CSE421: Design and Analysis of Algorithms

Homework 1

Anup Rao Due:

Each problem is worth 10 points:

1. If c is the first company on the applicant a's preference list and a is the first applicant on company c's preference list, does it have to be the case that c and a must be matched to each other in every stable matching?

   Solution. We will show that c and a must be matched to each other in every stable matching. Assume for the sake of contradiction that c and a are not matched in a stable matching. Then there is a company c' and an applicant a' such that company c is paired with applicant a', and applicant a is paired with company c'. But c prefers a to a' and a prefers c to c', so this matching is not stable.

2. Prove or disprove: In every stable matching, there must be at least one party who is matched to his/her top choice. HINT: Play with some 3-couple examples. Note that the problem asks about *every* stable matching, rather than the unique matching that is found by the algorithm discussed in class. Thus if you wish to prove the statement true, you need to prove it for every stable matching, whereas to show that it is false, you just need to find some stable matching that disproves it.

   Solution. The statement is not true. Here is a counterexample. The preferences are:

X	B	A	C
Y	C	B	A
Z	A	C	B

A	Y	X	Z
B	Z	Y	X
C	X	Z	Y

   Consider the matching: $X-A$, $Y-B$, $Z-C$. We will first show that this is a stable matching. Indeed, X prefers only B to his current partner, but B prefers Y. Y prefers only C to his current partner, but C prefers Z to Y. Z prefers only A to C, but A prefers X to Z. We are now done because no one is matched to someone that is first on their preference list.

3. Prove that any graph with n vertices and at least $n+k$ edges must have at least $k+1$ cycles.

   Solution. We prove the statement by induction on k.

   The base case is when $k = 0$. Suppose the graph has c connected components, and the i'th connected component has n_i vertices. Then there must be some i for which the i'th connected component has at least n_i edges, or else the total number of edges in the graph would be less than $n_1 + \ldots + n_c = n$. If this i'th connected component had no cycles, then it would be a tree, but that is not possible, because a tree must have $n_i - 1$ edges. So, this connected component must contain a cycle.

   For the inductive step, let $k \geq 1$ and assume the statement holds for graphs with $n+k-1$ edges, and let us prove that it holds for a graph with $n+k$ edges. Suppose G is a graph with

$n+k$ edges and n vertices. By induction, G must contain at least k cycles. Let e be an edge that belongs to a cycle C. If we delete e from G, we obtain a graph G' with $n+k-1$ edges. By induction, G' must contain at least k cycles. However, these k cycles cannot include C, because G' does not contain the edge e of C. If we add back e to G, we must then have $k+1$ cycles, because we will obtain the new cycle C that was not present in G'.

4. Assume have functions f, g such that $f = O(g)$, and that $f(x), g(x) > 1$ for every x. For each of the following statements, decide whether you think it is true or false and give a proof or counterexample:

   (a) $\log f(n) = O(\log g(n))$.

   Solution. The statement is true if f and g were also promised to be increasing. In this case, $f(n) = O(g(n))$ implies that there exist constants c and N such that $\forall n \geq N$, $f(n) \leq cg(n)$.

   $$\begin{aligned} f(n) &\leq cg(n) \\ \Rightarrow \log f(n) &\leq \log g(n) + \log c \end{aligned}$$

   Then

   $$\log g(n) + \log c \leq c' \log g(n)$$

   holds as long as $g(n)$ is an increasing function.

   Common Errors. There were attempts to prove the statement to be True by showing there exists a constant c, n_0 such that $\log g(n) \geq c$ for $n \geq n_0$. This is not true, as demonstrated by taking $g(n) = 1 + \frac{1}{n}$

   (b) $2^{f(n)} = O(2^{g(n)})$.

   Solution. The statement is **False**.
   Take $f(n) = \log n^2$ and $g(n) = \log n$. It is clear that $f(n) \leq 2g(n)$, thus $f(n) = O(g(n))$.

   $$\begin{aligned} 2^{f(n)} &= 2^{\log n^2} = n^2 \\ 2^{g(n)} &= 2^{\log n} = n \end{aligned}$$

   Clearly $n^2 \neq O(n)$, which shows that $2^{f(n)} \neq O\left(2^{g(n)}\right)$.

   (c) $f(n) = O(g(n)^2)$.

   Solution. The statement is **True**.
   We have $f(n) = O(g(n))$, which implies there exist constants c and N such that $\forall n \geq N$, $f(n) \leq cg(n)$.

   $$\begin{aligned} f(n) &\leq cg(n) \\ &\leq cg(n)g(n) \quad [\text{since } g(n) > 1] \end{aligned}$$

   Hence, $\forall n \geq N$, we have $f(n) \leq cg(n)^2$ which is equivalent to saying $f(n) = O(g(n)^2)$