**Worth:** 10% **Due:** By 9:59pm on Tuesday 27 October

**Remember to write the *full name* and *student number* of *every group member* prominently on your submission.**

---

*Please read and understand the policy on Collaboration given on the Course Information Sheet. Then, to protect yourself, list on the front of your submission **every** source of information you used to complete this homework (other than your own lecture and tutorial notes). For example, indicate clearly the **name** of every student from another group with whom you had discussions, the **title and sections** of every textbook you consulted (including the course textbook), the **source** of every web document you used (including documents from the course webpage), etc.*

*For each question, please write up detailed answers carefully. Make sure that you use notation and terminology correctly, and that you explain and justify what you are doing. Marks **will** be deducted for incorrect or ambiguous use of notation and terminology, and for making incorrect, unjustified, ambiguous, or vague claims in your solutions.*

---

1. (Worth 20%) 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) 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) 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) 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) Write a detailed proof that your algorithm always produces an optimal solution.

2. (Worth 25%) Consider nested function calls, such as $f(g(x), h(y, k()))$, and how they are evaluated by an interpreter. Suppose that, for technical reasons, functions require one time step for each nested function call they make, and only one function call can be made at each time step. Then the example expression above can be evaluated in four steps, as follows:

   - time 0: call $f()$ — now we know that we have to call $g()$ and $h()$,
   - time 1: $f$ calls $g()$,
   - time 2: $f$ calls $h()$ — now we know that we have to call $k()$,

- time 3: $h$ calls $k()$,
- time 4: all done!

You should have no trouble convincing yourself that every order of calls will require the same number of steps (for example, if $f$ calls $h()$ first, it does not change the other calls that must be made, just their order).

Now suppose that we have access to a parallel computer that can execute multiple functions calls at the same time. It is still the case that functions require one time step for each nested function call they make — but now different functions can make calls at the same time. Then, the example above can be evaluated in only three steps — we cannot do better, because $f$ requires two time steps (one to call $h$ and one to call $g$):

- time 0: call $f()$ — now we know that we have to call $g()$ and $h()$,
- time 1: $f$ calls $h()$ — now we know that we have to call $k()$,
- time 2: $f$ calls $g()$ *and* $h$ calls $k()$, in parallel,
- time 3: all done!

This situation can be represented by the following abstract "Efficient Function Calls" problem.

**Input:** A nested function call expression $E$ of the form "$f(e_1, \ldots, e_k)$," where $f$ is a function name and each $e_i$ is either a variable or another nested function call expression (it is possible to have $k = 0$).

**Output:** The order of nested calls for each function in $E$, to minimize the total number of steps required to evaluate the expression on the parallel computer.

Give an algorithm to solve this problem efficiently: include a brief high-level English description, as well as a detailed pseudo-code implementation. Justify that your algorithm is correct, and analyze its worst-case running time.

3. (Worth 25%) Suppose you are given a network $N$, a *maximum* flow $f$ on $N$, and one edge $e_0 \in N$ such that $f(e_0) = c(e_0)$. The flow $f$ is guaranteed to be maximum and to have integer flow values $f(e)$ for all edges $e$.

   (a) If we *decrease* $c(e_0)$ by 1 (i.e., let $c'(e_0) = c(e_0) - 1$), then we expect that the maximum flow on $N$ will also decrease by one unit. But does this always happen?
   Either give a specific example where the maximum flow on $N$ does not change (and show that this is the case), or give a general argument that the maximum flow on $N$ always changes.

   (b) Irrespective of your answer on the previous part, there are cases when this change in capacity causes the maximum flow to decrease.
   Give an *efficient* algorithm that takes a network $N$, a maximum flow $f$ on $N$, and one edge $e_0 \in N$ such that $f(e_0) = c(e_0)$ and that determines a new maximum flow in the network $N'$, where $N' = N$ except for $c'(e_0) = c(e_0) - 1$.
   Include a brief justification that your algorithm is correct (i.e., explain how your algorithm works), and a brief analysis of your algorithm's worst-case runtime (which should be as small as possible).

   (c) If we *increase* $c(e_0)$ by 1 (i.e., let $c'(e_0) = c(e_0) + 1$), then we expect that the maximum flow on $N$ will also increase by one unit. But does this always happen?
   Either give a specific example where the maximum flow on $N$ does not change (and show that this is the case), or give a general argument that the maximum flow on $N$ always changes.

(d) Irrespective of your answer on the previous part, there are cases when this change in capacity causes the maximum flow to increase.

Give an *efficient* algorithm that takes a network $N$, a maximum flow $f$ on $N$, and one edge $e_0 \in N$ such that $f(e_0) = c(e_0)$ and that determines a new maximum flow in the network $N'$, where $N' = N$ except for $c'(e_0) = c(e_0) + 1$.

Include a brief justification that your algorithm is correct (i.e., explain how your algorithm works), and a brief analysis of your algorithm's worst-case runtime (which should be as small as possible).