Computer Science CSC263H
St. George Campus

March 12, 2020
University of Toronto

Homework Assignment #6

Due: March 26, 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 haven't 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 *directed* graph G with nodes $\{1, 2, 3, 4, 5, 6, 7\}$ that is represented by its adjacency lists:

$A(1) = 4, 7, 5$
$A(2) = \text{NIL}$
$A(3) = 2, 1, 5$
$A(4) = 7, 2$
$A(5) = 2$
$A(6) = 3, 5$
$A(7) = 2$

a. Draw a Depth-First Search forest generated by the DFS of G under the following assumptions: *the DFS starts at node 1 and it explores the edges in the order of appearance in the above adjacency lists (for example, edge* $(4, 7)$ *is explored before edge* $(4, 2)$*), and all the vertices are explored.* Do not draw forward, back, or cross edges. Show the discovery and finishing times $d[u]$ and $f[u]$ of every node u of G, as computed by this DFS of G.

b. How many back edges, forward edges, and cross-edges are found by the above DFS?

c. Suppose the graph G above represents seven courses and their prerequisites. For example, $A(1) = 4, 7, 5$ means that course 1 is a prerequisite for (i.e., it must be taken before) courses 4, 7 and 5; and $A(2) = \text{NIL}$ means that course 2 is not a prerequisite for any course.

Using Part (b) above and a theorem that we learned in class, prove that it is possible to take all the courses in a sequential order that satisfies all the prerequisite requirements. State the theorem that you use in your argument; do ***not*** give a specific course order here.

d. Now list the courses in an order they can be taken without violating any prerequisite. To do so you ***must*** use your DFS of Part (a) and an algorithm that we covered in a tutorial.

e. Draw a Breadth-First Search tree of G that ***starts at node*** **6** and explores the edges in the order of appearance in the above adjacency lists.

Question 2. (1 marks) **Prove or disprove:**

1. Consider an undirected, connected graph $G = (V, E)$ with distinct integer weights on each edge in E, and let T_1 be a MST for G. If T_3 is a spanning tree for G with $w(T_3) = w(T_1) + 1$ and T_4 is a spanning tree for G with $w(T_4) = w(T_3)$, then $T_3 = T_4$.

2. If $G = (V, E)$ is a connected graph with all positive edge weights, $E' \subseteq E$, and $G' = (V, E')$ is a spanning subgraph of G of minimum weight (over all spanning subgraphs of G), then G' is an MST of G.

3. If $G = (V, E)$ is a connected graph with edge weights that are positive, negative, or 0, $E' \subseteq E$, and $G' = (V, E')$ is a spanning subgraph of G of minimum weight (over all spanning subgraphs of G), then G' is an MST of G.

Question 3. (1 marks) L is a list of integers representing a set of navigable (by canoe) lakes. P is an array of length $|L|$, where for each $u \in L$, $P[u]$ is a linked list containing pairs (v, x) where v represents a lake in L having a portage (footpath) between u and v, and x is the (positive) length of the portage between lakes u and v.

Each portage has a distinct length, and every pair of lakes has, at most, one portage between them. It is also possible to make a trip between any pair of lakes using one or more portages. Each trip between a pair of lakes has a *toughness*: the length of the longest portage on the trip (i.e., longest footpath of the trip where canoeists must haul canoes and gear on their backs). Each pair (i, j) of lakes has a *rating*, which is defined as the minimum toughness of any trip between lakes i and j. Canoeists want to determine for each pair of lakes (i, j), the rating of (i, j) and a trip with this rating between these two lakes. The next three questions show how to do it efficiently.

a. Explain, in precise English, how to represent this set-up as a weighted undirected graph $G = (V, E)$.

b. Prove that if $G = (V, E)$ contains a cycle C and $e = (u, v)$ is the maximum weight edge in C, then the rating of all pairs of lakes is the same in G and $G' = (V, E - e)$.

c. Let T be any MST for G. Prove that for each pair of lakes (i, j), the rating of (i, j) in G is also the rating of (i, j) in T. Note that this implies that the rating of (i, j) in G is the toughness of the unique (!) trip between lakes i and j in the MST T of G. In other words, the MST T of G gives the best trip (the one with the smallest toughness) between any pair of lakes in G.

[***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) Let G be a digraph (i.e., a directed graph). A node u of G is called a *source* if there is no edge leading into u (i.e., u is not adjacent to any node).

Consider the following *source-deletion* operation, applied to a digraph that has at least one source: Arbitrarily choose a source u; remove from the graph the node u and all edges incident from u.

Suppose we apply this operation repeatedly, until it can no longer be applied. There are two reasons why we may be unable to continue the process of applying the operation: either the remaining digraph has no sources, or it has no nodes at all!

a. Prove that if a non-empty digraph is acyclic, then it has at least one source.

b. Prove that a digraph is acyclic if and only if the repeated application of the source-deletion operation results in the empty graph. (Hint: Use part (a) for the only-if direction.)

c. Based on the characterisation of acyclicity shown in (b), you should develop an algorithm to determine whether a given digraph is acyclic, as described below. Your algorithm should run in $O(n+m)$ time, where n is the number of nodes and m is the number of edges of the input graph. You should assume the graph is input in the usual way, using adjacency lists.

Your algorithm should begin by computing the *in-degree* of each node. The in-degree of a node v is defined to be the number of nodes u such that (u, v) is an edge. Your algorithm should compute, for each node v, the value $I(v)$ of the in-degree of v, as well as the set S of nodes of in-degree 0. It should then, for each node in S do the following: remove it from S, and effectively remove the node from the graph by changing the array I appropriately, inserting new elements into S as appropriate.

You should describe your algorithm first in clear English, and then also give the pseudo-code. Explain why your algorithm works, and why it has the stated time complexity.

Question 5. (0 marks) Let $G = (V, E)$ be an undirected, connected graph, and $w : E \to \mathbb{R}$ be an edge weight function such that edges have *distinct* weights. Suppose that for every edge $e \in E$ there is a cycle of G that contains e. Let e_{max} be the edge with maximum weight in G. Prove that no minimum spanning tree of G contains e_{max}.

Question 6. (0 marks) Let G be an arbitrary undirected connected graph where each edge has a weight, and T be a minimum spanning tree of G.

a. Suppose we augment G by adding a new node v and new weighted edges connecting v to some nodes of G. Let G' be the resulting (weighted) graph. Prove or find a counterexample to the following statement: We can always obtain a minimum spanning tree T' of G' by adding to T an edge of minimum weight among all the new edges (those connecting v and the nodes of G).

b. Suppose we augment G by adding a new edge e of weight w between two nodes of G. Let G' be the resulting (weighted) graph. Describe an algorithm that builds a minimum spanning tree of G' in $O(n)$-time, where n is the number of nodes in G. Assume that each of G and its minimum spanning tree T, is given in the adjacency-list representation. Your algorithm's english description should be clear and brief (do not use pseudo-code). Prove the correctness of your algorithm and explain why its worst-case running time is $O(n)$.