University of Waterloo
CS240 Spring 2022
Assignment 4

Due Date: Wednesday, July 6 at 5:00pm

The integrity of the grade you receive in this course is very important to you and the University of Waterloo. As part of every assessment in this course you must read and sign an Academic Integrity Declaration before you start working on the assessment and submit it **before the deadline of June 1** along with your answers to the assignment; i.e. **read, sign and submit A04-AID.txt now or as soon as possible**. The agreement will indicate what you must do to ensure the integrity of your grade. If you are having difficulties with the assignment, course staff are there to help (provided it isn't last minute).

The Academic Integrity Declaration must be signed and submitted on time or the assessment will not be marked.

Please read https://student.cs.uwaterloo.ca/~cs240/s22/assignments.phtml#guidelines for guidelines on submission. **Each question must be submitted individually to MarkUs as a PDF** with the corresponding file names: a4q1.pdf, ... , a4q5.pdf . It is a good idea to submit questions as you go so you aren't trying to create several PDF files at the last minute.

Late Policy: Assignments are due at 5:00pm on Wednesday. Students are allowed to submit **one** late assignment, 2 days after the due date on Friday by 5:00pm. Assignments submitted after Friday at 5:00pm or Wednesday at 5:00pm (if you have already used your one late submission) will not be accepted for grading but may be reviewed (by request) for feedback purposes only. If you want to use your one time late assignment allowance, please send an email to "cs240@uwaterloo.ca" with the title "Using one time late assignment allowance".

Problem 1 [6 marks]

Consider a set $X = \{ x_1, \ldots, x_n \}$ where every x_i is a positive integer and $x_i \leq n^f$ for some constant f. We represent x_i as strings over an alphabet $0, 1, \ldots, 2^t - 1$ (i.e., representing each x_i in base 2^t) and store all x_i from X in a compressed trie T. For every node u of T we keep an array $A(u)$ of size $2^t + 1$. $A(u)[i]$ contains a pointer to the child u_i of u such that the edge from u to u_i is marked with i; $A(u)[i] =$ `NULL` if there is no such u_i in T. You can assume for simplicity that t divides $f \log n$.

Specify all of the following in big-O in terms of f, n, and t (as necessary). What is the (maximum) height of T? What is the total space usage of T and all $A(u)$? What time is

needed to search for a key k in T ?

Solution:

If $x_i \leq n^f$ is represented as a string over an alphabet of size $R = 2^t$, then the length of this string is at most $\lceil \log_R n^f \rceil = O(\frac{f \log n}{t})$. ($O(\min\{n, \frac{f \log n}{t}\})$ is also accepted)

A compressed trie on n strings has up to $n-1$ internal nodes and n leaves. But we need to store an array $A(u)$ of size $O(2^t)$ in every internal node u. Hence the total space used by internal nodes is $O(n2^t)$ in the worst case. For each leaf node, it takes $O(\frac{f \log n}{t})$ to store the key. Hence the total space used by leaf nodes is $O(n\frac{f \log n}{t})$ in the worst case. Overall, the total space used by the whole data structure is $O(n \cdot 2^t + n \cdot \frac{f \log n}{t})$. ($O(n \cdot 2^t)$ is also accepted)

To search for a key k, we must move down from the root of T to the leaf that corresponds to k. Hence the worst-case search time is proportional to the height of T. The search time is $O(\frac{f \log n}{t})$. ($O(\min\{n, \frac{f \log n}{t}\})$ is also accepted)

For instance if $t = \sqrt{\log n}$ the query time is $O(\sqrt{\log n})$ (for a constant f). This shows that we can achieve sub-logarithmic search time when dictionary contains integers and we have sufficient memory.

Problem 2 [2+2+2+2+3+4 marks]

(a) Draw the standard trie that is obtained after inserting the following keys into an initially empty trie:

$$1011\$, 001\$, 1101\$, 10010\$, 10\$, 11\$, 10100\$, 1\$, 000\$, 101\$, 00\$$$

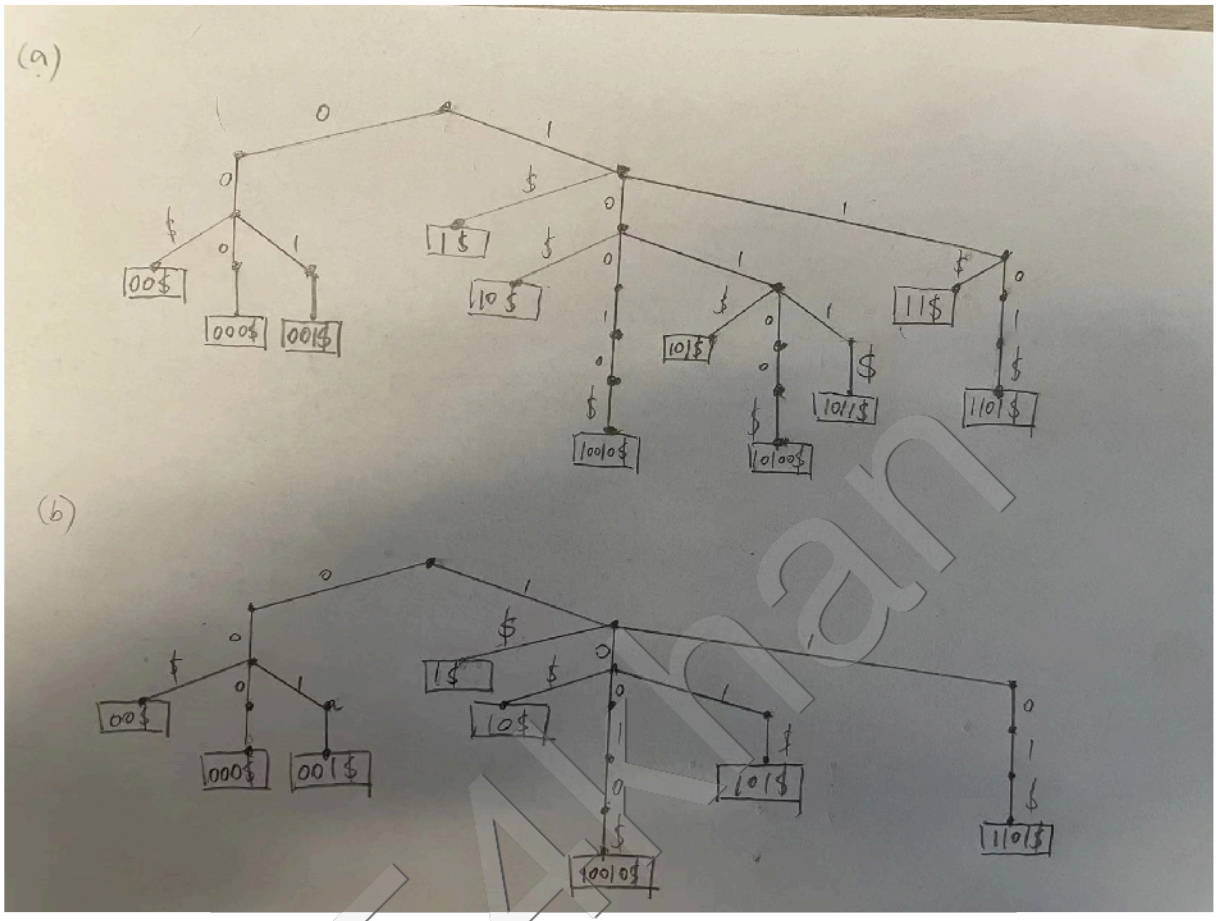
(b) From your answer to part (a), draw the trie that is obtained after removing the following keys:

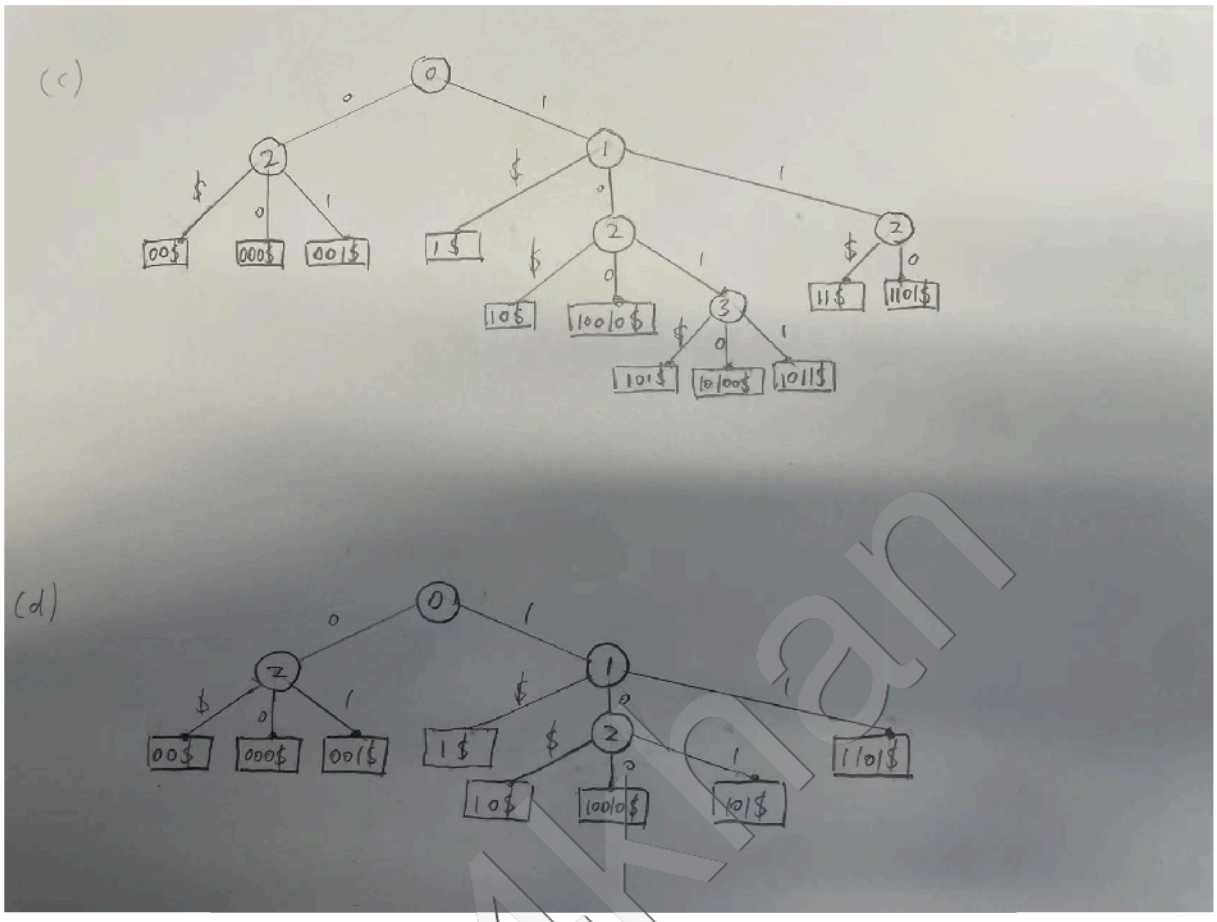
$$10100\$, 11\$, 1011\$$$

(c) Repeat part (a), except use a compressed trie.

(d) Repeat part (b), but on the compressed tree that is obtained at (c).

Solution to (a)(b)(c)(d):





(e) Find the exact height of the trie with keys that are binary representations of numbers $0, 1, 2, 3, 4, \ldots, 2^k - 1$ without leading 0s, and inserted into the trie in increasing order. That is, insert $0\$, 1\$, 10\$, 11\$, 100\$$, etc. Justify your answer.

