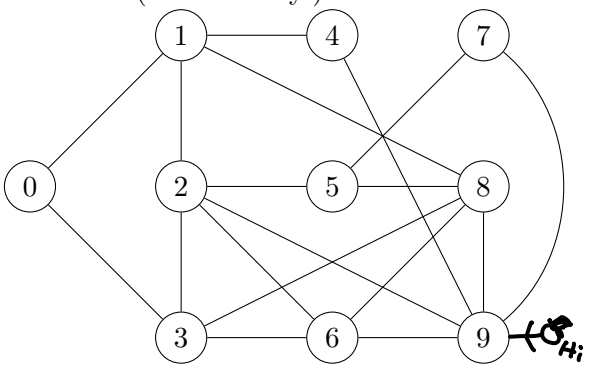
CSC 404 - Exam 2 - Name:

Solve 9 of the following 10 problems :-). Or you can do all 10 and I will throw a few bonus points your way.

Problem 1 (Reachability!)**.** Consider the following graph:



a. By running a breadth-first search, determine all of the vertices/nodes that can be reached starting at 0. Draw the resulting tree that stems from 0 and branches out to all of the reached vertices/nodes.

b. Which nodes can be reached within 3 edges/steps from 0?

c. By running a depth-first search, determine all of the vertices/nodes that can be reached starting at 0. Draw the resulting tree that stems from 0 and branches out to all of the reached vertices/nodes.

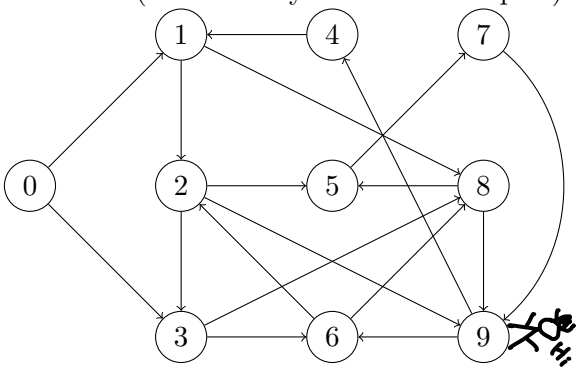
d. Construct the Adjacency Matrix, A.

$$A = \begin{pmatrix} -- & -- & -- & -- & -- & -- & -- & -- & -- & -- \\ -- & -- & -- & -- & -- & -- & -- & -- & -- & -- \\ -- & -- & -- & -- & -- & -- & -- & -- & -- & -- \\ -- & -- & -- & -- & -- & -- & -- & -- & -- & -- \\ -- & -- & -- & -- & -- & -- & -- & -- & -- & -- \\ -- & -- & -- & -- & -- & -- & -- & -- & -- & -- \\ -- & -- & -- & -- & -- & -- & -- & -- & -- & -- \\ -- & -- & -- & -- & -- & -- & -- & -- & -- & -- \\ -- & -- & -- & -- & -- & -- & -- & -- & -- & -- \\ -- & -- & -- & -- & -- & -- & -- & -- & -- & -- \end{pmatrix}$$

e. By using A^3, determine the number of paths of length 3 from 0 to 9. Then, list these paths.

f. By using A^4, determine the number of paths of length 4 from 0 to 9. Then, list these paths :-).

Problem 2 (Reachability - Directed Graphs!)**.** Consider the following directed graph:



a. By running a breadth-first search, determine all of the vertices/nodes that can be reached starting at 0. Draw the resulting tree that stems from 0 and branches out to all of the reached vertices/nodes.

b. Which nodes can be reached within 3 edges/steps from 0?

c. By running a depth-first search, determine all of the vertices/nodes that can be reached starting at 0. Draw the resulting tree that stems from 0 and branches out to all of the reached vertices/nodes.

d. Similar to to non-directed graphs, we can construct an adjacency matrix A that records (directed) edges between two vertices. Since $0 \to 1$ we have a 1 in the $(0, 1)$ spot. Since we do not have an arc from 1 to 0 we have a 0 in $(1, 0)$ splot. Construct the matrix A.

