

Solutions to Practice Problems for Homework 2

1. Let the random variable X be the number of times that `A[i] != x` is executed.

- (a) Here X can only take on one of 3 values: 1 (occurs with probability $1/2$ when x is in `A[0]`), 2 (occurs with probability $1/4$ when x is in `A[1]`) and n (occurs with probability $1/4$ when x is not in `A`). Thus

$$E[X] = 1/2 \cdot 1 + 1/4 \cdot 2 + 1/4 \cdot n = n/4 + 1$$

- (b) Here X can take on values between 1 and n each with probability of $1/(2n)$ when the search is successful and with probability of $1/2$ it takes on a value of n (an unsuccessful search). So

$$E[X] = \left[\sum_{i=1}^n \frac{i}{2n} \right] + n \cdot \frac{1}{2} = \frac{1}{2n}(1 + 2 + \dots + n) + n/2 = \frac{1}{2n} \frac{n(n+1)}{2} + \frac{n}{2} = 3n/4 + 1/4$$

- (c) Here when the search is successful, X takes on value i with probability $(1/2)^i$ for $i = 1, \dots, n$ and when the search is unsuccessful, X takes on value n (and this occurs with probability $1/(2^n)$). Thus

$$E[X] = \left[\sum_{i=1}^n i \cdot (1/2)^i \right] + n \cdot \frac{1}{2^n} = 2 - \frac{1}{2^{n-1}} - \frac{n}{2^n} + \frac{n}{2^n} = 2 - \frac{1}{2^{n-1}}$$

2. Let the random variable X be the number of times that `ptr.value != x` is executed.

We have that:

position of item searched for	number times <code>ptr.value != x</code> done	probability
1	1	$1/2$
2	2	$1/4$
\vdots	\vdots	\vdots
i	i	$1/(2^i)$
\vdots	\vdots	\vdots
$n - 1$	$n - 1$	$1/(2^{n-1})$
n	n	$1/(2^{n-1})$

So expected number of times `ptr.value != x` is executed is $\left(\sum_{i=1}^{n-1} i/(2^i) \right) + n/(2^{n-1})$.

Applying formula that $\sum_{i=1}^k \frac{i}{2^i} = 2 - \frac{1}{2^{k-1}} - \frac{k}{2^k}$ with $k = n - 1$ yields that expected number of times `ptr.value != x` is executed is:

$$2 - \frac{1}{2^{n-2}} - \frac{n-1}{2^{n-1}} + \frac{n}{2^{n-1}} = 2 - \frac{2}{2^{n-1}} + \frac{1}{2^{n-1}} = 2 - \frac{1}{2^{n-1}}$$

So you on average you look at under two items (it approaches 2 as n goes to infinity).

3. Your implementation is to be designed for the following conditions. There are 10^8 phone numbers that will be inserted with 10^5 different area code/exchange (first 6 digits of the number) combinations occurring. Further, suppose you only have space in main memory to hold 8,000,000 bytes. (An integer takes 4 bytes (32 bits) and a long takes 8 bytes.)

For my primary data structure I will use a hash table, T , with open addressing for collision resolution. Each slot in T will hold the key (which is the area code/exchange pair (4 bytes)) and the page location for a secondary hash table (8 bytes). I have selected open addressing since there is no deletion required and by using open addressing we can fit the entire hash table in main memory.

The associated data for each area code/exchange pair there will be the location of a page holding a secondary hash table to keep all of the data for the phone numbers with that area code and exchange.

The key for the secondary hash tables will be the last 4 digits of the phone number and the associated data will be the name and address associated with that phone number. While we know that there are an average of $10^8/10^5 = 1000$ phone numbers in each of these, there could be as many as 10,000 entries in some. Hence for this table I will use chaining with a hash table size of 1000. (In the worst case of 10,000 entries in the table, we would expect each list to have 10 items which can still be searched quickly. Overall, there is just an expectation of one item per list.)

Let's now work out the hash table size for the primary hash table T . Let's suppose that we only want to use half of memory (4,000,000 bytes) for the primary hash table. Since $\lfloor 4000000/12 \rfloor = 333333$ a good size for the hash table is 262144 which is a power of 2 and so we can use the primary and secondary hash functions used from lab 2. Hence the load factor $\alpha = 100000/262144 \approx .38$. With this the expected number of probes in an unsuccessful search is $1 + 1/(1 - \alpha) \approx 1.62$.

We now describe how the required methods will be implemented.

- Insert a new area code/exchange combination into the system. (This will be used infrequently).

First an insertion is made into T with the given area code/exchange as the key. Then we create a new (empty) hash table for the given area code/exchange. This will take $O(1)$ time (we expected to make 1.62 probes for the insert) and then initialize an array of 1000 head pointers to "null" and write the page to disk.

- Insert into the system a new item that consists of a phone number, address, and name. (You can assume that the area code/exchange from the number corresponds to one of the area code/exchanges already in the system.)

First T is used to find the location of the page with the secondary hash table for the given area/code exchange pair. (Then the array of head pointers and associated lists for that hash table can be build in the 4,000,000 bytes that are still available in main memory). This step has $O(1)$ expected time to find the location of the page. Then in $O(1)$ time the page can be read into memory. Then using the last 4 digits as the key we hash to find the slot of the secondary hash table that holds the location of the list for that phone number and insert the new item in the list. This step has $O(1)$ expected time. Finally, the page can be written back to disk in $O(1)$ time. Hence overall this has expected $O(1)$ time complexity.

- Given a phone number, return the address and name.

First T is used to find the location of the secondary hash table for the given area/code exchange pair. (Then that hash table can be brought into memory as discussed above). Then using the last 4 digits as they key we hash to find the slot in the secondary hash function that holds the location of the list for that phone number and finally we search that list to see if the given key is in use. If so, the name can be retrieved. Each step takes $O(1)$ expected time.

- Given a phone number, remove the corresponding item from the system.

This is just like the above, but we remove the item from the list in the secondary hash table. Thus it too takes $O(1)$ expected time.