Math 182 HW3 Jun Ryu

Section 1

1) False; we give a counterexample.

Suppose we have 2 men (m_1, m_2) and 2 women (w_1, w_2) with the following preferences below:

m_1	w_1	m_2	w_2
1. w_1	1. m_2	1. w_2	1. m_1
2. w_2	2. m_1	2. w_1	2. m_2

In this case, no pairs of 1 man and 1 woman rank each other as first so it does not appear in any stable matching.

2) True; If we consider our pair (m, w) and suppose there exists other pairs (m, w^*) and (m^*, w) that appear in a perfect matching, then this matching cannot be stable since both m and w prefer each other over w^* and m^*, respectively.

3) We resolve the question by claiming there is a case where there is no stable pair of schedules. Thus, we provide a counter example:

Network A	Network B
Show a_1 (rating 5)	show b_1 (rating 2)
show a_2 (rating 10)	show b_2 (rating 8)

In this case, if a_1 is paired with b_1, Network B will try to switch the spots of b_1 and b_2 in order to win one slot instead of zero. If a_1 is paired with b_2, Network A will try to switch the spots of a_1 and a_2 in order to win two slots instead of one. Thus, this arrangement is not stable since both networks can unilarily change their own schedule to try to win more slots.

5) a) Yes, there always exists a perfect matching with no strong instability.

We give an algorithm to prove this:

the trick is to somehow break all ties in the preferences and then run a normal stable matching algorithm.

One way to break a tie is just by an alphabetical order; suppose a man m has a tie between w and w^* in his preferences, then we compare the names of w and w^* and WLOG order it in alphabetical order. Now, we run the normal stable matching algorithm and conclude there are no instabilities

$\Rightarrow$ no strong instability

b) No, and we prove this using a counterexample. Suppose we have 2 men (m_1, m_2) and 2 women (w_1, w_2)

Suppose the following arrangement:

1. w_1 or w_2

1. m_1
2. m_2

1. w_1 or w_2

1. m_1
2. m_2

In this case, all possible pairs have weak instability since WLOG, suppose m_1 is paired with w_1. Then, m_1 and w_2 form a weak instability.

Section 6

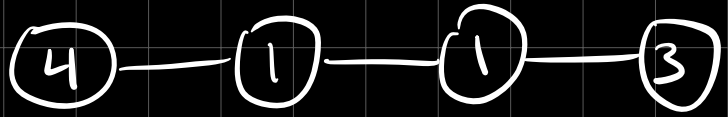
1) a) Counterexample:

consider 

This algorithm will simply pick out the middle node, but the max. weight independent set consists of the first and the last nodes.

b) Counterexample:

Consider



This algorithm will compare between (4+1) and (3+1) returning 5, but the max. weight actually is retrieved by the first and the last nodes, giving us 7.

c) Since all nodes that are part of an independent set cannot be joined by an edge, we have the following recurrence relation:

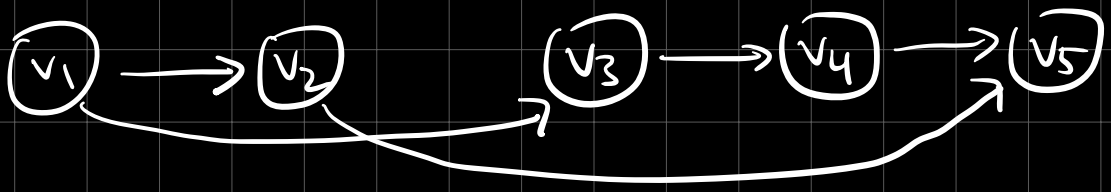
$X_i = w_i + X_{i-2}$, where X_i represents total weight and w_i represents the weight of the node at position i. However, we must also consider the fact that the node in the middle can be greater than the sum of its 2 adjacent nodes. Thus, we take the max function as follows:

$$X_i = \max(X_{i-1}, w_i + X_{i-2})$$

Using this, we can deduce our independent set from our maximum weight and since we have n iterations of this algorithm with constant time each, our running time is $O(n)$.

3) a) Counterexample:

Suppose we have



The given algorithm will return $v_1 \to v_2 \to v_5$ giving us length 2, but we desire $v_1 \to v_3 \to v_4 \to v_5$ with length 3.

b) we have the following pseudocode:

```
def longestpath(n):
    Array P = [1...n]
    P[1] = 0          (since v1 → v1 has length 0)
```

For $i = 2 \ldots n$:

$P = -\infty$ (will keep updating this value in each iteration of the loop)

For all edges (i, j):

if $P[j] + 1 > P$, then

$P = P[j] + 1$

Set $P[i] = P$ (this is the longest path from P_1 to P_i)

Return $P[n]$

(this returns the longest path in $O(n^2)$ time)

13) We are given that if the product of the ratios along the cycle is above 1, it is an opportunity cycle. Hence, let us build a directed graph $G = (V, E)$ with each directed edge connecting each pair of stocks.