$$A = \begin{pmatrix} -- & 1 & -- & -- & -- & -- & -- & -- & -- & -- \\ 0 & -- & -- & -- & -- & -- & -- & -- & -- & -- \\ -- & -- & -- & -- & -- & -- & -- & -- & -- & -- \\ -- & -- & -- & -- & -- & -- & -- & -- & -- & -- \\ -- & -- & -- & -- & -- & -- & -- & -- & -- & -- \\ -- & -- & -- & -- & -- & -- & -- & -- & -- & -- \\ -- & -- & -- & -- & -- & -- & -- & -- & -- & -- \\ -- & -- & -- & -- & -- & -- & -- & -- & -- & -- \\ -- & -- & -- & -- & -- & -- & -- & -- & -- & -- \\ -- & -- & -- & -- & -- & -- & -- & -- & -- & -- \end{pmatrix}$$

e. By using A^3, determine the number of paths of length 3 from 0 to 9. Then, list these paths.

f. By using A^5, determine the number of paths of length 5 from 0 to 9. Then, list these paths :-). (Now I will make you look at A^5)

g. By using A^6, determine the number of paths of length 6 from 0 to 9. Then, list **two** such paths :-).

Problem 3 (More Swapping Pennies!)**.** In Homework 4 we looked into a simple game called matching pennies. Here two players Eve (Even) and Bob (Odd) start with two pennies and depending on the outcome of a secret revel pennies are passed between them – first person to 4 pennies wins.

Let's change things up – here the secret revel will be a game of Rock-Paper-Scissors. If who ever wins receives an opponents penny. But, here is the twist – if there is a tie the two players swap penny piles (e.g., if Eve has 3 pennies and Bob has 1 penny following a tie Eve will have 1 penny and Bob will have 3 pennies).

Consider the following example, where Eve and Bob both start with two pennies. (Here the states `Si` denote how many pennies Eve has with `S0 = SL` and `S4 = SW`.

Round	Eve	Bob	Eve (#)	Bob (#)	State	Description
0	-	-	2	2	S2	Initially both have two pennies!
1	R	S	3	1	S3	Rock Beats Scissors - Eve wins a Penny!
2	P	P	1	3	S1	Paper Tie - Swap Penny Piles
4	S	P	2	2	S2	Scissors Beats Paper - Eve wins a Penny
5	S	R	1	3	S1	Rock Beats Scissors - Eve loses a penny
6	S	S	3	1	S3	Scissors Tie - Swap Penny Piles
7	S	P	4	0	S4 = SW	Scissors Beats Paper - Eve wins/wins overall

a. If both parties randomly throw R, P, or S, then each have a $1/3 = 0.3333$ chance of winning, losing, or tie.

i. Construct the state diagram for this game. Add loops on 1 at the two 'end game states' - `SW` and `SL` (i.e., make them absorbing states).

ii. Construct the state transition matrix, P, for this game. Label the rows SL, S1, S2, S3, SW. (You should see a 1 in the top left and bottom right)

$$P = \begin{array}{c} \\ SL \\ S1 \\ S2 \\ S3 \\ SW \end{array} \begin{array}{c} \begin{array}{ccccc} SL & S1 & S2 & S3 & SW \end{array} \\ \begin{pmatrix} 1 & -- & -- & -- & -- \\ -- & -- & -- & -- & -- \\ -- & -- & -- & -- & -- \\ -- & -- & -- & -- & -- \\ -- & -- & -- & -- & 1 \end{pmatrix} \end{array}$$

iii. Compute P^{100} (or your favorite power of P). You should see a matrix with (essentially) zeros in the middle and all of the data on the left and right columns. From this, record the probability of Eve winning/losing. (Remember we start with 2 pennies, but you can also see the probabilities of starting with 1 or 3 pennies)

b. Eve has been tipped off that if Bob has three pennies, i.e., we are at state `S1`, then he is more likely to throw Rock than Scissors or Paper (50% Rock to 25% Scissors and 25% Paper). With that, whenever Bob is up to three pennies Eve plans to always throw Paper.

i. Construct the state diagram for this game. Add loops on 1 at the two 'end game states' - `SW` and `SL` (i.e., make them absorbing states).

ii. Construct the state transition matrix, P, for this game.

$$P = \begin{array}{c} \\ SL \\ S1 \\ S2 \\ S3 \\ SW \end{array} \begin{array}{c} \begin{array}{ccccc} SL & S1 & S2 & S3 & SW \end{array} \\ \begin{pmatrix} 1 & -- & -- & -- & -- \\ -- & -- & -- & -- & -- \\ -- & -- & -- & -- & -- \\ -- & -- & -- & -- & -- \\ -- & -- & -- & -- & 1 \end{pmatrix} \end{array}$$

iii. Compute P^{100} (or your favorite power of P). You should see a matrix with (essentially) zeros in the middle and all of the data on the left and right columns. From this, record the probability of Eve winning/losing.

