CSC265 F18: Assignment 6

Due: December 6, by midnight

Guidelines: (read fully!!)

- Your assignment solution must be submitted as a *typed* PDF document. Scanned handwritten solutions, solutions in any other format, or unreadable solutions will **not** be accepted or marked. You are encouraged to learn the LaTeX typesetting system and use it to type your solution. See the course website for LaTeX resources. Solutions typed using LaTeX receive 2 bonus marks.
- Your submission should be no more than 6 pages long, in a single-column US Letter or A4 page format, using at least 9 pt font and 1 inch margins.
- To submit this assignment, use the MarkUs system, at URL `https://markus.teach.cs.toronto.edu/csc265-2018-09`
- This is a *group assignment*. This means that you can work on this assignment with *at most one other* student. You are *strongly encouraged* to work with a partner. Both partners in the group should work on and arrive at the solution together. Both partners receive the same mark on this assignment.
- Work on all problems together. For each problem, one of you should write the solution, and one should proof-read and revise it. The first page of your submission must list the *name*, *student ID*, and *UTOR email address* of both group members. It should also list, for each problem, which group member wrote the problem, and which group member proof-read and revised it.
- You **may not** consult any other resources except: your partner; your class notes; your textbook and assigned readings. *Consulting any other resource, or collaborating with students other than your group partner, is a violation of the academic integrity policy!*
- You may use any data structure, algorithm, or theorem previously studied in class, or in one of the prerequisites of this course, by just referring to it, and without describing it. This includes any data structure, algorithm, or theorem we covered in lecture, in a tutorial, or in any of the assigned readings. Be sure to give a *precise reference* for the data structure/algorithm/result you are using.
- Unless stated otherwise, you should justify all your answers using rigorous arguments. Your solution will be marked based both on its completeness and correctness, and also on the clarity and precision of your explanation.

Question 1. (14 marks)

Consider an ADT which maintains a collection of disjoint subsets of the set $[n] = \{1, \ldots, n\}$, and supports the operations

- SPLIT(i) splits the set S containing i into two sets: $S_{\leq} = \{j \in S : j \leq i\}$ and $S_{>} = \{j \in S : j > i\}$;
- FIND-MIN(i) returns the minimum element of the set S containing i.

Suppose that you are given as input a sequence $\sigma = (\sigma_1, \ldots, \sigma_m)$ of operations, where each σ_t is equal to either FIND-MIN(i) or SPLIT(i) for some $i \in [n]$. Assume that before σ_1, we start with a single set $S = [n]$. Assume further that, for each $i \in [n-1]$, SPLIT(i) is called at most once in σ, and that it's never called for $i = n$. Design an algorithm that, given σ as input, computes the value that must be returned by the ADT on each call of FIND-MIN(i). More precisely, your algorithm must return an array $R[1\,..\,m]$, so that if $\sigma_t =$ FIND-MIN(i), then $R[t]$ is the value returned by this instance of FIND-MIN(i), and if $\sigma_t =$ SPLIT(i), then $R[t] =$ NIL. Your algorithm must run in worst-case time complexity $O((n+m)\log^* n)$.

As an example, let $n = 5$, and assume that σ is the sequence

$$\text{FIND-MIN}(3), \text{SPLIT}(2), \text{FIND-MIN}(3), \text{FIND-MIN}(1), \text{SPLIT}(3), \text{FIND-MIN}(4).$$

Then you must return $R = [1, \text{NIL}, 3, 1, \text{NIL}, 4]$. After the first split, the sets maintained by the ADT are $\{1, 2\}, \{3, 4, 5\}$, and after the second split, they are $\{1, 2\}, \{3\}, \{4, 5\}$.

Describe your algorithm in clear and precise English, and justify why it runs in the required worst-case time complexity and why it correctly computes the array R. Be precise about what data structures you use.

Remark: *Your do **not** need to implement a data structure for the above ADT. You can use the fact that the entire sequence σ is given to your algorithm.*

Question 2. (14 marks)

Let $G = (V, E)$ be an undirected connected graph on the nodes $V = \{1, \ldots, n\}$, with edges $E = \{e_1, \ldots, e_m\}$, and for any $i \in \{1, \ldots, m\}$ define the graph $G_i = (V, E_i)$, $E_i = \{e_1, \ldots, e_i\}$. Moreover, G_0 is the totally disconnected graph $G_0 = (V, \emptyset)$. There exists a unique $i^* \in \{1, \ldots, m\}$ such that G_{i^*} is connected, and G_{i^*-1} has two connected components.

Assume the edges $e_1, \ldots, e_m$ are given as an array $A[1\,..\,m]$, where $A[i]$ contains the tuple of vertices $e_i = (u, v)$, $u, v \in \{1, \ldots n\}$. Give an algorithm that, when given as input the integer n, and the array A, outputs the i^* defined above. The algorithm should run in worst-case time complexity $O(m)$.

Describe your algorithm in clear and precise English, and justify why it runs in the required worst-case time complexity and why it correctly computes i^*.

HINT: Do a binary search for i^* and use some graph search.