CS 180 Homework 1 Solutions

Question 1

Decide whether you think the following statement is true or false. If it is true, give a short explanation. If it is false, give a counterexample.

True or false? Consider an instance of the Stable Matching Problem in which there exists a student s and a hospital h such that s is ranked first on the preference list of h and h is ranked first on the preference list of s. Then in every stable matching S for this instance, the pair (s, h) belongs to S.

Solution: True. Assume, for contradiction, that there exists some stable matching S that doesn't contain (s, h). This means s must be matched to some hospital $h' \neq h$ and h must be matched to some student $s' \neq s$. However, s prefers h to h' and h prefers s to s', which is an instability, meaning that this matching can't be stable. Thus, we have a contradiction, so every stable matching must include (s, h).

Note that it is insufficient to only argue that running Gale-Shapely will always output a stable matching containing the (s, h) pair, since stable matching isn't unique, so there could hypothetically be a stable matching without the (s, h) pair that Gale-Shapely wouldn't output.

Question 2

The Stable Matching Problem, as discussed in the text, assumes that all men and women have a fully ordered list of preferences. In this problem we will consider a version of the problem in which men and women can be *indifferent* between certain options. As before we have a set M of n men and a set W of n women. Assume each man and each woman ranks the members of the opposite gender, but now we allow ties in the ranking. For example (with $n = 4$), a woman could say that m_1 is ranked in first place; second place is a tie between m_2 and m_3 (she has no preference between them); and m_4 is in last place. We will say that w prefers m to m' if m is ranked higher than m' on her preference list (they are not tied).

With indifferences in the rankings, there could be two natural notions for stability. And for each, we can ask about the existence of stable matchings, as follows.

a. A *strong instability* in a perfect matching S consists of a man m and a woman w, such that each of m and w prefers the other to their partner in S. Does there always exist a perfect matching with no strong instability? Either give an example of a set of men and women with preference lists for which every perfect matching has a strong instability; or give an algorithm that is guaranteed to find a perfect matching with no strong instability.

Solution: There always exists a perfect matching with no strong instability. Consider the following algorithm:

- Resolve indifferences in every person's preference list by arbitrarily ordering them (so if m prefers w_1 the most but is indifferent to w_2 and w_3, then $w_1 > w_2 > w_3$ and $w_1 > w_3 > w_2$ are both acceptable orderings).
- Run the standard Gale-Shapely algorithm discussed in lecture.

Runtime Analysis: Since we have n people with preference lists of length n, the first step will take $O(n^2)$ time to complete. The second step will also take $O(n^2)$ time as discussed in lecture. This results in a total time of $O(n^2)$.

Proof of Correctness:

- We proved in lecture that Gale-Shapely always outputs a perfect matching, so our algorithm will also always output a perfect matching.
- Suppose, for contradiction, that our algorithm is capable of outputting a matching with a strong instability.

- This means that there's 2 pairs (m, w) and (m', w') where m prefers $w' > w$ and w' prefers $m > m'$.
- Since our transformation in step 1 preserves strong preferences (it only reorders tied rankings), this means that our input to Gale-Shapely would have also had m preferring $w > w'$ and w' preferring $m > m'$.
- However, this would be an unstable matching, despite having proven in lecture that Gale-Shapely always outputs a stable matching.
- Thus, by contradiction, the matching outputted by our algorithm will always have no strong instabilities.

b. A *weak instability* in a perfect matching S consists of a man m and a woman w, such that their partners in S are w' and m', respectively, and one of the following holds:
   - m prefers w to w', and w either prefers m to m' or is indifferent between these two choices; or
   - w prefers m to m', and m either prefers w to w' or is indifferent between these two choices.

   In other words, the pairing between m and w is either preferred by both, or preferred by one while the other is indifferent. Does there always exist a perfect matching with no weak instability? Either give an example of a set of men and women with preference lists for which every perfect matching has a weak instability; or give an algorithm that is guaranteed to find a perfect matching with no weak instability.

Solution: There does not always exist a perfect matching with no weak instability. Consider the following case with 2 men m and m', and 2 women w and w':
- m and m' both prefer w over w'
- w and w' both have no preference between m and m'

In every matching, either m or m' will be matched with w', who is their second choice. That man will prefer w over their current match, and w is indifferent, which is a weak instability. Thus, it's impossible to avoid a weak instability in this case.

