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

Assignment # 2 (Due July 14, 11:59 pm)

CSC 373H1 **Summer 2018**

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.)

Solution. (a) **Algorithm:**

```
AddMST($G, w, T, e_1, w_1$):
    Use BFS on $T$ to find the unique path $P$ from $u$ to $v$ in $T$ (where $\{u,v\} = e_1$).
    Let $e_0$ be an edge on $P$ with maximum weight.
    if $w(e_0) > w_1$:
        return $T - \{e_0\} \cup \{e_1\}$
    else:
        return $T$
```

Runtime: BFS takes time $\Theta(|V| + |E|)$; finding e_0 takes time $\mathcal{O}(|E|)$; total time is $\Theta(|V| + |E|)$.

Correctness: Because edge e_1 is the only difference between G and G_1, it is the only edge whose addition may result in a different MST from T. This happens only when e_1 can be swapped with some edge in T with higher weight: exactly what the algorithm does.

(b) **Algorithm:**

```
DelMST($G, w, T, e_0$):
    if $e_0 \notin T$:
        return $T$
    else:
        Let $e_0 = \{u,v\}$.
```

```
Run BFS on the edges of $T - \{e_0\}$, starting from $u$;
    assign colour *white* to every vertex encountered.
Run BFS on the edges of $T - \{e_0\}$, starting from $v$;
    assign colour *black* to every vertex encountered.
Loop over every edge in $E - \{e_0\}$ to find a minimum-weight edge $e_1$
    with one *white* endpoint and one *black* endpoint.
if there is no such edge $e_1$:
    return NIL
else:
    return $T - \{e_0\} \cup \{e_1\}$
```

Runtime: BFS takes time $\Theta(|V| + |E|)$; finding e_1 takes time $\mathcal{O}(|E|)$; total time is $\Theta(|V| + |E|)$.

Correctness: Because edge e_0 is the only difference between G and G_1, it is the only edge whose removal may result in a different MST from T. From the proof of correctness of Kruskal's algorithm, we know that it is always "safe" to add an edge of minimum weight between two connected components (while constructing a MST): exactly what the algorithm does.

□

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.

Solution. (a) Counter-example: $G = (V, E)$ with $V = \{a, b, c\}$, $E = \{(a, b), (b, c)\}$, $c(a, b) = c(b, c) = 1$, and $L = \{b\}$. Since G is already a tree, there is only one spanning tree of G (G itself), and b is not a leaf in this tree. Thus, the MST with Fixed Leaves problem does <u>not</u> have a solution for this given G and L.

(b) Consider the graph $G = (V, E)$ with $V = \{a, b, c\}$, $E = \{(a, b), (b, c), (a, c)\}$, $c(a, b) = c(b, c) = 1$, $c(a, c) = 2$, and $L = \{b\}$. Then G contains exactly one MST: $T = \{(a, b), (b, c)\}$, but T is not a solution to the MST with Fixed Leaves problem because b is not a leaf in T.

With the same input, there are two optimal solutions to the MST with Fixed Leaves problem: $T_1 = \{(a, c), (a, b)\}$ and $T_2 = \{(a, c), (b, c)\}$. Neither of these is a MST in G.

(c)
```
# This is a variation of Kruskal's algorithm: we construct the tree edge-by-edge,
# starting with the nodes in $L$.
# Handle the only special case when two nodes of $L$ must be connected to each other.
**if** $V = \{a, b\}, E = \{(a, b)\}, L = \{a, b\}$:    **return** $\{(a, b)\}$
# Now, handle the general case.
$T := \emptyset$
# Start by selecting one edge $(v, u)$ for each node $v \in L$, where $u \notin L$ and $c(v, u)$ is minimum.
**for** $v \in L$:
    $c := \infty$
    **for** $(v, u) \in E$:
        # Remove all edges adjacent to $v$ from $E$, to ensure $v$ is a leaf in $T$.
        $E := E - \{(v, u)\}$
        **if** $u \notin L$ **and** $c(v, u) < c$:
            $e := (v, u)$
            $c := c(v, u)$
    # At this point, $e$ is a minimum-cost edge connecting $v$ to $V - L$.
    $T := T \cup \{e\}$
Now, run Kruskal's algorithm on the remaining graph, starting from the edges already in $T$.
**return** $T$
```

The algorithm's complexity is $\mathcal{O}(|E| \log |E|)$, same as Kruskal's, since that is the part of the algorithm that takes the longest.

(d) Let $G_0 = G - L$ (G_0 is G with every node of L removed, and every edge that contains a node from L removed).

Consider any optimal solution T to the MST with Fixed Leaves problem on input G, L. Now consider $T' = T - \{(v, u) : v \in L\}$ (T' is T with its leaves removed).

Claim: T' is a MST in G_0.

