Problem Set 6

Data Structures and Algorithms, Fall 2020

Due: October 29, in class.

Problem 1

Suppose we have stored n keys in a hash table of size m, with collisions resolved by chaining, and that we know the length of each chain, including the length L of the longest chain. Describe a procedure that selects a key uniformly at random from among the keys in the hash table and returns it in expected time $O(L \cdot (1 + 1/\alpha))$. Here, α is the load factor. You may assume there exists a function RANDOM(x, y) that returns an integer chosen uniformly at random from $[x, y]$, in $O(1)$ time.

Problem 2

(a) Consider a version of the division method in which $h(k) = k \mod m$, where $m = 2^p - 1$, k is a character string interpreted in radix 2^p, and $p > 1$ is an integer. (For example, if we use the 7-bit ASCII encoding, then $p = 7$ and string AB has key value $65 \times 128 + 66$.) Show that if we can derive string x from string y by permuting its characters, then x and y hash to the same value.

(b) Consider inserting the keys 10,22,31,4,15,28,17,88,59 into a hash table of length $m = 11$ using open addressing with the auxiliary hash function $h'(k) = k$. Illustrate the result of inserting these keys using linear probing, using quadratic probing with $c_1 = 1$ and $c_2 = 3$, and using double hashing with $h_1(k) = k$ and $h_2(k) = 1 + (k \mod (m - 1))$. (It suffices to show the eventual hash table.)

Problem 3

Define a family $\mathcal{H}$ of hash functions from a finite set U to a finite set B to be ϵ-universal if for all pairs of distinct elements k and l in U,

$$\Pr[h(k) = h(l)] \leq \epsilon$$

where the probability is over the choice of the hash function h drawn at random from the family $\mathcal{H}$. Show that an ϵ-universal family of hash functions must have

$$\epsilon \geq \frac{1}{|B|} - \frac{1}{|U|}$$

Problem 4

Design a data structure to support the following two operations for a dynamic multiset S of integers, which allows duplicate values:

- INSERT(S, x) inserts x into S.
- DELLARGEHALF(S) deletes the largest $\lceil |S|/2 \rceil$ elements from S.

Explain how to implement this data structure so that, starting from an empty set, any sequence of m INSERT and DELLARGEHALF operations runs in $O(m)$ time in total. Your implementation should also include a way to output the elements of S in worst-case $O(|S|)$ time.

Problem 5

Suppose that a counter begins at a number with b 1s in its binary representation, rather than at 0. Show that the cost of performing n INC operations is $O(n)$ if $n = \Omega(b)$. Do not assume that b is constant.

Problem 6

To conserve space for a stack, it is proposed to shrink it when its size is some fraction of the number of allocated cells. This supplements the array-doubling strategy for growing it. Assume we stay with the policy that the array size is doubled whenever the stack size grows beyond the current array size. Evaluate each of the following proposed shrinking policies, using amortized costs if possible. Do they offer constant amortized time per stack operation (i.e., push and pop)? You need to justify your answer. (The current array size is denoted as N.)

(a) If a pop results in fewer than $N/2$ stack elements, reduce the array to $N/2$ cells.

(b) If a pop results in fewer than $N/4$ stack elements, reduce the array to $N/4$ cells.

(c) If a pop results in fewer than $N/4$ stack elements, reduce the array to $N/2$ cells.

Problem 7

Consider the following modified algorithm for incrementing a binary counter.

INC($B[0, \cdots, \infty]$)

```
1: i ← 0
2: while (B[i] = 1) do
3:     B[i] ← 0
4:     i ← i + 1
5: B[i] ← 1
6: SOMETHINGELSE(i)
```

The only difference from the standard algorithm is the function call at the end, to a black-box subroutine called SOMETHINGELSE. Suppose we call INC n times, starting with a counter with value 0. The amortized time of each INC clearly depends on the running time of SOMETHINGELSE. Let $T(i)$ denote the worst-case running time of SOMETHINGELSE(i). For example, we proved in class that INC algorithm runs in $O(1)$ amortized time when $T(i) = 0$.

(a) What is the amortized time per INC if $T(i) = 4$? Justify your answer.

(b) What is the amortized time per INC if $T(i) = 2^i$? Justify your answer.

(c) What is the amortized time per INC if $T(i) = 4^i$? Justify your answer.