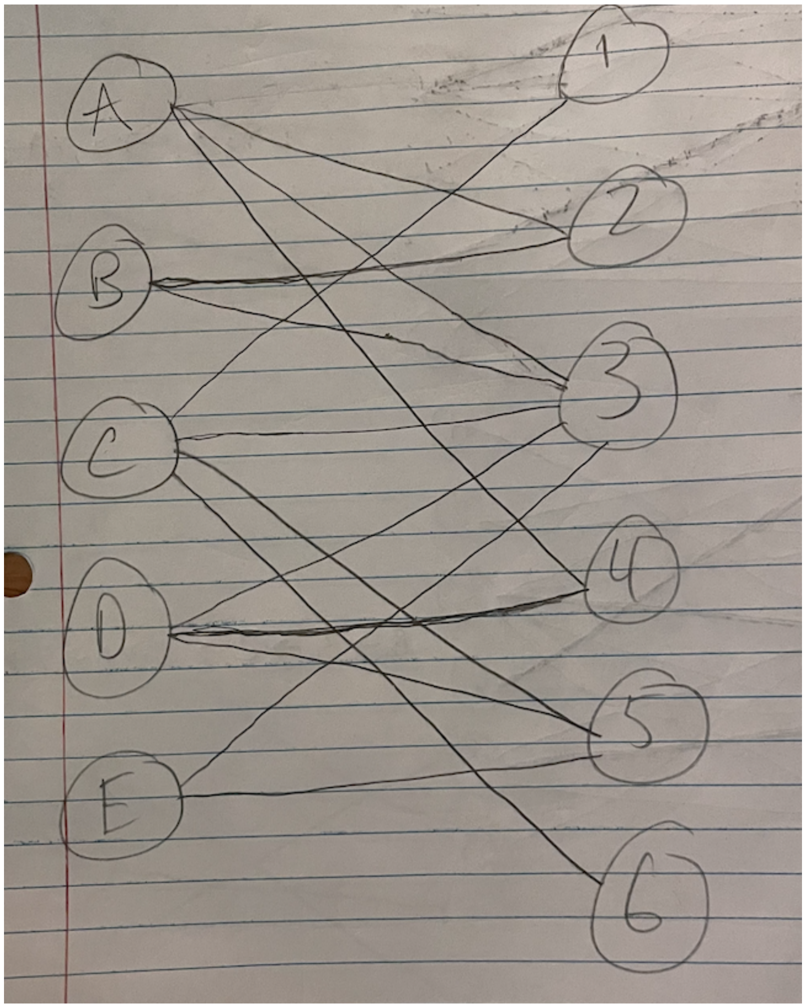
HW 6 - CS 120

Evan Jolley

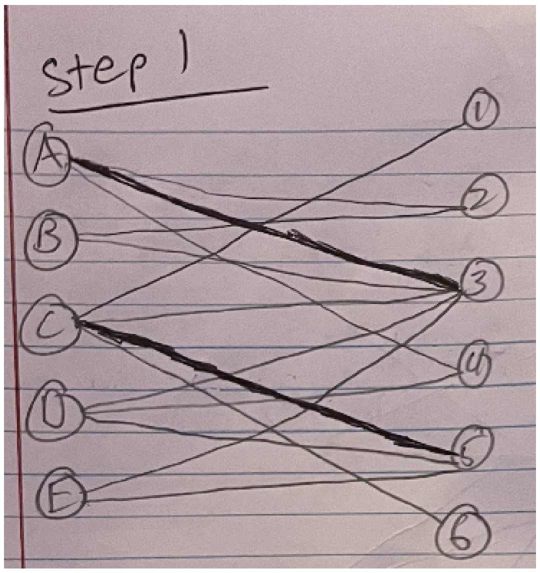
October 2022

1

1.1

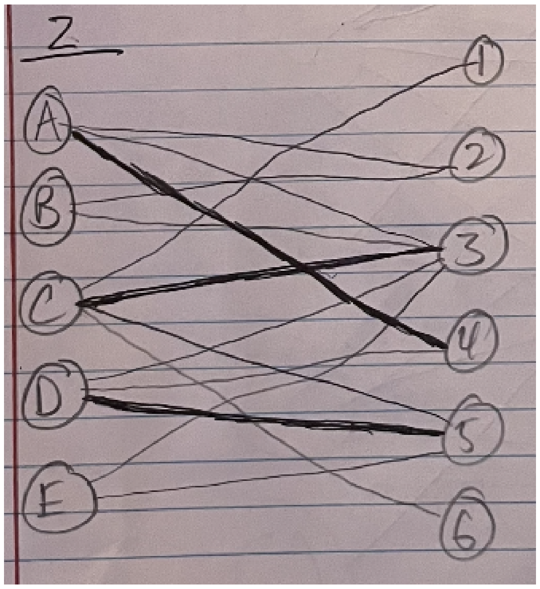


1.2



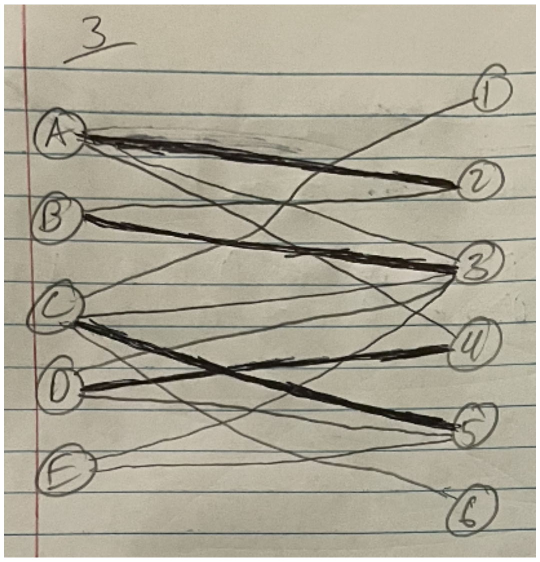
Augmenting path:

$$D \to 5 \to C \to 3 \to A \to 4$$



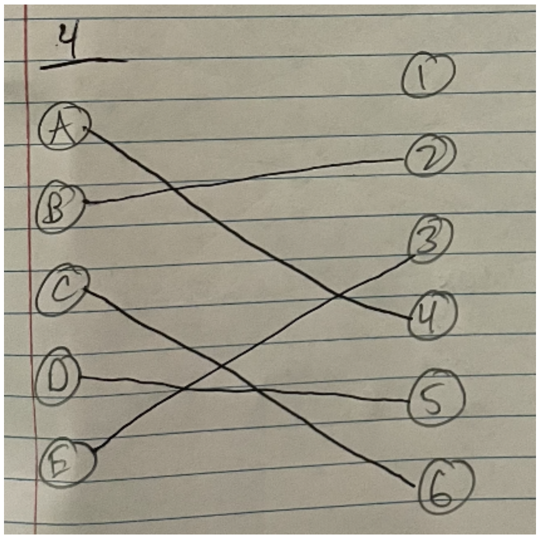
Augmenting path:

$$B \to 3 \to C \to 5 \to D \to 4 \to A \to 2$$



Augmenting path:

$$E \to 3 \to B \to 2 \to A \to 4 \to D \to 5 \to C \to 6$$



Now there are no more riders to be matched!

1.3

We will use Lecture 12's $GreedyIntervalScheduling$ to decide which rides this driver will be able to cover.

