Dept. of Computer Science **University of Toronto**

Assignment # 2 (Due July 15, 11:59 pm)

CSC 373H1 **Summer 2017**

Worth: 5%

1. **[20 marks]**

   (a) **[10 marks]** Give an efficient algorithm that takes the following inputs:

   - $G = (V, E)$, a connected undirected graph,
   - $w : E \to \mathbb{Z}^+$, a weight function for the edges of G,
   - $T \subseteq E$, a minimum spanning tree of G,
   - $e_1 = \{u, v\} \notin E$ (for $u, v \in V$), an edge not in G,
   - $w_1 \in \mathbb{Z}^+$, a weight for e_1,

   and that outputs a minimum spanning tree T_1 for the graph $G_1 = (V, E \cup \{e_1\})$ with $w(e_1) = w_1$.

   For full marks, your algorithm must be more efficient than computing a MST for G_1 from scratch. Justify that this is the case by analysing your algorithm's worst-case running time.

   Finally, write a detailed proof that your algorithm is correct. (Note that this argument of correctness will be worth at least as much as the algorithm itself.)

   (b) **[10 marks]** Give an efficient algorithm that takes the following inputs:

   - $G = (V, E)$, a connected undirected graph,
   - $w : E \to \mathbb{Z}^+$, a weight function for the edges of G,
   - $T \subseteq E$, a minimum spanning tree of G,
   - $e_0 \in E$, an edge in G,

   and that outputs a minimum spanning tree T_0 for the graph $G_0 = (V, E - \{e_0\})$, if G_0 is still connected — your algorithm should output the special value Nil if G_0 is disconnected.

   For full marks, your algorithm must be more efficient than computing a MST for G_0 from scratch. Justify that this is the case by analysing your algorithm's worst-case running time.

   Finally, write a detailed proof that your algorithm is correct. (Note that this argument of correctness will be worth at least as much as the algorithm itself.)

2. **[20 marks]**
   Consider the following "MST with Fixed Leaves" problem:

   Input: A weighted graph $G = (V, E)$ with integer costs $c(e)$ for all edged $e \in E$, and a subset of vertices $L \subseteq V$.

   Output: A spanning tree T of G where every node of L is a leaf in T and T has the minimum total cost among all such spanning trees.

   (a) **[3 marks]** Does this problem always have a solution? In other words, are there inputs G, L for which there is no spanning tree T that satisfies the requirements?

   Either provide a counter-example (along with an explanation of why it is a counter-example), or give a detailed argument that there is always some solution.

   (b) **[3 marks]** Let G, L be an input for the MST with Fixed Leaves problem for which there *is* a solution.

   Is every MST of G an optimal solution to the MST with Fixed Leaves problem? Justify.

   Is every optimal solution to the MST with Fixed Leaves problem necessarily a MST of G (if we remove the constraint that every node of L must be a leaf)? Justify.

(c) **[7 marks]** Write a greedy algorithm to solve the MST with Fixed Leaves problem. Give a detailed pseudo-code implementation of your algorithm, as well as a high-level English description of the main steps in your algorithm.

What is the worst-case running time of your algorithm? Justify briefly.

(d) **[7 marks]** Write a detailed proof that your algorithm always produces an optimal solution.

3. **[20 marks]**
An edge in a flow network is called *critical* if decreasing the capacity of this edge reduces the maximum possible flow in the network.

Give an efficient algorithm that finds a critical edge in a network. Give a rigorous argument that your algorithm is correct and analyse its running time.

4. **[20 marks]**

(a) **[8 marks]**
Suppose we want to compute the shortest path from node s to node t in a *directed* graph $G = (V, E)$ with edge lengths $\ell_e > 0$ for each $e \in E$.
Show that this is equivalent to finding a *pseudo-flow* f from s to t in G such that $|f| = 1$ and $\sum_{e \in E} \ell_e f(e)$ is minimized. There are no capacity constraints.
Part of this problem requires you to **define** precisely what we mean by "pseudo-flow" in a general, directed graph. This is a natural extension of the notion of flow in a network.

(b) **[12 marks]**
Write the shortest path problem as a linear or integer program **where your objective function is *minimized*, based on your answer to the previous part**. Give a detailed justification that your solution is correct.

5. **[20 marks]**

(a) **[10 marks]**
In the Traveling Salesman Problem (TSP), we are given a directed graph $G = (V, E)$ with an integer weight $w(e)$ for each edge $e \in E$, and we are asked to find a simple cycle over all the vertices (a "circuit") with minimum total weight. (Note that the weights $w(e)$ can be positive or negative.)
Show how to represent an arbitrary instance of the TSP as an integer program. Justify that your representation is correct, and describe how to obtain a solution to the instance of the TSP from solutions to your integer program.

(b) **[10 marks]**
Your company has recently discovered that several of its problems can be solved using linear programming. Your company doesn't want to write their own solver program, since many efficient programs are available for sale. But your boss has been reading his spam again and went out and purchased the MELPSE system (Most Efficient Linear Program Solver Ever!) in a fit of misdirected leadership.
As expected, the claim is slightly overstated, and the package comes with some serious limitations. From the advertisement:

> MELPSE is the fastest and most streamlined LP solver ever! Using the latest technology, it will find the best non-negative values for all your variables, get the biggest value for your objective function, and it will even make sure that $\mathbf{A}x \le b$!

In fact, this is all that MELPSE can do: it only supports non-negative variables (it implicitly enforces a constraint $x_i \geq 0$ on all variables x_i), only allows maximizations of the linear objective function, and insists that all constraints are in the form $\sum_{i=1}^{n} a_{j,i} x_i \leq b_j$ (where $a_{j,i}$ and b_j are real numbers, perhaps negative).

The linear programs you need to solve are usually minimization problems, where variables occasionally take negative values, and some constraints are expressed with equality or greater-than-or-equal. The boss has already spent your entire budget, so to save your team you will need to figure out how to use the MELPSE system to solve your problems.

i. One of your problems fits the restrictions of MELPSE except that you need to *minimize* your objective function. Describe precisely how to convert your program into one MELPSE can solve; that is, describe how to create an equivalent LP where the objective is being *maximized*. Briefly explain how to use a solution to your new program to find a solution to your original problem.

ii. Another of your problems contains a constaint of the form

$$a_1 x_1 + a_2 x_2 + \cdots + a_n x_n \geq b.$$

Describe precisely how to convert this to a constraint of the form

$$a'_1 x_1 + a'_2 x_2 + \cdots + a'_n x_n \leq b'$$

as required by MELPSE.

iii. This problem also has a constraint of the form

$$a_1 x_1 + a_2 x_2 + \cdots + a_n x_n = b.$$

Describe precisely how to create an equivalent LP that can be solved by MELPSE (or in the form of part (b)).

iv. From the previous parts we know how to solve a minimization problem with $\geq$ or $=$ constraints. We can assume that all constraints are in the form $a_1 x_1 + a_2 x_2 + \cdots + a_n x_n \leq b$. But we have a problem where one of the variables, x_1, could be negative. Describe precisely how to construct an equivalent LP where we replace x_1 with two new variables which can only take non-negative values. Briefly explain how to use a solution to your new program to find a solution to your original problem.

v. Write an efficient high-level algorithm that will convert a linear program that might be a maximization problem, might contain greater-than-or-equal or equality constraints, and may contain variables lacking a non-negativity constraint, into an equivalent linear program that can be solved by MELPSE. Give a good bound on the size (the size being the number of variables and number of constraints) of your new linear program and briefly justify it.