Proof: For a contradiction, suppose T^* is a spanning tree of G_0 and $c(T^*) < c(T')$. Then $T^* \cup \{(v, u) \in T : v \in L\}$ forms a spanning tree of the original graph G whose total cost is smaller than that of T and where each node of L is a leaf. This contradicts the fact that T is an optimal solution for the original input.

Claim: The spanning tree generated by our algorithm is an optimal solution to the MST with Fixed Leaves problem on input G, L.

Proof: In every optimal solution, each node of L must be connected to $V - L$ by exactly one edge. Any choice other than a minimum-cost edge would increase the cost of the resulting spanning tree and would be sub-optimal. Also, connecting one node of L to another node of L would make it impossible for both nodes to be connected to the rest of the graph and still be leaves (unless G consists of exactly one edge).

Once these edges have been selected, and other edges to the vertices of L eliminated (to guarantee every node of L is a leaf), the remaining graph is simply G_0 and Kruskal's algorithm is guaranteed to find a MST of G_0. Thus, there is no spanning tree of G where each node of L is a leaf and with a smaller cost.

□

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.

Solution. **Observation:** First we note that if (S,T) is a minimum cut in a flow network G then any edge $e=(u,v)$ with $u \in S$ and $v \in T$ is critical. This is because for every cut (S',T') in a flow network and for every flow f in the network, we have $|f| \le c(S',T')$, where $|f|$ is the value of the flow and $c(S',T')$ is the capacity of the cut. By the Max-Flow Min-Cut Theorem, if f^* is a maximum flow and and (S,T) is a minimum cut, then $|f^*| = c(S,T)$. Since decreasing the capacity of any edge e crossing the cut reduces the capacity of the cut, it follows that no flow of value $|f^*|$ or more is possible in the network G modified by reducing the capacity of e.

So it suffices to give an algorithm that, given a flow network $G=(V,E)$, finds an edge e crossing some minimum cut (S,T) in G.

Algorithm: (a) Compute a maximum flow f^* in G.
(b) Compute the residual graph G_{f^*}.
(c) Compute the minimum cut (S,T) constructed as part of the proof of Theorem 26.6 (Max-Flow Min-Cut Theorem), namely $S := \{v \in V \mid \text{there exists a path from } s \text{ to } v \text{ in } G_{f^*}\}$ and $T := V \setminus S$.
(d) Output any edge $e_{crit} = (u,v)$ with $u \in S$ and $v \in T$.

Correctness: To prove the correctness of the algorithm it suffices to show that an edge e_{crit} in the last step exists.

Note that every cut (S,T) has at least one edge crossing it, since by definition $s \in S$ and $t \in T$, and for every node $u \in V$ there is a path in G from s to t which includes u. Now follow this path starting with s (which is in S) until it reaches an edge (u,v) with $u \in S$ and $v \in T$.

This completes the proof that the algorithm is correct. It remains to explain how the four steps in the algorithm are implemented, and to estimate their run times.

Running Time: For steps (a) and (b) we use the Edmonds-Karp algorithm which runs in time $O(|V||E|^2)$ (p. 730 in the text).

For step (c), we use breadth first search in the residual graph G_{f^*} to make a Boolean array showing which nodes are reachable from s by paths in E_{f^*}. This defines the set S for the minimum cut and can be done in time $O(|E_{f^*}|) = O(|E|)$.

Now we find the edge e_{crit} using the following algorithm:

```
for each edge $e = (u,v) \in E$:
    if $u \in S$ and $v \notin S$:
        return $e$
```

This takes time $O(|E|)$ (assuming E is given by adjacency lists).

Hence the entire algorithm runs in time $O(|V||E|^2)$.

□

4. **[20 marks]**

   (a) **[8 marks]**

   Suppose we want to compute a shortest path from node s to node t in a *directed* graph $G = (V,E)$ with integer edge weights $\ell_e > 0$ 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.

Solution. (a) For any directed graph $G = (V,E)$ with integer edge weights $\ell_e > 0$ (for $e \in E$) and any two vertices $s \neq t \in V$, a *pseudo-flow from s to t* is a function $f : E \to \mathbb{N}$ such that for every vertex $v \in V - \{s,t\}$, $f^{\text{in}}(v) = f^{\text{out}}(v)$ (where $f^{\text{in}}(v)$ and $f^{\text{out}}(v)$ are defined as for "normal" flows in networks). For any pseudo-flow f from s to t in a directed graph G, the *value* of f is defined as: $|f| = f^{\text{out}}(s)$.

Every pseudo-flow f from s to t with $|f| = 1$ corresponds to a path in G from s to t: $|f| = 1$ means $f^{\text{out}}(s) = 1$, which means there is exactly one edge (s,v_1) with $f(s,v_1) = 1$ and $f(s,u) = 0$ for all other edges (s,u) (where $u \neq v_1$). The same reasoning applied inductively shows that there is some sequence of vertices $v_1, v_2, \ldots, v_k$ such that $f(s,v_1) = f(v_1,v_2) = \cdots = f(v_k,t) = 1$ but $f(e) = 0$ for every other edge e.