We've proven by counterexample that there doesn't always exist a perfect matching with no weak instabilities.

Question 3

Take the following list of functions and arrange them in ascending order of growth rate. That is, if function $g(n)$ immediately follows function $f(n)$ in your list, then it should be the case that $f(n)$ is $O(g(n))$.

$$\begin{aligned}
g_1(n) &= 2^{\sqrt{\log n}} \\
g_2(n) &= 2^n \\
g_3(n) &= n(\log n)^3 \\
g_4(n) &= n^{4/3} \\
g_5(n) &= n^{\log n} \\
g_6(n) &= 2^{2^n} \\
g_7(n) &= 2^{n^2}
\end{aligned}$$

Solution: In order from smallest to largest, we have:

$$\begin{aligned}
g_1(n) &= 2^{\sqrt{\log n}} \\
g_3(n) &= n(\log n)^3 \\
g_4(n) &= n^{4/3} \\
g_5(n) &= n^{\log n} \\
g_2(n) &= 2^n \\
g_7(n) &= 2^{n^2} \\
g_6(n) &= 2^{2^n}
\end{aligned}$$

The original homework had a typo where g_6 was listed as 2^{2n} instead of 2^{2^n}. In that case, $g_6(n)$ would come before $g_7(n)$, and still come after $g_2(n)$. **Because of the late correction, both interpretations will be accepted.**

It's easy to compare the powers of 2 to each other, as well as the powers of n to each other just by looking at the exponent:

$$2^{\sqrt{\log n}} < 2^n < 2^{n^2} < 2^{2^n}$$
$$n(\log n)^3 < n^{4/3} < n^{\log n}$$

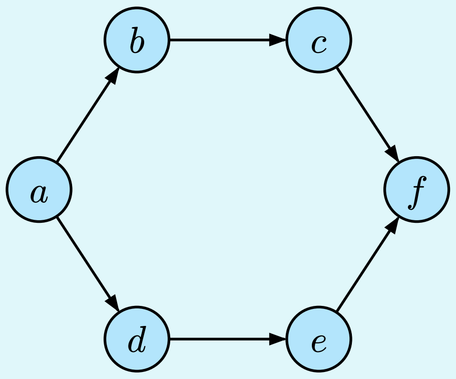
The reason $n(\log n)^3$ is less than $n^{4/3}$ is because $\log n$ is the inverse of 2^n, which grows much faster than any polynomial, so raising $\log n$ to any constant positive exponent will never outgrow n raised to any constant positive exponent. Thus, we have that $n(\log n)^3 < n^{4/3}$.

We can also determine that $2^{\sqrt{\log n}} < 2^{\log n}$ by looking at the exponent, and $2^{\log n} = n$, so $2^{\sqrt{\log n}} < n$, and thus transitively must be less than $n(\log n)^3$.

To compare $n^{\log n}$ and 2^n, it's easiest to compare their logarithms. $\log\left(n^{\log n}\right) = (\log n)^2$, whereas $\log(2^n) = n$. As discussed before, $(\log n)^2$ cannot outgrow n, so we have that $n^{\log n} < 2^n$.

Question 4

Consider the following acyclic graph G. How many topological orderings does it have?



Solution: There are 6 total orderings. There are several ways to solve this. One way is to simply list out every possible ordering: abcdef, abdcef, abdecf, adbcef, adbecf, adebcf.

Another way is to do combinatorics: a must go first and f must go last, and b must be before c and d must be before e. There are $4! = 24$ different ways to arrange $bcde$, and we must divide by $2 \cdot 2$ to account for the ordering constraint. Thus, there are $24/4 = 6$ total orderings.

Question 5

a. Prove by induction that $2^0 + 2^1 + ... + 2^{n-1} = 2^n - 1$

Solution:

Base Case: For $n = 1$, we have that $2^0 = 2^1 - 1$, which is true.

Inductive Hypothesis: Assume that $2^0 + 2^1 + ... + 2^{n-1} = 2^n - 1$.

