University of Waterloo
CS240 Fall 2020
Assignment 4

Due Date: Wednesday, Nov 18 at 5pm

The integrity of the grade you receive in this course is very important to you and the University of Waterloo. As part of every asessment 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 November 18th** 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 `http://www.student.cs.uwaterloo.ca/~cs240/f20/guidelines.pdf` for guidelines on submission. **Each question must be submitted individually to MarkUs as a PDF** with the corresponding file names: a4q1.pdf, a4q2.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. **Remember, late assignments will not be marked.**

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 $\ell_{\max}$ with a suffix of end-of-word characters so that all strings 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 contains an array of children pointers of size $d + 1$. 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 [2+2+3+3=10 marks]

Consider a hash table dictionary with table of size $M = 10$ and hash function $h_1(k) = k \bmod 10$. Given the following insertion sequence: $4371, 1323, 6173, 4199, 4344, 1679, 1989$, draw the resulting hash table if we resolve collisions using the following methods:

(a) Separate chaining

(b) Linear probing

(c) Double hashing with a second hash function of $h_2(k) = \lfloor x/1000 \rfloor$.

(d) Cuckoo hashing with a second hash function of $h_2(k) = \lfloor x/1000 \rfloor$.

Problem 3 [3+3+3+2=11 marks]

In this question, we consider a modified version of the open-addressing hashing method linear probing. The sequence of probes for a particular key follows linear probing, however, inserting a key may require additional calls to the insertion operation with different keys. Inserting a key k works as follows:

- If the probe position is empty, we simply insert key k.
- If the probe position is not empty and contains key k', then, if $k' > k$, we replace k' with k at that position and call the insert operation on k'. If $k' < k$, then we continue with key k and check the next probe position. 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, as long as there is an open space in 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, $M > n$, give an example of inserting n keys into an empty hash table where the last insertion requires at least cn^2 probes for some constant $c > 0$.

The most straightforward approach would be to use the hash function $h(x) = x \bmod M$, but full credit will also be given for any other hash function. However, your hash function must map to all indices of the hash table; for example, a constant hash function like $h(k) = 0$ is not a valid solution.

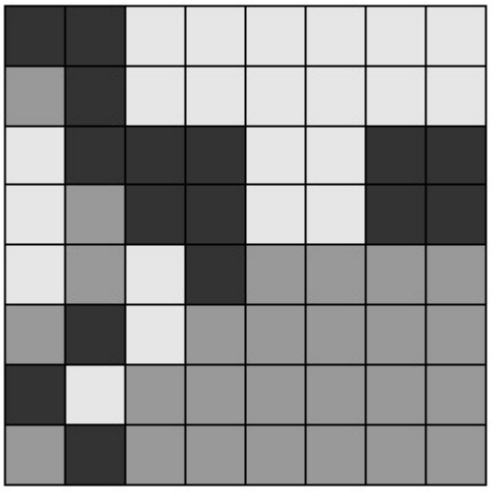
(d) Consider the arrangement of keys in the hash table resulting from this algorithm and give a modified version of the search operation that, in many cases, will reduce the number of probes required for an unsuccessful search. Briefly describe the cases where your modified algorithm reduces the number of probes and why the probe count is reduced.

Problem 4 Quadtrees [2+3+5=10 marks]

For all parts of this question, use the convention that each internal node of a quadtree has exactly four children, corresponding to regions NE, NW, SW and SE, in that order.

a) Give three 2-dimensional points such that the corresponding quadtree has height at least 50. Give the (x, y) coordinates of the three points and show the shape of the quadtree. (Do not give the plane partitions.)

b) One application of quadtrees is image compression. An image (picture) is recursively divided into quadrants until the entire quadrant is only one colour. Using this rule, draw the quadtree of the following image. There are only three colours (shades of grey). For the leaves of the quad tree, use 1 to denote the lightest shade, 2 for the middle shade and 3 for the darkest shade of grey.



c) Another application is to compare two images. Given two black and white images (i.e. each pixel of the image is either 0 or 1) each of size $2^k \times 2^k$ store as quadtrees, give algorithms for the following operations:

i) Union. If corresponding pixels are both 0, then their union is 0; otherwise 1.

ii) Intersection. If corresponding pixels are both 1, then their intersection is 1; otherwise 0.

Runtime analysis is not required.

Problem 5 [3+1+6=10 marks]

Using kd-trees, we can report all points in a two-dimensional range in time $O(n^{1/2} + s)$, where n is the number of points stored in the data structure and s is the number of reported points. In this problem, you will describe a data structure that needs $O(n)$ space and answers two-dimensional range reporting queries in $O(n^{1/3}\log(n)+s)$ time. We assume for simplicity that all points have different x-coordinates.

(a) As a warm-up, consider the following particular case of a 2d range query: we store a set S of r points, and we have to process a query rectangle $Q = [a, b] \times [c, d]$, *but we know that all points in S satisfy the constraint $a \leq x \leq b$.* What data structure would you use to store the points in S, and how long would the query take?

Let S denote the set of points stored in the data structure we want to design. Divide S into equal-sized subsets S_i for $i = 1, \ldots, m$, with $m = n^{1/3}$, according to x-coordinates of points: for any points p in S_i and q in S_j, the x-coordinate of p is smaller than the x-coordinate of q if and only if $i \leq j$ (we assume that m is an integer, for simplicity). We keep two different data structures for every set S_i; one of them is a kd-tree that stores all points of S_i (you will have to find what the other one is).

Hint: you may want to add some fields to the sets to store information.

(b) How many elements are there in S_i, and what would be the cost of a range search in S_i, using a kd tree?

(c) Describe the data structure and give an algorithm that achieves the run-time claimed above (you have to justify the run-time).