c. (Bonus) If you have more ideas for variations to this game let me know (the above example was from a student's submission in HW4)

Problem 4 (McEliece Cryptosystem – one more time, because it is really cool)**.** Alice wants to use the McEliece Public Key Cryptosystem with a $[n, k]$ Hamming Error Correcting Code, C. To keep things 'small', let's use $[n, k] = [7, 4]$. i.e., $r = 3$.

a. KEY GEN!

i. Construct your favorite $k \times k$ permutation matrix S and an $n \times n$ permutation matrix P.

ii. Compute and publish $G_1 = SGP$.

b. ENCRYPTION!

i. Let $m = 1011$ be Bob's $k = 4$-bit message.

ii. Compute $c = mG_1$ and change one of the bits!

c. DECRYPTION!

i. Compute $c_1 = cP^T$

ii. Apply Error Correcting Code Decoder to c_1 to find codeword x_1 that is closest to c_1. Then, let x_0 be the first k bits of x_1.

iii. Compute $x_0 S^T$. Did this return m?

Problem 5 (A 'bit' of bad hashing)**.** In our chat about Hashing Algorithm we constructed a 'Bad Hash' by XORing all of the entries together (all of the collisions). In our new version of 'Bad Hash' we initialize the digest to be 0, XOR the next character, then XOR that result with its bit reversal. (Here, $rev(1101) = 1011$)

```
badHashV2(str2hash)
    d = 0
    for x in str2hash_unicode:
      d = d xor x
      d = d xor bit_reversal(d)
    return(d)
```

For example, let's play with the string 'Hash'. Notice that

$$Hash \sim [H, a, s, h] \sim [72, 97, 115, 104] \sim [01001000, 01100001, 01110011, 01101000]$$

Round 1: $d = 0$ and $x = 72 = 01001000$
$d = d \oplus x = 00000000 \oplus 01001000 = 01001000$
$d = d \oplus rev(d) = 01001000 \oplus 00010010 = 01011010 = 90$
Round 2: $d = 90 = 01011010$ and $x = 97 = 01100001$
$d = d \oplus x = 01011010 \oplus 01100001 = 00111011$
$d = d \oplus rev(d) = 00111011 \oplus 11011100 = 11100111 = 231$
Round 3: $d = 231 = 11100111$ and $x = 115 = 01110011$
$d = d \oplus x = 11100111 \oplus 01110011 = 10010100$
$d = d \oplus rev(d) = 10010100 \oplus 00101001 = 10111101 = 189$
Round 4: $d = 189 = 10111101$ and $x = 104 = 01101000$
$d = d \oplus x = 10111101 \oplus 01101000 = 11010101$
$d = d \oplus rev(d) = 11010101 \oplus 10101011 = 01111110 = 126$

Thus, `badHashV2('Hash') = 126 = 0b01111110`.

a. Run the badHashV2 on the string 'Cyber'.

$$[C, y, b, e, r] = [67, 121, 98, 101, 114] = [01000011, 01111001, 01100010, 01100101, 01110010]$$

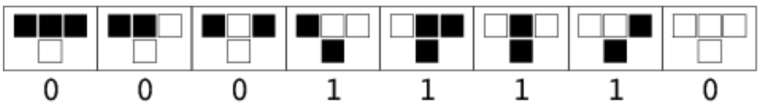
b. On D2L I have a link to an implementation of this 'Bad Hash'. Verify your work in part a. and determine another string with the same hash. (i.e., find a collision!).

c. Let's find some more collisions! Based on the birthday paradox, for a collection of about 16 strings we have a pretty decent chance of finding a collision. (Randomly) generate a list of 16 strings and compute the corresponding 'Bad Hash' and record any collision(s) that exist. (If not collisions rerun until you find a collision).

d. (Bonus) You know you want to build this into a Turing Machine.

