CS 577: Introduction to Algorithms — Homework 7

Out: 04/26/2022 — **Due: 05/05/2022**

Ground Rules

- Read the lecture notes.
- Read Chapters on Flows, Matchings, and NP problems.
- You should do ***all*** problems, but hand in only the graded problems.
- Write your name and student ID clearly on your submission.
- Clearly mark the beginning and end of your solution to each problem.

1. (Naive Ford-Fulkerson) Someone suggests that to find a maximum flow in a network flow problem on a weighted directed graph $G = (V, E, W)$, one can just look for an augmenting path $\langle s = v_0, v_1, \ldots, v_k = t \rangle$ of the following type: Every $(v_i, v_{i+1}) \in E$ and $f(v_i, v_{i+1}) < w(v_i, v_{i+1})$ for the current flow f.

   Formulate this idea into a pseudocode implementation.

   Find a counter example that shows that this approach is not sufficient in all cases (i.e., this is not a correct algorithm for the maximum flow problem, and the more involved technique involving back edges and the construction of the residue graph G_f is necessary.)

2. This problem is concerned with verifying a given solution to a network flow problem is optimal.

   Formally the problem is to find an algorithm to solve the following: Given a weighted directed graph $G = (V, E, W)$ with a source $s \in V$ and a destination $t \in V$, and a flow f. Decide whether f is a maximum flow.

   Your algorithm must run in linear time ($O(|V| + |E|)$).

3. **Graded Problem (Page limit: 1 sheet; 2 sides)** Suppose someone has already computed a maximum flow f for a network flow problem given by a weighted directed graph $G = (V, E, W)$ with a source $s \in V$ and a destination $t \in V$, where every edge has a positive integer weight w_e.

   Now suddenly it is revealed that in fact the weight value w_e for one particular edge $e = (u, v)$ is wrong, and in fact the correct value is $w'_e = w_e - 1$. But luckily it was found that the computed maximum flow f has $f_e \leq w'_e$.

   We want to use the maxflow-mincut theorem to prove that f is in fact still a maximum flow (with respect to the correct capacity weights).

   Consider the residual graph G_f with the original capacity weights. Suppose S is the set of reachable vertices in G_f from the source s. Prove that it is not possible that $u \in S$ and $v \notin S$. Conclude that the capacity of the cut (S, S^c) is unchanged in the residual graph G'_f according to the corrected capacity weights. Then conclude that f is still a maximum flow.

4. **Graded Problem (Page limit: 1 sheet; 2 sides)** Suppose we have $2n$ students, partitioned into L and R, where $|L| = |R| = n$. We want to pair them up in some order of the first n students L and the second n students R; this forms n teams $\{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$ (to do homeworks, say.)

   Each student $x \in L$ has a (linear) order of preferences of every member in R. Of course different x's may have very different preferences. Formally, each x has a private linear order of R. (There are a total of n linear orders of R, one for each x. These linear orders have no correlations.) Similarly every y has a private linear order of L. (There are a total of n linear orders of L, one for each y.)

   Our goal is to form n teams $\{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$ such that they satisfy the following condition: There are no $x_i \in L$ and $y_j \in R$, $i \neq j$, such that x_i prefers y_j over y_i (the one currently in the same team as x_i), and y_j also prefers x_i over x_j (the one currently in the same as team as y_j).

Here is a proposed algorithm: List $L = (x_1, x_2, \ldots, x_n)$ in some arbitrary order (a queue Q). Initially every $x \in L$ and $y \in R$ are "free". Remove the x at the top of the queue Q. Let x start a "proposal process": x will go down the list of R according to the private linear order of x; if x proposes to y, then there are 3 possibilities. (1) y is currently "free". (2) y is currently "tentatively teamed up" with some x' but y prefers x over x'. (3) y is currently "tentatively teamed up" with some x' but y prefers x' over x. In case (1), (x, y) is "tentatively teamed up". In case (2), y "breaks up" with x' and gets "tentatively teamed up" with x, and x' becomes "free" again, and gets back on to Q. In case (3), y rejects x and x goes down the private list of x and proposes to the next y'.

- Write a pseudocode for this algorithm.
- Prove that every time x does the "proposal process" it does get "tentatively teamed up" (which either decreases $|Q|$ by 1, or keeps $|Q|$ unchanged; how is that latter case possible?)
- Prove that this algorithm is in fact equivalent to the following slight modification: Whenever some $x \in L$ gets back onto Q, it remembers the last y that broke up with it, and when it is time for x to propose again (when it gets off Q), it does not start from the top of its private list of x, but the y' right after y on that private list.
- Prove that the algorithm eventually does terminate. (**Hint**: Design some measuare of "overall quality" so that even if $|Q|$ does not decrease per round, this measuare gets improved.)
- Prove a time bound to the algorithm.
- Prove that the algorithm correctly achieves **our goal** enunciated earlier.

5. Review the concept of P and NP. Can state three NP-complete problems (what are their definitions? what makes them NP-problems, and does it mean that they are NP-complete?) Make sure you understand all the NP-completeness proofs we give in class. Be sure to be able to re-produce these proofs.