Your name
CS 261, Spring 2020
Homework 2
Due Monday, April 13

1. Suppose that we use hash chaining for a hash table with n keys, but with a hash table that is too small, so that the load factor α is $3 \log n$. (This will cause operations on the hash table to take logarithmic time rather than constant time.) Use the Chernoff bound and linearity of expectation to prove that, for a random hash function and for this load factor, the expected number of hash chains that are at most twice their expected size is $o(1)$.

we can calculate the number of cells based on the load factor:

$$N = \frac{n}{\alpha} \tag{1}$$
$$= \frac{n}{3 \log n} \tag{2}$$

calculate the expected size of hash chains in one cell:

$$E[lk] = \frac{n}{N} = 3 \log n \tag{3}$$

Based on Chernoff:

$$Pr[X >= 2E[lk]] <= (\frac{e}{4})^{E[lk]} \tag{4}$$

This probability is an exponentially small function of E[lk]. Let's denote it as

$$Pr[X >= 2E[lk]] <= \frac{A * 3 \log n}{B * e^n} \tag{5}$$

where A and B are constants.
Thus, the expected number of hash chains is (5)*(2):

$$E[X >= 2E[lk]] <= Pr[X >= 2E[lk]] * N \tag{6}$$
$$= \frac{A * 3 \log n}{B * e^n} * \frac{n}{3 \log n} \tag{7}$$
$$= \frac{An}{Be^n} \tag{8}$$
$$= O(1) \tag{9}$$

■

2. In Python, there is a function hash() built into the language that can map most types of object to an integer. For an integer x, the value of hash(x) is just x mod $2^{31} - 1$, so for all integers smaller than $2^{31} - 1$, the hash of x is just x itself. Other types of objects have less-predictable hash values. The hash value, computed in this way, is used as the hash function for dictionaries by taking the result modulo the dictionary size. Python uses open addressing, but not linear probing, for its dictionaries.

Suppose you are given a hash table of fixed size s, initially empty, and that (unlike Python) you are going to use linear probing for this table, using the Python hash() function. Describe a sequence of $s/2$ integers such that, when inserted in that sequence into the table, the average time per insertion is $\Omega(s)$.

Say the first hash value is 1, inserting takes one operation. The second hash value is s+1, then inserting takes 2 operations. The third hash value is 2s+1, then inserting takes 3 operations. Continue this sequence, where the hash value is arithmetic progression and the difference is s. The average time per insertion is calculated below:

$$(1+\frac{s}{2})*\frac{s}{2}*\frac{1}{2}*\frac{1}{n}=\frac{2+s}{4} \tag{10}$$

This is $\Omega(n)$

■

3. Suppose that we insert n keys into an initially-empty cuckoo hash table, all of them successfully. Suppose also that, of the two hash functions $h1$ and $h2$ used for this insertion, $h1$ is bad and returns the same hash value for all n of the keys. In this scenario, how many pairs of keys can have equal values of $h2$? Use the case analysis from the lecture notes to prove that, in this scenario, all insertions take worst-case time $O(1)$.

Only one pair of keys can have equal values of h1, otherwise it will fall into infinite loop while insert keys into the hash table.

Say there is two arrays, G1 and G2 - one is to store the key-value pair based on h1 and the other one is to store the key-value pair based on h2. Since we only allow 2 keys to have equal value of h2, if there are M keys in total, there are M-2 keys have different h2 value. There are two cases:

1. a and b have the same h2 value. If a is already in G1 array, inserts b into G1, and then a will be bumped into G2. Here 2 operations in total.

2. a and b have the same h2 value and both of them have been inserted, b in G1 and a in G2 (as described in 1 above). Now inserting C: C will first be put in G1 where b is; b is bumped to G2 where a is; a is bumped back to G1 where C is; c will go to its position in G2 based on its h2 value. There are 4 operations in total. For all the values other than a and b, its insertion takes 4 operation if a and b are already inserted.

Thus, all insertions take worst-case time O(1).

■

4. The algebraic method for constructing a 2-independent hash function from the lecture notes chooses a large non-random prime number p and two random numbers a_0 and a_1 mod p, and defines the hash value for x to be $(a_1x + a_0)$ mod p mod N. Suppose we try to simplify this by skipping the "mod p" part. Instead, we choose two random numbers a_0 and a_1 mod N defining the hash value to be $(a_1x + a_0)$ mod N. Find a value of N for which this is not 2-independent, and explain why not.

If n equals to one of the common dividers of a_1 and a_0 or n equals to 1, this is not 2-independent. The reason is that, in this two cases, the hash value is always 0. ■