Inductive Case: We want to show that $2^0 + 2^1 + ... + 2^n = 2^{n+1} - 1$. To do this, we add 2^n to both sides of our inductive hypothesis:

$$\begin{aligned} 2^0 + 2^1 + ... + 2^n &= 2^n - 1 + 2^n \\ &= 2 \cdot 2^n - 1 \\ &= 2^{n+1} - 1 \end{aligned}$$

We've proven the base case and the inductive case, so by induction, this statement holds true for all integers $n \geq 1$.

b. Prove by induction that $1 + 2 + ... + n = \frac{n(n+1)}{2}$

Solution:

Base Case: For $n = 1$, we have that $1 = \frac{1 \cdot 2}{2}$, which is true.

Inductive Hypothesis: Assume that $1 + 2 + ... + n = \frac{n(n+1)}{2}$.

Inductive Case: We want to show $1 + 2 + ... + (n+1) = \frac{(n+1)(n+2)}{2}$. To do this, we add $n + 1$ to both sides of our inductive hypothesis:

$$\begin{aligned} 1 + 2 + ... + (n+1) &= \frac{n(n+1)}{2} + n + 1 \\ &= \frac{n(n+1) + 2(n+1)}{2} \\ &= \frac{n^2 + 3n + 2}{2} \\ &= \frac{(n+1)(n+2)}{2} \end{aligned}$$

We've proven the base case and the inductive case, so by induction, this statement holds true for all integers $n \geq 1$.

Question 6

Find an algorithm to solve the following problem:

Given an integer array `arr[]`, find the subarray (containing at least one element) which has the maximum possible sum, and return that sum.

Note: A subarray is a continuous part of an array.

Examples:

Input: `arr[]` $= [2, 3, -8, 7, -1, 2, 3]$
Output: 11
Explanation: The subarray $[7, -1, 2, 3]$ has the largest sum 11.

Input: `arr[]` $= [-2, -4]$
Output: -2
Explanation: The subarray $[-2]$ has the largest sum -2.

Input: `arr[]` $= [5, 4, 1, 7, 8]$
Output: 25
Explanation: The subarray $[5, 4, 1, 7, 8]$ has the largest sum 25.

Solution:

Algorithm: We can solve this with a greedy algorithm:

- Initialize the variables `cur` and `best` to $\texttt{arr}[0]$
- Iterate sequentially through `arr` (skipping the first element), and for every element `ele`:
  - Set `cur` to $\max(\texttt{ele}, \texttt{cur} + \texttt{ele})$
  - Set `best` to $\max(\texttt{best}, \texttt{cur})$
- Return `best`

Runtime Analysis: Since we only do one pass through the n-element array and do constant-time operations each loop iteration, our total time is $O(n)$.

Proof of Correctness:

- We can prove this by induction on n, the length of `arr`.
- **Base Case:** For $n = 1$, our algorithm runs no loop iterations, so it'll return $\texttt{arr}[0]$, which is the only subarray so it must be the maximal sum.
- **Inductive Hypothesis:** Assume that after n elements, `cur` is the maximum subarray sum within the first n elements that includes the nth element (I'll call this the "maximum trailing subarray sum").

- **Inductive Case:** We want to prove that after element $n + 1$, `cur` will contain the maximum trailing subarray sum of the first $n + 1$ elements.
- There are 2 cases for this: case 1 is where the maximum trailing subarray sum only consists of element $n + 1$, and case 2 is where it contains multiple elements.
- Case 1 is covered by the `ele` term in the $\texttt{cur} = \max(\texttt{ele}, \texttt{cur} + \texttt{ele})$ step
- Case 2 is covered by the $\texttt{cur} + \texttt{ele}$ term. There is no larger sum, because if there were, that would mean that the maximum trailing subarray sum for the first n elements would be greater than `cur`, which contradicts our inductive hypothesis.
- Since we proved both cases, we've proven the inductive case.
- By induction, for all integers $n \geq 1$, `cur` will contain the maximum trailing subarray sum when following our algorithm
- Now, suppose for contradiction that there exists a subarray with a larger sum than our final value of `best`, and that this subarray ends at index i and has sum s
- Since our algorithm loops over every index, it must have looped over i at some point
- Since `best` is the largest value of `cur` ever seen, and $s > \texttt{best}$, we have that $s > \texttt{cur}$.
- However, we've proven that `cur` should contain the maximum subarray sum ending at index i, so s cannot be greater than `cur` since it ends at index i.
- Thus, by contradiction, `best` is the largest subarray sum.