Solution: For $k = 0$ the only key is $0\$$ which gives height 2. For $k > 0$ the longest string $2^k - 1$, which is $11\cdots1\$$ (with k ones) has length k. This gives a path of length $k + 1$ in the trie. So the height of tree is $k + 1$.

(f) Repeat part (e) using compressed tries. Also, for this part, prove your result, including any structural properties of the compressed tries, using mathematical induction.

Note that in order to obtain full marks for proofs involving mathematical induction, solutions must clearly state:

- what you are doing induction on;
- the base case(s);
- the inductive hypothesis;
- the inductive step.

Solution: The height of the compressed trie is k (except for $k = 0$ which gives height 1). For $k > 0$ we prove this by induction on k.
Base case: if $k = 1$ then the strings are 0\$, 1\$ which give a compressed trie of height 1.

Now assume the compressed trie T, formed by elements of $0, 1, \ldots, 2^{k-1} - 1$ has height $k - 1$(induction hypothesis). We want to show that the compressed trie T', formed by $0, 1, \ldots, 2^k - 1$ has height k (induction step). We know that T' is obtained by inserting elements of $A = \{2^{k-1}, 2^{k-1} + 1, \ldots, 2^k - 1\}$ into T.

Every number in A has a $(k-1)$-prefix followed by a 0 or 1. Every such prefix is already a leaf in T. Therefore, appending a 0 or 1 to those leaves increases the height of the trie by one without the possibility of any compression. Hence the height of T' is one more than T. This completes the induction.

Problem 3 [3+3+3+3+1 marks]

Consider a hash table dictionary with universe $U = \{0, 1, 2, \ldots, 25\}$ and size $M = 5$. If items with keys $k = 17, 10, 20, 13$ are inserted in that order, draw the resulting hash table if we resolve collisions using:

(a) Chaining with $h(k) = (k + 2) \bmod 5$.
Solution: The final hash table after all insertions is $[\{13\}, \emptyset, \{20, 10\}, \emptyset, \{17\}]$, where brackets indicate a linked list. Note that we're assuming that we insert at the head of a linked list. If we insert at the tail, we get an equivalent answer with the keys 10 and 20 swapped.

(b) Linear probing with $h(k) = (k + 2) \bmod 5$
Solution: The final hash table after all insertions is $[13, \emptyset, 10, 20, 17]$.

(c) Double hashing with $h_1(k) = (k + 2) \bmod 5$ and $h_2(k) = 1 + (k \bmod 4)$.
Solution: The final hash table after all insertions is $[13, \emptyset, 10, 20, 17]$.

(d) Cuckoo hashing with $h_1(k) = (k + 2) \bmod 5$ and $h_2(k) = \lfloor k/5 \rfloor$.
Solution: The final hash tables after all insertions are primary:$[13, \emptyset, 20, \emptyset, 17]$, and secondary:$[\emptyset, \emptyset, 10, \emptyset, \emptyset]$.

(e) Identify a serious problem with the choice of hash functions in part (d).
Solution: $h_2(25) = 5$ and index 5 does not exist in the hashtable.

Problem 4 [3+4 marks]

(a) Assume that we have a hash table of size 6 and that our keys are selected uniformly at random from the set $A = \{1, 2, 3, \ldots, 600\}$. Consider the following two hash functions:

$$h_1(k) = k \mod 6,$$

$$h_2(k) = 2k \mod 6.$$

Which hash function is better? Justify your answer.

Solution: h_1 is better, since all values of h_2 have even values.

(b) Let S be a set of n keys mapped to a hash table also of size n using chaining. We make the uniform hashing assumption, and we call c_n the expected number of empty slots. Prove that $\lim_{n\to\infty} c_n/n = 1/e$, where $e \approx 2.71828183$.

