CSC263 – Problem Set 4 - Updated!

Due **Saturday, October 22** before 10pm

Question 1

Suppose we have have two hash tables H_1 and H_2 which use the same hash function h, have the same array size m, and use closed addressing (with linked lists) to resolve collisions. Let n_1 and n_2 be the number of keys-value pairs stored in H_1 and H_2, respectively.

Assume for this question that there are no duplicate keys between H_1 and H_2 (in other words, the two hash tables have a combined set of $n_1 + n_2$ distinct keys). Also assume that all hash functions run in constant time.

(a) **[2 marks]** Give an algorithm for merging H_2 *into* H_1. Note that the resulting H_1 should still have the same hash function and array size, but now contains $n_1 + n_2$ keys. H_2 remains unchanged. Your algorithm must have worst-case running time $\mathcal{O}(n_1 + m)$, and in particular **must not** depend on n_2 (don't assume any relationships between n_1, n_2, and m).

In addition to the pseudocode, include a brief English explanation (2-3 sentences) of why your algorithm is correct.

(b) **[3 marks]** Prove that the worst-case running time of your algorithm is $\mathcal{O}(n_1 + m)$.

Note that the m term may be a bit surprising! Be sure to explain where it comes from. (If you can prove your algorithm runs in $\mathcal{O}(n_1)$ time, this is sufficient to also prove an upper bound of $\mathcal{O}(n_1 + m)$. But be *very* confident that your analysis is correct.)

(c) **[2 marks]** Now suppose H_1 and H_2 use different hash functions h_1 and h_2, but everything else is the same as in the original problem description.

Give an algorithm for merging H_2 into H_1 under this assumption; your algorithm's running time must not depend on n_1.

(d) **[2 marks]** Give a good upper bound on the worst-case running time of your algorithm from part (c).

Question 2

In this question, you'll perform a more sophisticated analysis of the open addressing strategy that uses the simplest linear probing sequence:

$$h(k, i) = hash(k) + i \pmod{m}$$

where m is the length of the array, and $hash$ is some hash function.

For this question, assume that $n < m$ distinct keys are randomly chosen and inserted into the hash table, and that for each key choice the simple uniform hashing assumption applies:

$$\forall 0 \leq i < m,\ \Pr[hash(k) = i] = \frac{1}{m}.$$

Remember to fully justify your answers to this question!

(a) **[3 marks]** Let P_n be the probability that the n keys are stored in n consecutive locations in the hash table, where "consecutive" is modulo m. (Or in other words, the indices $\{m-2, m-1, 0, 1, 2\}$ are considered consecutive.)

~~Find an exact, simplified expression for P_n in terms of n and m; remember that you can assume the simple uniform hashing assumption, and that collisions are resolved using the linear probe sequence defined above.~~

Updated: find an exact values for P_2 and P_3 in terms of m. (There's some case analysis, but because the numbers are small, it should be much more tractable!)

(b) **[3 marks]** Now assume that the n keys have been inserted, and they are all stored in consecutive locations in the hash table.

Suppose a new key k (distinct from all the others) is chosen, and the simple uniform hashing assumption applies to this choice as well (i.e., all hash values are equally likely).

Find the average number of array spots visited when this key k is inserted, in terms of n, m, and/or α. Here, "visited spots" includes the empty spot where k is eventually inserted.

(c) **[3 marks]** Repeat part (b), except now assume that *none* of the n keys occupy consecutive spots in the array. (That is, $n \leq \frac{m}{2}$ and a key is always surrounded by empty spots.)