=============================================================================

CSC 263 Tutorial 5 Winter 2019

=============================================================================

1. Describe how to use a hash table (or a direct-address table) to solve each of the following problems. Provide as much detail as possible about the

hash table (size, hash function if possible) and analyse the complexity of

your solution. Then, answer the following questions: Is a hash table the

best way to solve the problem? Why or why not?

(a) Given an unsorted list of the names of every student in a class (with

n students), determine if two of them have the same names.

answer:

Hash every name into a hash table of size roughly n. This already  
detects whether or not any two keys are identical. In the worst-case, this requires time Theta(i) for the i-th insertion, so total time Theta(n^2).  
But on average, we expect that each insertion will take time Theta (1),  
so the total time is only Theta(n). Given what we know of hashing -- the worst-case is actually rare in practice -- this is an excellent solution.

(b) Given an unsorted list of the names of every student in a class (with

n students), return the list of names in sorted order. Does a hash table even make sense in this situation? Discuss.

answer:

In this situation, a hash table would be no better than the original unsorted array: using it would require us to generate the list of sorted names using a method like selection sort (linear search for the first name, followed by linear search for the second name, etc.). This would take total time Theta(n^2) -- **worse than just sorting the original input list.**

2. Recall the high-level implementation of each of the dictionary operations

for a hash table using chaining (covered in last week's lecture):

SEARCH(k):

search the linked list at T[h(k)] for k

INSERT(x):

search the linked list at T[h(x.key)] for x.key

if found:

replace old element with x

else:

insert x in a new node at the head of the linked list

DELETE(x):

search the linked list at T[h(x.key)] for x.key

if found:

remove the node with x from the linked list

In this question, you will explore "open addressing": hashing into a table where every element is stored directly in the table (without using linked lists at each location).

To insert into a hash table using open addressing, we examine ("probe")

different locations until we find one that is unused. The sequence of

locations examined is called a "probe sequence". Technically, the hash

function is redefined to take as input both a key and a "probe number":

h(k,i) gives the location to examine for key k and probe number i.

- For "linear probing," define h(k,i) = (h'(k) + i) mod m, where

h'(k) is an ordinary hash function.

- For "quadratic probing," define h(k,i) = (h'(k) + i^2) mod m, where

h'(k) is an ordinary hash function. It's also possible to

define a more general quadratic form but we will not study this.

- For "double hashing," define h(k,i) = (h'(k) + i h''(k)) mod m, where

h'(k) and h''(k) are two different hash functions such that h''(k) is

guaranteed to always be positive and relatively prime with m.

(a) Insert the keys 10, 22, 31, 4, 15, 28, 17, 88, 39 into a hash table

of size m = 11 using open addressing with the primary hash function

h'(k) = k mod m. Illustrate the results for linear probing, quadratic

probing, and double hashing with h''(k) = 1 + (k mod (m-1)).

answer:

Linear Probing:  
              key:  22  88          4   15  28  17  39  31  10  
            index:  0   1   2   3   4   5   6   7   8   9   10  
  
        Quadratic Probing:  
              key:  22  88          4   15  28  17      31  10  
            index:  0   1   2   3   4   5   6   7   8   9   10  
              \* key 39 cannot be inserted: every probe location fails...  
  
        Double Hashing:  
              key:  22      39  17  4   15  28  88      31  10  
            index:  0   1   2   3   4   5   6   7   8   9   10

(b) Write a detailed algorithm for INSERT for a hash table using open

addressing, in pseudocode. You may use h(k,i) for your hash function

without worrying about how it is implemented.

answer: (See Section 11.4 in CLRS.)

(c) Write a detailed algorithm for SEARCH for a hash table using open

addressing, in pseudocode. You may use h(k,i) for your hash function

without worrying about how it is implemented.

answer: (See Section 11.4 in CLRS.)

(d) Think about how to write DELETE for a hash table using open

addressing. Why can we not simply replace the element with NIL once we

find it? What could we do instead and how does this affect INSERT and

SEARCH? Discuss various options.

answer: See the discussion in section 11.4 in CLRS.