University of Waterloo
CS240 - Spring 2017
Assignment 4

Due Date: Wednesday, July 5th, 5pm

Please refer to the course webpage for guidelines on submission. Submit your written solutions electronically as a PDF with file name `a04wp.pdf` using MarkUs. We will also accept individual question files named `a04q1w.pdf`, `a04q2w.pdf`, etc., if you wish to submit questions as you complete them.

Unless the base is explicitly specified, we assume in all questions that the logarithm operation is base 2.

Problem 1 [5+5+2=12 marks]

Suppose you want to sort a set of n strings over an alphabet of size d. Furthermore, suppose that the maximum-length string in your set is of length $\ell_{\max}$, and that the average length of the strings in your set is ℓ_{avg}. We assume that our sorted set will follow the standard alphabetical ordering; for example, `a` $<$ `ab` $<$ `b`.

(a) One method we saw in class for accomplishing this task is *radix sort*. First, we pad all strings of length less than $\ell_{\max}$ to have length $\ell_{\max}$, then we run radix sort on our set of strings as usual. Using the parameters defined above, describe the runtime and the amount of space used by this approach.

(b) Another method to accomplish this task makes use of *multiway tries*. First, we build a multiway trie containing every string in our set. Assume that we have k nodes in our multiway trie, and that each node is stored in memory as an array of size d. Then, we perform a preorder traversal and read each string, giving us a sorted ordering. Using the parameters defined above, describe the runtime and the amount of space used by this approach.

Hint. When describing the runtime, consider separating out the total time needed to initialize new nodes in the multiway trie.

(c) For which situations would the radix sort approach be preferable? For which situations would the multiway trie approach be preferable?

Problem 2 [6+6=12 marks]

Consider a set of positive integers $X = \{x_1, \ldots, x_n\}$ where each x_i is represented in base 2^s — that is, over the alphabet $\{0..2^s - 1\}$ — and $x_i \leq n^c$ for some constant c. We assume for simplicity that s divides $c \log n$.

Suppose we store all x_i strings in a compressed trie T in the following way. For each node $v \in T$, we maintain an array A_v of size 2^s. In this array, $A_v[j]$ contains a pointer to the child v_j of v such that the edge of T connecting v and v_j is labelled with the symbol j. If no such v_j exists, then $A_v[j] = \emptyset$.

(a) What is the upper bound on the maximum height of T in terms of s, n, and c?

(b) What is the upper bound on the maximum space usage of all arrays A_v? Consequently, what is the upper bound on the maximum space usage of T?

Problem 3 [4+4+4+4=16 marks]

Consider a hash table dictionary for integers from the universe $U = \{0, 1, \ldots, 25\}$. If we have a hash table of size $M = 5$, and we are given the insertion sequence $\{17, 10, 20, 13\}$, draw the resulting hash table if we resolve collisions using:

(a) Chaining with $h(k) = (k + 2) \bmod 5$.

(b) Linear probing with $h(k) = (k + 2) \bmod 5$.

(c) Double hashing with $h_1(k) = (k + 2) \bmod 5$ and $h_2(k) = 1 + (k \bmod 4)$.

(d) Cuckoo hashing with $h_1(k) = (k + 1) \bmod 5$ and $h_2(k) = \lfloor k/5 \rfloor$.

Show the hash table after each insertion as well as the final table after all insertions.

Problem 4 [7+5+5+3=20 marks]

In this question, we consider a modified version of open-addressing hashing. In this modified version, the insert operation works as follows. To insert a key k, we perform linear probing until we either reach an empty position in the hash table or probe a position that is not empty. Say the nonempty position in the table contains the key k'. Then, if $k' > k$, we replace k' with k at that position and call the insert operation on k'. In this way, the value of the key we are trying to insert into the hash table is strictly increasing.

For each question that follows, we may assume that no key is ever deleted from the hash table.

(a) For a hash table of size $M = 10$ and the hash function $h(x) = x \bmod 10$, run this modified hash algorithm using the insertion sequence $\{31, 26, 16, 23, 11, 30, 20\}$. Show the hash table after each insertion is complete; that is, after no more probing is required.

(b) Argue that, regardless of the values of M and $h(x)$, the insertion operation will always terminate after a finite number of probes.

Hint. You may assume that there is still space in the hash table upon each insertion and that all probe sequences consist of some permutation of the positions of the hash table.

(c) Argue that the number of probes made during an insert could be $\Omega(n^2)$. That is, for arbitrarily large values of n, give an example of a hash table with n keys and size $M > n$ such that insertion of a key into the hash table will require at least cn^2 probes for some constant $c > 0$.

The most straightforward approach would be to use linear probing with the hash function $h(x) = x \bmod M$, but full credit will be given for any other hash function and probe sequences, so long as all probe sequences consist of some permutation of the positions of the hash table.

(d) The main advantage of our modified algorithm is that we can use fewer probes for an unsuccessful search than with arbitrary open-addressing hashing. Explain why this is the case by describing a modification of the search operation for our modified hashing algorithm such that the number of probes for an unsuccessful search is significantly reduced. You need not provide a formal analysis, but some justification of your modification to the search operation is required.