Data Structures

Theoretical Homework 6

Abraham Murciano

February 12, 2020

1. After inserting the following numbers

   $$78, 41, 617, 18, 54, 64, 115, 8, 65, 76, 92, 47$$

   into a hash table of size 13, with chaining collision resolving, and with a hash function of $h(k) = k \,\%\, 13$, the load factor, α, will be

   $$\frac{n}{m} = \frac{12}{13}$$

   The average run-time for a successful search in this table would be $\Theta(1 + \alpha)$. This is because the key first has to be hashed, which takes constant time, then the resulting hash would lead us to a list of items. We have to then iterate over this list until we find out that the key we are looking for is indeed in this list. The average length of a list will determine the run-time.

   The average length of a lists in this particular instance of the hash table is

   $$\frac{1+1+2+0+1+1+1+0+2+0+0+2+1}{13} = \frac{12}{13} = \alpha$$

   Hence the average run-time for a successful search is one hash function, $\Theta(1)$, plus $\frac{\alpha}{2}$ comparisons to check if an item in a list is the one we are looking for. Hence $\Theta(1 + \alpha)$.

2. If we have more keys than buckets, it will be impossible to define a hash function that causes no collisions. This is because we are only able to define an injective function $f : A \to B$ if $|A| \leq |B|$. But our function is mapping from our set of keys, size of which is k, to m buckets. So we will not be able to make our hash function an injective function, meaning there must be collisions, because we are given $k > m$.

3. If we use a hash table with a slightly altered chaining collision resolution method by which we use **balanced** binary search trees instead of linked lists to store all the data which was mapped to a specific bucket, the run-times of the hash table operations are as follows.

   - For the search operation, the worst case scenario is where all the data was hashed to the same bucket, so we have n elements in a balanced tree. We must hash the key and then search said tree, time of which is $\Theta(\lg(n))$.

     The best case scenario is either where the key we seek is at the root of a tree, or if the key we seek is not in the tree and the tree is empty at the bucket the key hashes to. In these two cases, the search is $\Theta(1)$.

     The average case is $\Theta(1 + \lg(\alpha))$. Hashing the key is constant, and the average size of a tree is α. So searching the tree will on average take $\Theta(\lg(\alpha))$ time.

   - For the insertion operation we must first search for it to verify that it does not already exist in the table, by hashing the key and then searching the corresponding tree. Then we must insert it into the corresponding BST.

     The worst case is, again, when all the keys are in a single bucket. The search will take $\Theta(\lg(n))$, and the insertion (and re-balancing) will take $\Theta(\lg(n))$, so the total worst case run-time is $\Theta(\lg(n))$.

     The best case scenario is where there are no kays in the table that hash to the same value as the key to be inserted. So the search is $\Theta(1)$ and the insertion into an empty tree is $\Theta(1)$, so in total, $\Theta(1)$.

     Average insertion time would be the average search time of the hash table plus the average insertion time for a tree with α nodes. Therefore the total average insertion time is $\Theta(1 + \lg(\alpha))$.

- Finally, regarding the delete operation, the worst case is $\Theta(1)$ to hash, plus $\Theta(\lg(n))$ to search the tree, plus $\Theta(\lg(n))$ to delete from a tree with n nodes. This is similar to the worst case for the insert operation, so it is $\Theta(\lg(n))$.

  Best case run-time to delete is when the key we wish to delete is the only key in its bucket. Here, it takes $\Theta(1)$ to hash, $\Theta(1)$ to search the tree and $\Theta(1)$ to delete from a single-element tree. The total comes to $\Theta(1)$.

  Average case run-time to delete would be similar to the average for insertion, for similar reasons; so $\Theta(1+\lg(\alpha))$.

4. In a hash table with m buckets and a hash function

   $$h(k) = k \,\%\, m,$$

   it is possible to save space in the table by storing a shorter key $q(k)$ instead of the full key within each entry. This could be done by storing the quotient

   $$q(k) = \left\lfloor \frac{k}{m} \right\rfloor$$

   When searching the table for a specific key, we can reconstruct the key k of any table entry with the $q(k)$ that we stored and the $h(k)$ which we are computing during the search like so.

   $$k = q(k) \times m + h(k)$$

   However, this would only work in a hash table that uses chaining to resolve collisions. If the table uses open addressing instead, then it would be impossible to use this optimisation. This is because in open addressing, any key can potentially end up at any address, so there may be a different key in the same spot as the key we seek, but with the same quotient $q(k)$. Therefore it would be impossible to determine which key it originated from. On the other hand, when using the chaining method, the index of the bucket in which an entry is located is enough to give us the missing information to know which key any particular quotient key belongs to.