You can use the fact that $\lim_{n\to\infty}(1-1/n)^n = 1/e$. We recommend you use indicator variables $I_0, \ldots, I_{n-1}$, that take values 0 or 1, depending on whether the corresponding entry in the table is empty or not; then, find the expected value of each I_i.

Solution: Let

$$I_j = \begin{cases} 1 & \text{if } T[j] \text{ is empty} \\ 0 & \text{otherwise} \end{cases}$$

The number of empty slots is $\sum_{j=0}^{n-1} I_j$, so the expected value is $c_n = \sum_{j=0}^{n-1} E[I_j]$.

The probability that and element x does not map to $T[j]$ is $\frac{n-1}{n} = 1 - \frac{1}{n}$ and the probability that none of the n elements map to $T[j]$ is $(1-\frac{1}{n})^n$. Hence

$$E[I_j] = (1-\frac{1}{n})^n.$$

Therefore

$$c_n = \sum_{j=0}^{n-1}(1-\frac{1}{n})^n = n(1-\frac{1}{n})^n.$$

Finally

$$\lim_{n\to\infty}\frac{c_n}{n} = \lim_{n\to\infty}(1-\frac{1}{n})^n = \frac{1}{e}.$$

Problem 5 [4+4+4 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'. If $k' \leq k$, continue trying to insert k in the position after k', as is done in normal linear probing. 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.
Solution: The final hash table after all insertions is $[20, 11, 30, 23, 31, \emptyset, 16, 26, \emptyset, \emptyset]$.

(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.

Solution: The crucial observation is that the value of the key that we're trying to insert is strictly increasing. More precisely, for the original insert (say, of key k_1), we use at most M probes since the hash table is not full and the probe sequence is a permutation; hence, at some point in the probe sequence, we will find the empty slot that exists. If, during that insert, we encounter a key that is larger than k_1 (say, key k_2) then for k_2 we yet again use at most M probes, and we know that $k_2 > k_1$. This continues: for every key for which we need to probe, we will use at most M probes and the key will be larger than all previous ones. In particular, we may call the insertion routine for each key at most once, which means that we use at most nM probes to insert the key. (Note that the largest possible number of probes is actually smaller than nM .)

(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 $c > 0$. Justify insertions.

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 as long as it can map to all entries of the table.

Solution: Consider a hash table of size $2n$, and let the key in slot i have value $(i+1)\cdot 2n$, for $i = 0, \ldots, n-1$. The hash function is $h(k) = k \mod 2n$, and we use linear probing. For example, for $n = 5$, the hash table is $[10, 20, 30, 40, 50, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset]$. Note that (sadly) all keys hash to the same value, namely $h(k) = ((i+1)2n) \mod 2n = 0$. Therefore, all keys also have the same probe sequence, namely $\{0, 1, 2, 3, \ldots\}$. We insert a key 0, which hashes to 0. It probes slot 0, where it finds a larger key (namely, $1 \cdot 2n$), which it evicts. Then key $1 \cdot 2n$ gets reinserted. It probes slot 0, then slot 1, where it finds a larger key (namely, $2 \cdot 2n$), which it evicts. Then key $2 \cdot 2n$ gets reinserted. It probes slot 0, then slot 1, then slot 2, where it finds a larger key (namely, $3 \cdot 2n$), which it evicts. This repeats; when key $i \cdot 2n$ gets reinserted, it probes slot 0, then 1, then 2, etc., until it reaches slot i, where it finds a larger key (namely, $(i+1) \cdot 2n$), which it evicts. Finally, key $n \cdot 2n$ (the largest key that was already in the table) probes slots

0,1,2, etc., until it reaches slot $n+1$, which is free. The key is reinserted there and the operation ends. All in all, for each i, key $i \cdot 2n$ required $i+1$ probes until it either found a larger key in an occupied slot or it found an empty slot. Hence, the total number of probes is

$$\begin{aligned}\sum_{i=0}^{n}(i+1) &= (n+1)(n+2)/2 \\ &\geq n^2/2.\end{aligned}$$