By Theorem 2.3 from the same lecture, we know that if the input is sorted by increasing order of end time, then the algorithm will find an optimal solution. This sorting of the rides in our problem looks like this (each pair corresponds to (driver, end time):

$(A, 10:29), (D, 10:44), (B, 11:14), (D, 11:29), (C, 11:44), (A, 11:59), (B, 12:14),$

$(A, 12:29), (D, 12:44), (C, 12:59), (B, 13:14), (A, 13:29), (C, 13:44), (D, 13:59)$

The algorithm starts iterating through the input, selecting all intervals that do not conflict with intervals that are already selected. The intervals collected through this process are as follows:

$$(A, 10:29), (D, 11:29), (B, 12:14), (C, 12:59), (A, 13:29), (D, 13:59)$$

Thus, the rides that should be assigned to this driver are:

1. A 10:00-10:29
2. D 11:15-11:29
3. B 11:30-12:14
4. C 12:30-12:59
5. A 13:00-13:29
6. D 13:30-13:59

2

If we think of this same scenario, but our two options are $60 + 10$ and $30 + 1$, I think many people would lean towards 60 (Maximization approach). If our two options are $60 + 10$ and $30 + 9$, I think many people would quickly lean the other way (Maximin approach). What I believe, and what I think was the point of the lecture, is that neither of these approaches feels correct 100 times out of 100. Given our situation of $60 + 10$ and $30 + 6$, I believe the thirty year-old should receive the kidney, as a 40 percent reduction in added life does not out weigh Patient B already having lived $2x$ Patient A's lifespan.

3

3.1

What I am trying to prove is that for an initial matching M, the vertices that are matched within M will all still be matched in M'. Intuitively this makes sense, because our $MaxMatchingAugPaths$ algorithm from Lecture 13 never unmatches a vertex, it simply changes what it is matched to. Let's provide a bit more rigor to this using an induction proof.

Base Case:

We begin with a matching M. Using the proof of Lemma 4.2 from Lecture 13, we can see that all matched vertices in M remain matched in the next iteration of our algorithm:

$$M_1 = (M - ((v_1, v_2), (v_3, v_4), ..., (v_{l2}, v_{l1}))) \cup ((v_0, v_1), (v_2, v_3)..., (v_{l1}, v_l))$$

All vertices that are present in our M edges ($(v_1, v_2), (v_3, v_4)$, etc.) will be present in our M_1, albeit through different edges ($(v_0, v_1), (v_2, v_3)$, etc.) Thus, we know that $V(M) \subseteq V(M_1)$.

Inductive Hypothesis:

My inductive hypothesis is that $V(M) \subseteq V(M_{i-1})$ where M_{i-1} is a matching within the $MaxMatchingAugPaths$ algorithm as we iterate from M to M'. I.e. the matchings that appear in our algorithm are:

$$M, M_1, M_2, ..., M_{i-1}, M_i, ..., M'$$

Inductive Step:

Now I want to prove that $V(M) \subseteq V(M_i)$. This can be accomplished in a few steps:

- We know that $V(M) \subseteq V(M_{i-1})$ given our inductive hypothesis.
- By following the Lemma 4.2 from Lecture 13 and the same logic as our base case, we know that $V(M_{i-1}) \subseteq V(M_i)$.
- Given that being a subset is transitive: $V(M) \subseteq V(M_{i-1}) \subseteq V(M_i) \rightarrow V(M) \subseteq V(M_i)$.

We have thus proven $V(M) \subseteq V(M_i)$ through induction.

3.2

Let's say there is a matching M that maximizes the weight, and a matching $M*$ that maximizes the size of the matching.

By 3a, we know that $V(M) \subseteq V(M*)$ because $M*$ has maximum size. In other words, all vertices that are matched in M are also matched in $M*$.

As we iterate from M to $M*$, we add more vertices to our matching and all of their weights to our total weight. From 3a, we also know that vertices are never unmatched (e.g. removed from $V(M)$). Thus, $w(M*) = w(M) + w(\textit{added vertices})$, or, $w(M*) \geq w(M)$.

Given that we defined M as a matching that maximizes weight, $w(M)$ is the largest possible value $w()$ can return. Thus, $w(M*) = w(M)$, and we can conclude that $M*$ also maximizes weight.

4

4.1