15-122 : Principles of Imperative Computation, Spring 2014

Written Homework 7

Due before class: Thursday, March 6, 2014

Name: _____

Andrew ID: _____

Recitation: _____

The written portion of this week's homework will give you some practice working with hash functions and hash tables. You can either type up your solutions or write them *neatly* by hand, and you should submit your work in class on the due date just before lecture begins. Please remember to *staple* your written homework before submission.

Question	Points	Score
1	7	
2	3	
Total:	10	

You must do this assignment in one of two ways and bring the stapled printout to the handin box on Thursday:

1) Write your answers *neatly* on a printout of this PDF.

2) Use the TeX template at `http://www.cs.cmu.edu/afs/cs.cmu.edu/academic/class/15122-s14/www/theory7.tgz`

1. **Dealing with Collisions**

Consider three implementations of a hash table that use different techniques for resolving collisions.

In the first hash table, we use separate chaining to resolve collisions. If the size of the hash table is m, then key k is added to the linked list that is referenced at index $h(k)$ mod m in the table, where h is the hash function being used. To resolve collisions, all keys that hash to the same index are stored in the same linked list (chain).

In the second hash table, we use linear probing to resolve collisions. In linear probing, if a key k is inserted or looked up, on the $(i+1)$st attempt we look at index $(h(k)+i)$ mod m, where h is the hash function being used and m is the size of the hash table. (We succeed in insert if we find NULL there; we succeed in lookup if we find an element there with matching key.)

> For example, if the hash function returns 4 and there is a key stored at index 4 of the hash table, we try index 5, then 6, then 7, etc. (with wraparound back to the beginning of the table if necessary) until we find an unoccupied cell.

In the third hash table, we use quadratic probing to resolve collisions. In quadratic probing, we follow a similar procedure as in linear probing, except we look at index $(h(k)+i^2)$ mod m on the $(i+1)$st attempt.

> For example, if the hash function returns 4 and there is a key stored at index 4 of the hash table, we try index 5 (= 4 + 1), then index 8 (= 4 + 4), then index 13 (= 4 + 9), etc. (with wraparound back to the beginning of the table if necessary) until we find an unoccupied cell.

NOTE: for this question, the hash function $h(k)$ does not perform a modulus by the table size; this is done afterwards. Also, for this question, you may assume that there is no integer overflow (i.e. even for large i, i^2 will still be non-negative).

(1) (a) For a hash table of size m with n keys, if $n = 2m$ and the keys are *not* evenly distributed and separate chaining is used to resolve collisions, what is the worst-case runtime complexity of a search for a specific key using big O notation?

Solution:

For a hash table of size m with n keys, if $n = 2m$ and the keys are evenly distributed and separate chaining is used to resolve collisions, what is the worst-case runtime complexity of a search for a specific key using big O notation?

Solution:

(2) (b) Assume that we hash a set of integer keys into a hash table of capacity $m = 13$ using a hash function $h(k) = k$.

Show how the set of keys below will be stored in the hash table by drawing the *final* state of each chain of the table after all of the keys are inserted, one by one, in the order shown. If a chain is empty, indicate NULL where appropriate.

`54, 23, 67, 88, 39, 75, 49, 5`

Solution:

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

(2) (c) Show where the sequence of keys shown below are stored in the hash table if they are inserted one by one, in the order shown, with $h(k) = k$ and $m = 13$, using linear probing to resolve collisions.

```
54, 23, 67, 88, 39, 75, 49, 5
```

Solution:

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

(1) (d) Show where the sequence of keys shown below are stored in the hash table if they are inserted one by one, in the order shown, with $h(k) = k$ and $m = 13$, using quadratic probing to resolve collisions.

```
54, 23, 67, 88, 39, 75, 49, 5
```

Solution:

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

(1) (e) Quadratic probing suffers from one problem that linear probing does not. In particular, given a non-full hashtable, insertions with linear probing will always succeed, while insertions with quadratic probing may or may not succeed (i.e. they may never find an open spot to insert). Using $h(k) = k$ as your hash function and $m = 6$ as your table capacity, give an example of a non-full hashtable and a key that cannot be successfully inserted using quadratic probing.

Solution: (HINT: Start by inserting the keys 36, 78, 12, 90.)

```
    0   1   2   3   4   5       Key to insert: ________
  -----------------------------
 |   |   |   |   |   |   |   |
  -----------------------------
```

2. **Strings as Keys**

In a popular programming language, strings are hashed using the following function:

$$(s[0] * 31^{p-1} + s[1] * 31^{p-2} + ... + s[p-2] * 31^1 + s[p-1] * 31^0) \ \% \ m$$

where $s[i]$ is the ASCII code for the ith character of string s, p is the length of the string, and m is the size of the hash table.

(1) (a) If 15105 strings were stored in a hash table of size 3021 using separate chaining, what would the load factor of the table be? If the strings above were equally distributed in the hash table, what does the load factor tell you about the chains?

Solution:

(2) (b) Using the hash function above with a table size of 3021, give an example of two different strings that would "collide" in the hash table and would be stored in the same chain. Show your work. (Use short strings please!)

Solution: