Computer Science CSC263H
St. George Campus

February 27, 2020
University of Toronto

Homework Assignment #5

Due: March 12, 2020, by 5:30 pm

- **You must submit your assignment through the Crowdmark system.** You will receive by email an invitation through which you can submit your work. If you havent used Crowdmark before, give yourself plenty of time to figure it out!
- You must submit a **separate** PDF document with for **each** question of the assignment.
- To work with one or two partners, you and your partner(s) must form a **group** on Crowdmark (one submission only per group). We allow groups of up to three students, submissions by groups of more than three students will not be graded.
- The PDF file that you submit for each question must be typeset (**not** handwritten) and clearly legible. To this end, we encourage you to learn and use the LaTeX typesetting system, which is designed to produce high-quality documents that contain mathematical notation. You can use other typesetting systems if you prefer, but handwritten documents are not accepted.
- If this assignment is submitted by a group of two or three students, for each assignment question the PDF file that you submit should contain:
  1. The name(s) of the student(s) who *wrote* the solution to this question, and
  2. The name(s) of the student(s) who *read* this solution to verify its clarity and correctness.
- By virtue of submitting this assignment you (and your partners, if you have any) acknowledge that you are aware of the homework collaboration policy that is stated in the csc263 course web page: `http://www.cs.toronto.edu/~sam/teaching/263/#HomeworkCollaboration`.
- For any question, you may use data structures and algorithms previously described in class, or in prerequisites of this course, without describing them. You may also use any result that we covered in class, or is in the assigned sections of the official course textbook, by referring to it.
- Unless we explicitly state otherwise, you should justify your answers. Your paper will be marked based on the correctness and completeness of your answers, and the clarity, precision, and conciseness of your presentation.
- The total length of your pdf submission should be no more than 4 pages long in a 10pt font.

Question 1. (1 marks) Consider the forest implementation of the disjoint-sets abstract data type, with an initial forest of n distinct elements (each one in a one-node tree). Let σ be any sequence of k UNIONs followed by k' FINDs; so *all* UNIONs occur before the FINDs. Prove that the algorithm using Path Compression only (it does *not* use the Weighted-Union rule) executes σ in $O(k + k')$ time, i.e., in time proportional to the length of σ, in the worst-case.

Do *not* make assumptions on k or k' (for example, do not assume that $k = n - 1$ or that $k' \leq k$). As we did in class, assume that the parameters of each UNION are two set representatives, i.e., two tree roots (so there are *no* FINDs "inside" each UNION).

HINT: Note that if a vertex becomes a child of a root during the execution of one of the FINDs (because of Path Compression), then it remains a child of this root during all the subsequent FINDs. To compute the "cost" of executing all the FINDs use an amortization-like charging scheme.

Question 2. (1 marks)
In the following questions $G = (V, E)$ is an undirected graph with $V = \{1, \ldots, n\}$, $|E| = m$, and G is stored as adjacency list L, where $L[i]$ is a linked list of vertex i's neighbours. Define the following operations:

DEGREE(G, i): Return the degree of vertex $i \in G.V$

AVERAGEDEGREE(G): Return the average degree over all vertices in $G.V$

CONTAINSEDGE(G, i, j): Given $i, j \in G.V$, return TRUE if there is an edge $(i, j) \in G.E$, FALSE otherwise

INSERTEDGE(G, i, j): Given $i, j \in G.V$ and edge $(i, j) \notin G.E$, add edge (i, j) to $G.V$

a. Assume that G is *dense*, that is $m \geq n(n-1)/4$. Consider the representation of G by the adjacency list of its "complement" graph $\overline{G} = (V, \overline{E})$, where $\overline{E} = \{(i, j) : i, j \in V \wedge (i, j) \notin E\}$.

Contrast the worst-case space complexity of representing G by its adjacency list L, versus representing G by $\overline{L}$, the adjacency list for $\overline{G}$. Explain your reasoning and express your answer in terms of n and m.

Contrast an upper bound on the worst-case time complexity of carrying out the operations above on G when it is given as the adjacency list $\overline{L}$ versus adjacency list L? Explain your reasoning in each case, and express your answer in terms of n and m.

b. Suppose we modify G's adjacency list L with L', where the list $L[i]$ of vertices adjacent to i is replaced by AVL tree $L[i]'$, where the keys are the vertices adjacent to i. What is the space complexity of this approach? Contrast the worst-case time complexity of this approach for the operations above versus using adjacency list L. Explain your reasoning and express your answer in terms of n and m.

Question 3. (1 marks)

Dictionary D contains n lower-case k-character strings from our Latin alphabet as keys, with satellite values COLOUR and PARENT that you are free to modify.

You may assume that SEARCH(D, t) returns a pointer to key t if $t \in D$, or NIL if $t \notin D$, in worst-case $O(\log n)$ time. You may also assume that it takes $O(n)$ time to iterate over all pointers to D's keys in some arbitrary order.

You are given **start** and **target** k-character strings from dictionary D. For example you might be given **start** = "ape" and **target** = "man". Your goal is to find a shortest sequence of k-character strings from **start** to **target**, where adjacent strings differ in exactly one of the k positions, and each string is dictionary D.

a. Use pseudocode to describe an algorithm to find a shortest sequence from **start** to **target**, or return "No sequence exists!" otherwise. You must use no more than $O(nk)$ space in addition to dictionary D.

b. What is an upper bound, as a function of both n and k, on the worst-case time complexity of your algorithm? Explain.

[***The questions below will not be corrected/graded. They are given here as interesting problems that use material that you learned in class.***]

Question 4. (0 marks) We want a data structure that maintains a set I of integers and supports the following two operations:

1. INSERT(x), insert a new integer x into I (assume that x is not already in I).
2. SEARCH(x), which returns TRUE if integer x is currently in I, and returns FALSE otherwise.

If we keep I in a sorted array A, then, using binary search, the worst-case time for a SEARCH(x) is $O(\log n)$, where n is the size of I. But the worst-case time for an INSERT(x) is $O(n)$, and the *amortized* insertion time (i.e., the worst-case total time to execute n INSERTs divided by n) is also $O(n)$. Our goal is to decrease the *amortized* insertion time, may be at the cost of increasing the worst-case time of SEARCH.

To do so, we keep I in a *doubly linked list L of sorted arrays* as follows. Let n be the number of elements in I, and $< b_{k-1}, b_{k-2}, \ldots, b_0 >$ be the binary representation of n; note that $\Sigma_{i=0}^{i=k-1} b_i 2^i = n$ and $k = O(\log_2 n)$. For every $i = 0, 1, \ldots, k-1$ such that $b_i = 1$, the doubly-linked list L contains a sorted array A_i of size 2^i; A_i stores 2^i elements of I in increasing sorted order. The arrays of L are listed in order of increasing size. Each integer of I is in exactly one of the sorted arrays of L. Note that although each array of L is sorted, there is no particular relationship between the elements in different arrays of L.

a. Draw two instances of the data structure L, one for set $I = \{6, 8, 4, 13, 9\}$ and one for set $I = \{21, 12, 7, 14, 5, 16, 10\}$.

In the questions below, you should give as efficient algorithms as possible. Describe your algorithms in clear and concise English, there is no need to use pseudocode.

b. Describe an algorithm to perform a SEARCH(x) operation with this data structure. Give a good upper bound on the worst-case time complexity of your algorithm (using the O notation) and justify your answer.

c. Describe an algorithm to perform an INSERT(x) operation with this data structure. Give a good upper bound on the worst-case time complexity of your algorithm (using the O notation) and justify your answer.

HINT: Think about how we insert an element in a Binomial Heap, and use the Merge part of MergeSort.

d. Suppose we execute a sequence of n INSERTs starting from an empty set I. Determine a good upper bound on the *amortized* time of an INSERT (i.e., the worst-case total time to execute these n INSERTs divided by n). Justify your answer in two different ways, i.e., give two separate proofs, each proof using a different argument (e.g., use aggregate analysis and the accounting method).

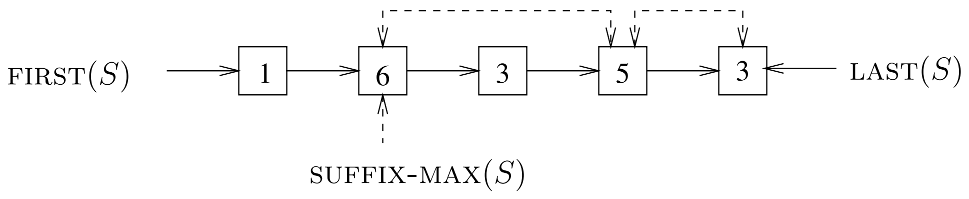
e. Describe an algorithm to perform a DELETE(x) operation, i.e., given a pointer to integer x in one of the arrays of L, remove x, in $O(n)$ time in the worst case. Explain why the worst-case time complexity of your algorithm is $O(n)$.

Question 5. (0 marks) Consider the abstract data type MAXQUEUE that maintains a sequence S of integers and supports the following three operations:

- DEQUEUE(S): removes the first element of S and returns its value.
- ENQUEUE(S, x): appends the integer x to the end of S.
- MAXIMUM(S): returns the largest integer in S, but does not delete it.

An element x is a *suffix maximum* in a sequence if all elements that occur after x in the sequence are strictly smaller in value. For example, in the sequence 1,6,3,5,3, the suffix maxima are the second, fourth, and fifth elements.

One way to implement the MAXQUEUE abstract data type is to use a singly linked list to represent the sequence S, with additional pointers FIRST(S) and LAST(S) to the first and last elements of that list; and to have a doubly linked list of the suffix maxima (arranged in the same order as they are in the sequence), with an additional pointer SUFFIX-MAX(S) to the first element of that list. For example, the sequence $S = 1, 6, 3, 5, 3$ would be represented as follows:



a. Describe algorithms to implement each of the three operations (namely, DEQUEUE(S), ENQUEUE(S, x), and MAXIMUM(S)) for the MAXQUEUE abstract data type using the above representation. *For each operation, first explain the basic idea of how your algorithm works in a few clear English sentences*; you can then give more details using English (and pseudo-code if necessary).

Your algorithms should be such that the operations have amortised complexity $O(1)$, assuming that initially the MAXQUEUE contains the empty sequence.

b. Prove that your algorithms achieve the desired amortised complexity. *Hint:* Use the accounting method.

c. Determine an asymptotic tight bound for the worst-case cost of an *individual operation* in a sequence of m operations. Justify your bound.

d. Find an alternative implementation of this abstract data type such that the worst-case time needed for an individual operation is $O(\log n)$, where n is the number of elements in the sequence. The amortised complexity in this case need not be $O(1)$.