CS1102: Data Structure and Algorithms

Tutorial 10 - Hashing

Week of 5 April 2010

1. A good hash function is essential for good hash table performance. A good hash function is easy to compute and will evenly distribute the possible keys. Comment on the performance of the following hash functions. Assume the load factor α = number of keys / table size = 0.7 for all the following cases.

   a) The hash table has size 100. The keys are positive even integers. The hash function is:
   h (key) = key % 100

   b) The hash table has size 49. The keys are positive integers. The hash function is:
   *h (key) = (key * 7) % 49*

   c) The hash table has size 100. The keys are positive integers in the range of [0,10000]. The hash function is:
   h (key) = floor (sqrt (key)) % 100

   d) The hash table has size 2003. The keys are English words. The hash function is:
   h (key) = (sum of positions in alphabet of key's letters) % 2003
   E.g.: letter 'a' has position 1 and letter 'z' has position 26.

   e) The hash table has size 1009. The keys are *valid* email addresses. The hash function is: (see: http://www.asciitable.com/ for ASCII values)
   h (key) = (sum of ASCII values of last 10 characters) % 1009

   f) The hash table has size 101. The keys are integers in the range of [0,1000]. The hash function is:
   *h (key) = floor (key * random)% 101 , where 0.0 ⩽ random ⩽ 1.0*

2. Suppose that a hash table with size m = 13 and the following keys are to be mapped into the table in the sequence given:

```
10  100  32  45  58  126  3  29  200  400  0
```

   a) Show the final hash table after all the values are hashed using simple hash function key % m. Use linear probing to resolve collisions.
   b) Show the final hash table after all the values are hashed using the following hash function and use quadratic probing to resolve collisions.

```
int hashFunction(int key, int m) {
  int sum = digitSum(key); // digitSum add all digits of key
  return sum % m;          // e.g. key = 126, sum = 9
}
```

3. Binary Search versus Hash Table

Suppose that 511 distinct integers are arranged in ascending order in an array named **values**. Suppose also that the following code is used in a method to locate an integer named **key** in the array.

```
int leftIndex = 0;
int rightIndex = values.length() - 1;
while (leftIndex <= rightIndex) {
        int middleIndex = (leftIndex+rightIndex)/2;
        if (values[middleIndex] == key) {
                return true;
        } else if (values[middleIndex] < key) {
                leftIndex = middleIndex+1;
        } else {
                rightIndex = middleIndex-1;
        }
}
return false;
```

a. Compute the total number of comparisons necessary to locate all the 511 distinct integers in the array using the above code. Keep in mind that a comparison occurs in one of the two lines

```
if (values[middleIndex] == key) { ...
} else if (values[middleIndex] < key) { ...
```

You are NOT allowed to reuse the result from other searches; you can only perform each search separately.

b. Now suppose that the 511 distinct integers are already stored in a hash table with size p, where p is a prime. The hash function is $h(key) = key \% p$. We use separate chaining to resolve conflicts. We assume all the 511 distinct integers are uniformly distributed, and there are either ceiling(511/p) or floor(511/p) nodes in each chain. Find out the minimal p such that the number of comparisons necessary to locate all 511 values in this hash table is smaller than or equal to the result of Q3.a. Keep in mind that even though the first node in the chain is the value we are searching for, we still need a comparison.
You are NOT allowed to reuse the result from other searches; you can only perform each search separately.

c. One of the problems with separate chaining is that when there might be some very long chains, which will slow down the search, because we can only compare the value sequentially. One suggestion is instead of using linked list as a chain, we use a sorted array, i.e. each entry in the hash table contains a reference to an

sorted array. Arrays may have different sizes. Each array is sorted such that we can now apply binary search in Q3.a on it to speed up the search. We can use the same hash function as above and the same p as the result of Q3.b, but adopting this sorted array instead of linked list to perform separate chaining. Suppose the 511 integers are already stored in the new hash table. Is this a good solution in this case? Is this a good solution for general purpose? Explain the reasons.