University of Waterloo
CS240E, Winter 2024
Assignment 4

Due Date: Tuesday, March 19, 2024, at 5pm

Be sure to read the assignment guidelines (`http://www.student.cs.uwaterloo.ca/~cs240e/w24/assignments.phtml`). Submit your solutions electronically as individual PDF files named a4q1.pdf, a4q2.pdf, ... (one per question).

Grace period: submissions made before 11:59PM on March 20, will be accepted without penalty. Please note that submissions made after 11:59PM **will not be graded** and may only be reviewed for feedback.

Question 1 [8 marks]

Our goal is to improve the space complexity of the vEB implementation from class (at the expense of losing worst-case time guarantees). Give an adapted implementation of vEB that:

- uses $O(n \log u)$ bits of space, and
- supports all operations in $O(\log \log u)$ expected amortized time.

Hint: the asymptotic bound on space is equivalent to storing $O(n)$ integers that are at most u.

Question 2 [2+5+4=12 marks]

One method of hashing with open addressing is to use quadratic probing. In the simplest form of quadratic probing, the ith element of the probe sequence is $h(k,i) = (h(k) + i^2) \bmod M$.

a) Assume that $h(k) = 0$. Give the probe sequence $\langle h(k,0), \dots, h(k,M-1) \rangle$ for $M = 11$ and for $M = 14$. No justification is needed.

b) Show that this method misses many slots of the hash table. In particular, show that the probe sequence $\langle h(k,0), \dots, h(k,M-1) \rangle$ contains at most $\lceil \frac{M+1}{2} \rceil$ many different values from $\{0, \dots, M-1\}$.

Hint: You should notice in part (a) that many indices appear twice in the probe sequence. Can you detect the pattern in which they repeat?

c) Argue that if $M \geq 3$ is prime and $\alpha \leq \frac{1}{2}$, then the probe sequence always finds an empty slot.

You are allowed to use modular arithmetic rules without proof, see Appendix B in the textbook for details.

Question 3 [2+4(+5)+2 = 8(+5) marks]

We have seen one method of obtaining a universal family of hash-functions in class. This assignment discusses another one. Let us assume that all keys come from some universe $\{0, \ldots, U-1\}$, where $U = 2^m$. Therefore any key k can be viewed as bitstring x_k of length m by taking its base-2 representation.

Let us assume further that the hash-table-size M is $M = 2^b$ for some integer b, with $b < m$. To choose a hash-function, we now randomly choose each entry in a $b \times m$-matrix H to be 0 or 1 (equally likely). Then compute $h_k = (Hx_k)\%2$, where x_k is now viewed as a vector and '%2' is applied to each entry. The output is a b-dimensional vector with entries in $\{0, 1\}$; interpreting it as a length-b bitstring gives a number $\{0, ..., M-1\}$ that we use as hash-value $h(k)$. For example, if $k = 18$, $m = 5$, $b = 3$ and H is as shown below, then $h(k) = 1$ since

$$\underbrace{\begin{pmatrix} 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix}}_{H} \underbrace{\begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}}_{\text{18 as length-5 bitstring}} \%2 = \underbrace{\begin{pmatrix} 0 \\ 2 \\ 1 \end{pmatrix}}_{Hx_k} \%2 = \underbrace{\begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}}_{\text{1 as length-3 bistring}}$$

a) Let H be the above matrix, $m = 5$ and $b = 3$. Consider the keys 9 and 13. What are their hash-values? Show your work.

b) Consider again $m = 5, b = 3$ and keys $k = 9$ and $k' = 13$. Consider the same matrix H, except that the bits in the third column are randomly chosen. What is the probability that $h(k) = h(k')$? Justify your answer.

c) (Bonus) Show that (for any m, b) this method of choosing the hash function gives a universal hash function family, or in other words, $P(h(k) = h(k')) \leq \frac{1}{M}$ for any two keys $k \neq k'$.

d) This method for obtaining universal hash-functions is much less popular than using the Carter-Wegman functions. Why do you think that that might be the case? (Expected length of answer is 1-3 sentences.)

Question 4 [2+5+5+5=17 marks]

A *skewed kd-tree* is a *kd*-tree where the splits are allowed to be less even: If a node v has n_v points in its subtree, then its sibling stores at least $n_v/2$ and at most $2n_v$ points in its subtree.

a) For $n = 7$ and $n = 9$, show a skewed *kd*-tree that stores n points and has the maximum possible height among all such trees. (You need not prove that it is maximal.) You only have to show the tree-structure; no need to show suitable points.

b) Give an upper bound on the height of a skewed *kd*-tree that stores n points. Give an exact bound (no asymptotics).

Also state what your bound evaluates to for $n = 7, 9$. For full credit, your bound must be at most one bigger than the height you achieved in part (a) for $n = 7, 9$.

c) One would *insert* in a skewed *kd*-tree as follows. First insert the point where it needs to be (by splitting at an appropriate leaf). Then check size-balances at the ancestors, and (if needed) re-build a maximal subtree where the size-balance is violated. Consider the following potential function:

$$\Phi(\cdot) := c \sum_{v \in V} \log n_v \cdot \max\{0, |n_{v.left} - n_{v.right}| - 1\}$$

where as before n_v denotes the number of points stored in the subtree rooted at v, and c is a constant. Show that during an *insert* without rebuilding, this potential function increases by $O(\log^2 n)$.

You may use without proof that $\log(x+1) \leq \log x + \frac{1}{x}$ for $x > 0$.

(Motivation: With this potential function all operations have amortized run-time $O(\log^2 n)$, but to keep the assignment from getting too long you do *not* have to show this.)

- We can do a range-search on a skewed *kd*-tree in exactly the same manner as for a *kd*-tree. While the run-time is not as good as for a *kd*-tree, it is better than for quad-trees since at least we do not visit every node. Prove this. Specifically, prove that in a skewed *kd*-tree, the number of boundary nodes in any range-search is in $O(n^c)$ for some $c < 1$.

  Any $c < 1$ will give you full credit, but we recommend $c = 0.9$.

Question 5 [5+5+2=12 marks]

A *range-counting-query* is like a range search, except that you only need to report *how many* items fall into the range, you do not need to list which items they are.

a) Describe how any balanced binary search tree can be modified such that a range counting query can be performed in $O(\log n)$ time (independent of s, the number of points in the query-interval). Briefly state the changes needed, then describe the algorithm for the range counting query.

b) Now consider the 2-dimensional-case: Describe an appropriate range-tree based data structure such that you can answer range-counting-queries among 2-dimensional points in time $O((\log n)^2)$. Then describe the algorithm for the range counting query.

c) Prof. Conn Fused thinks that they can do a similar approach for kd-trees, so report the number of points in $O(\log^2 n)$ time. Why is this not correct?

For all questions concerning points and their data structures, the points are in general position.