5. We are to insert the following keys into a hash table of size 13 using open addressing collision resolution.

   $$72, 18, 78, 11, 53, 31, 4, 15, 13, 14$$

   The primary hash function we are to use is

   $$h_1(k) = k \,\%\, 13$$

   (a) If we use linear probing, the resulting table will look like this:

Index	0	1	2	3	4	5	6	7	8	9	10	11	12
Key	78	53	15	13	4	18	31	72	14			11	

   The average number of key comparison steps it will take to search for each of the keys is

   $$\frac{1+1+1+1+1+2+1+1+4+8}{10} = \frac{21}{10} = 2.1$$

   (b) When using quadratic probing with $c_1 = 2$ and $c_2 = 1$, we would obtain the following table.

Index	0	1	2	3	4	5	6	7	8	9	10	11	12
Key	78	53	15	13	4	18		72	31			11	14

   This time the average number of key comparison steps for the search function is only

   $$\frac{1+1+1+1+1+2+1+1+2+3}{10} = \frac{14}{10} = 1.4$$

   (c) When using double hashing with a secondary hash function

   $$h_2 = 1 + (k \,\%\, 7),$$

   the resulting table would look like this.

Index	0	1	2	3	4	5	6	7	8	9	10	11	12
Key	78	53	15	14	4	18		72	13	31		11	

   The average number of comparison steps works out to

   $$\frac{1+1+1+1+1+2+1+1+4+3}{10} = \frac{16}{10} = 1.6$$

6. For the universal hash function

$$h_{a,b}(k) = ((ak + b) \,\%\, p) \,\%\, m$$

with $p = 691$ and $m = 7$,

$$\begin{aligned} h_{3,4}(315) &= ((3 \times 315 + 4) \,\%\, 691) \,\%\, 7 \\ &= ((945 + 4) \,\%\, 691) \,\%\, 7 \\ &= (949 \,\%\, 691) \,\%\, 7 \\ &= 258 \,\%\, 7 \\ &= 6 \end{aligned}$$

7. For a set of hash functions to be universal, the probability of choosing a random function from the set and having a collision ($h(x) = h(y)$ with $x \neq y$) for any specific x and y must be no greater than $\frac{1}{m}$, where m is the number of buckets in the hash table.

   Let the set of hash functions

$$H = \{h_{a,b}(k) = ((ak + b) \,\%\, p) \,\%\, m \quad : \quad a \in \mathbb{N}_0 \ \wedge \ b \in \mathbb{N}_1 \ \wedge \ a, b < p\}$$

   with p being some prime number greater than the greatest key to be hashed.

   In order to prove that H is not universal, we will show an example where for some p, m, x, and y, the number of functions in H which cause collisions is more than one $m^{\text{th}}$of the functions.

   Let $m = 13$, $p = 23$, $x = 2$, and $y = 4$. There are $p(p-1) = 506$ different functions in H. If more than $\left\lfloor \frac{p(p-1)}{m} \right\rfloor = 38$ of them cause collisions, then H is not universal.

   When $a = 0$, $h_{a,b}$ will always cause keys x and y to collide irrespective of the values of b, x, and y. Since there are $p - 1 = 22$ possible values for b, there are 22 such functions with $a = 0$.

   Aside from these 22 functions which cause our model to collide, there are an additional 20 functions which also cause collisions. These functions are:

$$\begin{aligned} &h_{5,3}(x) = h_{5,3}(y) = 0 \\ &h_{5,4}(x) = h_{5,4}(y) = 1 \\ &h_{5,5}(x) = h_{5,5}(y) = 2 \\ &h_{5,6}(x) = h_{5,6}(y) = 3 \\ &h_{5,7}(x) = h_{5,7}(y) = 4 \\ &h_{5,8}(x) = h_{5,8}(y) = 5 \\ &h_{5,9}(x) = h_{5,9}(y) = 6 \\ &h_{5,10}(x) = h_{5,10}(y) = 7 \\ &h_{5,11}(x) = h_{5,11}(y) = 8 \\ &h_{5,12}(x) = h_{5,12}(y) = 9 \\ &h_{18,10}(x) = h_{18,10}(y) = 0 \\ &h_{18,11}(x) = h_{18,11}(y) = 1 \\ &h_{18,12}(x) = h_{18,12}(y) = 2 \\ &h_{18,13}(x) = h_{18,13}(y) = 3 \\ &h_{18,14}(x) = h_{18,14}(y) = 4 \\ &h_{18,15}(x) = h_{18,15}(y) = 5 \\ &h_{18,16}(x) = h_{18,16}(y) = 6 \\ &h_{18,17}(x) = h_{18,17}(y) = 7 \\ &h_{18,18}(x) = h_{18,18}(y) = 8 \\ &h_{18,19}(x) = h_{18,19}(y) = 9 \end{aligned}$$

   Thus we have 42 functions which cause our chosen x and y to collide, which is more than 38. Therefore the probability of choosing a function that causes a collision is not smaller than $\frac{1}{m}$ for all x and y, so H is not universal.