivalent* if, for all $x \in \Sigma^*$,

$$\delta(q,x) \in F \text{ if and only if } \delta(q',x) \in F.$$

Note that q and q' are equivalent states if and only if, for all $a \in \Sigma$, the states $\delta(q,a)$ and $\delta(q',a)$ are equivalent, and either $q, q' \in F$ or $q, q' \in Q - F$. (This can be proved by induction, but that is not part of the assignment,)

(a) Explain how to use the DISJOINT SETS abstract data type to test whether two deterministic finite automata accept exactly the same set of strings. Justify the correctness of your algorithm.

(b) What data structure should you use to implement the DISJOINT SETS abstract data type for this application? Justify your answer.