University of Waterloo
CS240 - Spring 2023
Assignment 5

Due Date: Wednesday July 26, 5pm

You should have submitted AID02 before the due date of this assignment. 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/s23/assignments.phtml#guidelines` for guidelines on submission. **Each question must be submitted individually to MarkUs as a PDF** with the corresponding file names: a5q1.pdf, a5q2.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, with the grace period until 11:59pm. Assignments submitted after 11:59pm on the due date will not be accepted but may be reviewed (by request) for feedback purposes only.

Question 1 [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 $h(k) = (k + 2) \bmod 5$.

(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 $h_1(k) = (k + 2) \bmod 5$ and $h_2(k) = \lfloor k/5 \rfloor$.

(e) Identify a serious problem with the choice of hash functions in part (d).

Question 2 [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.

(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.

Question 3 [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 $k' \le k$, 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.

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

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 $c > 0$. Justify insertions.

The most straightforward approach would be to use linear probing with the hash function $h(x) = x \bmod M$, but you can choose to use another hash function if you want (as long as it can map to all entries of the table).

Question 4 [4+6+5 marks]

(a) Give the (x, y) coordinates of three points in the plane for which the corresponding quadtree has height exactly 6. (Do not give the plane partitions.)

(b) The *spread* of a point set is the ratio of the maximum distance between points to the minimum distance between points. Prove that spread of any set of n points in the plane is $\Omega(\sqrt{n})$.

(c) A set of n points in the plane is called *dense* if its spread is $O(\sqrt{n})$. Prove that the height of the quadtree on any dense point set is $O(\log n)$.

Question 5 [4+2+8 marks]

Using kd-trees, we can report all points in a two-dimensional range in time $O(n^{1/2} + k)$, where n is the number of points stored in the data structure and k 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 $O(n^{1/3} \log(n) + k)$ 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 $i = 1, ..., m$, 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 $i < j$ (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).

(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).

Hint: if we consider a query $Q = [a, b] \times [c, d]$, consider which sets S_i need to be queried, and observe that (at most) two of them should be handled differently from the other ones. The two data structures for S_i will be handy here.