

**CS 265**  
**Enriched Data Structures**  
**ASSIGNMENT # 2**  
**DUE DATE: October 27, 2011**

1. Give an algorithm for the following problem. The input is a sequence of  $n$  numbers  $\{x_1, x_2, \dots, x_n\}$ , another sequence of  $n$  numbers  $\{y_1, y_2, \dots, y_n\}$ , and a number  $z$ . Your algorithm should determine whether or not  $z \in \{x_i + y_j \mid 1 \leq i, j \leq n\}$ . You should use universal hashing families, and your algorithm should run in expected time  $O(n)$ .

Provide justification that your algorithm is correct and runs in the required time. Be very clear about which theorems from class and/or the text you are using, and how.

2. Define a family  $\mathcal{H}$  of hash functions from a finite set  $U$  to a finite set  $B$  to be  $\epsilon$ -universal if for all pairs of distinct elements  $k$  and  $l$  in  $U$ ,  $Pr\{h(k) = h(l)\} \leq \epsilon$ , where the probability is taken over the drawing of hash function  $h$  at random from the family  $\mathcal{H}$ . Show that an  $\epsilon$ -universal family of hash functions must have  $\epsilon \geq \frac{1}{|B|} - \frac{1}{|U|}$ .

*Hint:* Try to count (in two different ways) the total number (or probability) of collisions, over all pairs  $k, l$  and over all hash functions  $h \in \mathcal{H}$ . The identity  $a^2 = a + 2\binom{a}{2}$  may come in handy.

3. We can implement a queue  $Q$  using two stacks  $H$  and  $T$  as follows: Think of the stack  $T$  as containing the “tail” of the queue (i.e., the recently inserted items), with the most recently inserted item at the top. Think of the stack  $H$  as containing the “head” of the queue (i.e., the older items), with the oldest item of the queue at the top. Conceptually,  $Q$  consists of  $T$  and  $H$  placed “back-to-back”.

To ENQUEUE an item  $x$  on  $Q$ , we actually PUSH  $x$  into  $T$ . To DEQUEUE an item, we POP  $H$ , provided  $H$  is not empty. If  $H$  is empty, we transfer the items of  $T$  to  $H$  (by popping each item of  $T$  and then pushing it into  $H$ ), and then POP  $H$ .

These algorithms are given below in pseudo-code. (We assume that initially the stacks  $H$  and  $T$  are empty, and that the function STACKEMPTY( $T$ ) returns **true** if  $T$  is an empty stack and **false** otherwise.)

```
ENQUEUE( $Q, x$ )          DEQUEUE( $Q$ )
  PUSH( $T, x$ )           if STACKEMPTY( $H$ ) then
                        loop
                          exit when STACKEMPTY( $T$ )
                           $x :=$  POP( $T$ )
                          PUSH( $H, x$ )
                        end loop
                        end if
                        return POP( $H$ )
```

Assume that each PUSH, POP and STACKEMPTY operation takes  $\Theta(1)$  time.

- (a) What is the worst-case time complexity of a single operation in a sequence of  $m$  ENQUEUE and DEQUEUE operations? Derive matching upper and lower bounds. That is, define an initial situation by describing what  $H$  and  $T$  look like at the start, and then define a sequence of  $m$  operations, where the sequence consists of ENQUEUE's and DEQUEUE's. Then show that one of the operations in the sequence (probably the last operation) will have the claimed worst-case time. For the upper bound, show that no operation in any  $m$ -operation sequence can ever take more time than the claimed worst-case time.

- (b) Use the accounting method to prove that the amortised time complexity of each operation in a sequence of  $m$  ENQUEUE and DEQUEUE operations is  $O(1)$ .
4. Consider the ADT *Labelled List*, defined as follows, for any positive integer  $u$ .  
An object of this type is a sequence of elements, where each element  $x$  has a label  $\lambda(x) \in \{0, \dots, u-1\}$  such that, if  $x$  occurs before  $y$  in the sequence, then  $\lambda(x) < \lambda(y)$ .  
There are four operations:

*INSERT*( $L, x, y$ ): Given an element  $x$  in a labelled list  $L$  and a new element  $y$ , insert  $y$  after  $x$  in the sequence represented by  $L$ .

*PREPEND*( $L, y$ ): Given a labelled list  $L$  and a new element  $y$ , insert  $y$  at the beginning of the sequence represented by  $L$ .

*DELETE*( $L, x$ ): Given an element  $x$  in a labelled list  $L$ , delete  $x$  from the sequence represented by  $L$ .

*LABEL*( $L, x$ ): Given an element  $x$  in a labelled list  $L$ , return the current label  $\lambda(x)$  of  $x$ .

Note that INSERT, PREPEND, and DELETE can change the labels of any number of elements in the labelled list.

**Hint:** For this entire question, consider using a linked list. For an insert, assume that you have a pointer to  $x$  as part of the input.

- (a) Invent an application for this ADT and briefly explain how you would use it in your application.
- (b) Give a data structure implementing the Labelled List ADT with  $u = 2^n$  that handles any sequence of operations containing at most  $n$  INSERTS and PREPENDS starting from an empty labelled list. You may assume that  $n$  is known in advance.  
All operations must run in worst case constant time.  
Explain why your algorithms are correct and run in constant time.
- (c) Give a data structure implementing the Labelled List ADT, with  $u = 2^n$ , where  $n$  is the maximum length of the list. You may assume that  $n$  is known in advance. The amortized time of your operations must be constant in any sequence of operations starting from an empty Labelled List.  
Explain why your algorithms are correct and run in constant amortized time.