Similarly, every path $s, v_1, \ldots, v_k, t$ in G corresponds to a pseudo-flow f from s to t with $|f| = 1$ by setting $f(e) = 1$ for every edge e on the path and $f(e) = 0$ for every edge e not on the path.

Hence, finding a shortest path in G from s to t is equivalent to finding a pseudo-flow f with $|f| = 1$ such that $\sum_{e\in E} \ell_e f(e)$ is minimized.

(b) **Variables:** $x_{u,v} \in \mathbb{Z}$ for every edge $(u,v) \in E$ ($x_{u,v}$ is meant to be equal to $f(u,v)$ for some pseudo-flow f from s to t).

Objective Function: minimize $\sum_{(u,v)\in E} \ell_{u,v} x_{u,v}$ (by the answer to the previous part, this is equivalent to finding a shortest path).

Constraints:

- $x_{u,v} \geq 0$ for all $(u,v) \in E$ (pseudo-flow values cannot be negative);
- $\sum_{(u,v)\in E} x_{u,v} = \sum_{(v,u)\in E} x_{v,u}$ for all $v \in V - \{s,t\}$ (pseudo-flow is conserved);
- $\sum_{(s,v)\in E} x_{s,v} = 1$ (looking for $|f| = 1$).

By the reasoning from the previous part, we already know that paths from s to t in G are equivalent to pseudo-flows f from s to t with $|f| = 1$.

Feasible solutions to the integer program are also equivalent to pseudo-flows f from s to t with $|f| = 1$: the linear constraints correspond exactly to the properties satisfied by integer pseudo-flow values through the correspondence $x_{u,v} = f(u,v)$.

Hence, solutions to the integer program are equivalent to paths from s to t where the value of the objective function is equal to the total weight of the path. Any solution that minimizes the objective function is therefore equivalent to finding a shortest path.

□

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 $\mathbf{A}x \leq b$!

   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 $a_1x_1 + a_2x_2 + \cdots + a_nx_n \leq b$. 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 minimization 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.

Solution. (a) The following integer program represents an arbitrary instance of TSP.

Variables: one variable x_e for each edge $e \in E$

Objective function: minimize $\sum_{e \in E} w(e) \cdot x_e$

Constraints:

- $0 \leq x_e \leq 1$ for each x_e
- $\sum_{(u,v) \in E} x_{(u,v)} = 1$ for each vertex $v \in V$
- $\sum_{(v,u) \in E} x_{(v,u)} = 1$ for each vertex $v \in V$
- $\sum_{u \in V_1, v \in V_2, (u,v) \in E} x_{(u,v)} \geq 2$ for each nontrivial partition of vertices into V_1, V_2 (nontrivial means V_1 and V_2 are both non-empty)

The first constraint ensures that each edge e is either selected ($x_e = 1$) or not ($x_e = 0$)—remember that this is an *integer* program so variables cannot take on fractional values.

The second and third constraints ensure that each vertex has exactly one edge going in and one going out, which is necessary for having a simple cycle through every vertex.

The last constraint ensures that the edges selected cannot form a collection of disjoint cycles (a possibility not ruled out by the second and third constraints), by requiring that there are at least 2 edges crossing every possible nontrivial partition of the vertices.

Any solution $x^*_{e_1}, \ldots, x^*_{e_m}$ to the integer program above yields a circuit (because of the constraints) with minimum total weight (by our choice of objective function).

(b) i. Replace "minimize $c_1x_1 + \cdots + c_nx_n$" with "maximize $-c_1x_1 - \cdots - c_nx_n$". Solutions remain exactly the same.

ii. Let $a'_1 = -a_1, \ldots, a'_n = -a_n, b' = -b$, *i.e.*, simply multiply both sides by -1.

iii. Replace "$a_1x_1 + a_2x_2 + \cdots + a_nx_n = b$" with the two constraints "$a_1x_1 + a_2x_2 + \cdots + a_nx_n \leq b$" and "$a_1x_1 + a_2x_2 + \cdots + a_nx_n \geq b$".

iv. Replace x_1 with $(y_1 - z_1)$ everywhere (in the objective function and the constraints), where y_1 and z_1 are new variables, and add constraints $y_1 \geq 0$, $z_1 \geq 0$. A solution to the new LP yields a solution to the old LP with $x_1 = y_1 - z_1$.

v. Take the general LP and convert it to the form required by MELPSE by applying the transformations described in the previous parts one-by-one, in reverse order (*i.e.*, first replace unbounded variables with pairs of positive variables, then replace equality constraints with pairs of inequalities, then make all inequalities $\leq$, and finally turn the objective function into a maximization).

This results in an equivalent LP with at most twice as many variables and constraints.

□