Thus, we have, for a cycle C,

$\prod_{(i,j) \in C} r_{ij} > 1$ gives us an opportunity cycle

We take the logarithm of both sides,

$$\sum_{(i,j) \in C} \log r_{ij} > 0$$

This gives us that C is an opportunity cycle iff it is a positive cycle. Now, we just use a polynomial-time algorithm to find a positive cycle within G.

14) a) We construct a new graph such that it has all nodes from each G_i's and also it only contains edges that are contained in every G_i's. We do this to make sure this path can appear in every G_i. Now, we simply perform BFS in order to find the shortest path from s to t.

b) Let us denote the minimum cost as min(i) for graphs $G_0 \dots G_i$.

Also, let $G(i,j)$ with $i \leq j \leq b$ denote a graph where it consists of

all edges found in all of $G_i \dots G_j$ and let $\ell(i,j)$ be the length of the shortest path in $G(i,j)$. Now, let's take the case where going from G_i to G_{i+1} is the last change in the progression of graphs. Then, we have

$$\min(b) = \min(i) + (b-i)\,\ell(i+1,b) + K.$$

However, there are also cases where there are no changes. In this case, it is simply $\min(b) = (b+1)\,\ell(0,b)$. Thus, now we can combine both cases:

$$\min(b) = \min\left[(b+1)\,\ell(0,b),\ \min(i) + (b-i)\,\ell(i+1,b) + K\right]$$

Now, we notice that once we have all $G(i,j)$ graphs and $\ell(i,j)$ values given, then the algorithm will take $O(b^2)$ time to go through all pairs.

Section 7

1) a) There are 3 cuts:

Cut 1: $A = \{s\}, B = \{u, v, t\}$

Cut 2: $A = \{s, v\}, B = \{u, t\}$

Cut 3: $A = \{s, u, v\}, B = \{t\}$

All these cuts have min. capacity 2.

b) The above 3 cuts apply to Figure 7.25 as well, except the minimum capacity in this case is 6.

2) a) The value of the flow is

$$5 + 5 + 8 = 18.$$

This is not the maximum flow as shown in part b).

b) The minimum s-t cut occurs when we divide the nodes so that

set A consists of s and the very top node and set B consists of all other nodes. In this case, the min. capacity is $5+8+5+3 = 21 \neq 18$.

3) a) The value of the flow is $6+3+1 = 10$. This is not the maximum flow as shown in part b).

b) The min. s-t cut occurs when $A = \{s, a, b, c\}$, $B = \{d, t\}$.

In this case, the min. capacity is $5+5+1 = 11 \neq 10$

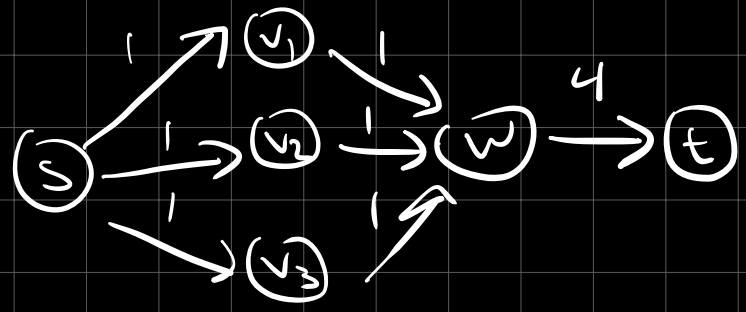
4) False: counterexample:

Suppose the following:

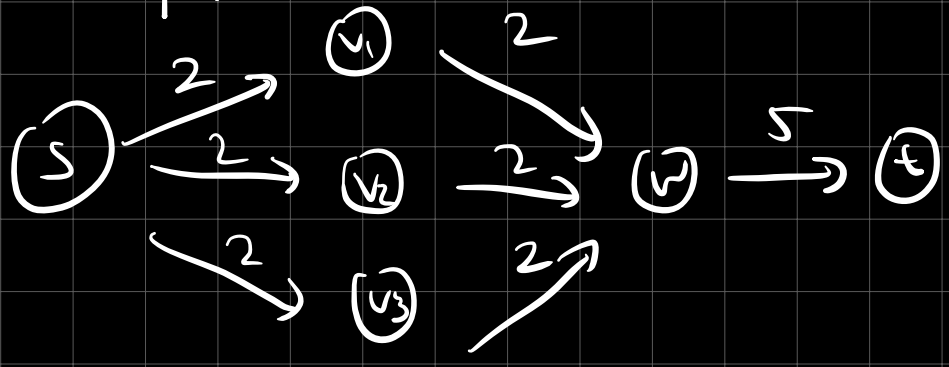
The maximum flow is clearly 1 which is not equal to 2 (does not saturate the edge out of s)

5) False; counterexample:

Suppose the following:



The minimum cut in this case is one where $A = \{s\}$, $B = V - A$ with capacity 3. Now, if we add 1 to all edges, we get



The same cut as above gives us capacity 6, which is not minimal as a new cut $B = \{t\}$, $A = V - B$ gives us capacity 5.