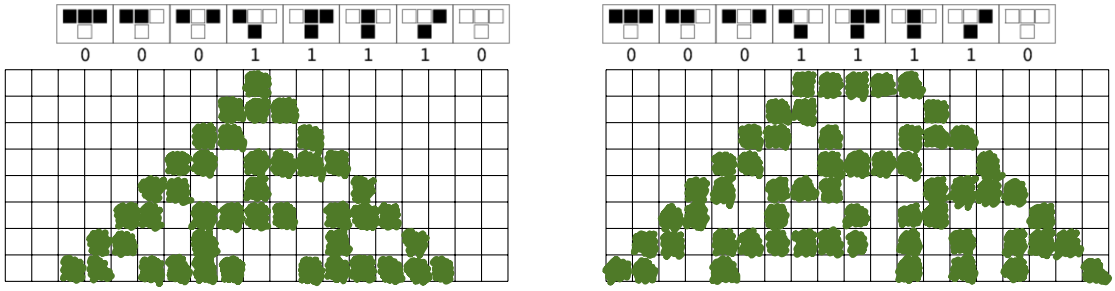
Problem 6 (1D Cellular Automata!)**.** In our last section we played around with Conway's Game of Life – a 2D cellular automata. Here we constructed new generations and stored them in a new 2D array and then 'animated' them by displaying individual frames on top of each other (think old school flip-books). We can back things up and look at 1D cellular automata. Here, each generation will be a 1D array and we 'animate' these by printing future generations below the current generation (though you can also do traditional flip-book animations).

As with the Game of Life (2D cellular automata), in 1D cellular automata future states will be determined by the current state of a cell (dead/alive) and the states of its two neighbors. Since there are $2 \cdot 2 \cdot 2 = 2^3 = 8$ possible binary states for the three cells neighboring a given cell, there are a total of $2^8 = 256$ one-dimensional cellular automata. These can be indexed with an 8-bit binary number, for example, the following table gives the evolution of rule 30 ($30 = 00011110$).

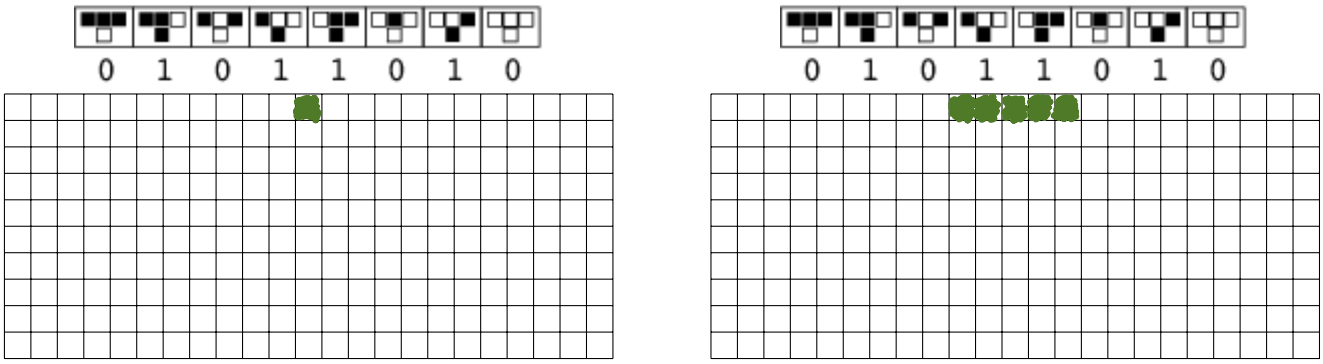


The first says if a cell is currently alive and its two neighbors are alive it will die. The second says if a cell is alive and the neighbor to the left is alive and the neighbor to the right is dead it will die....

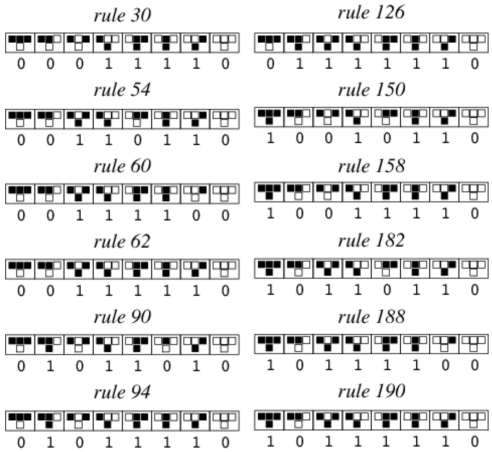
With this, we can construct future generations (rows) based off of different starting generations (first row).



a. Construct future generations using rule 90 ($90 = 01011010$)



b. (Bonus) Implement the 1D cellular automata in a language of your choice. Some other interesting 'Rules' to try it on are the following. Note - for some of these you will see a very structure emerge (e.g., symmetries). Whereas rules like Rule 30 are chaotic. In fact, Rule 30 is used in the pseudo random number generator for lager integers in the Wolfram Language. For more information see https://mathworld.wolfram.com/ElementaryCellularAutomaton.html

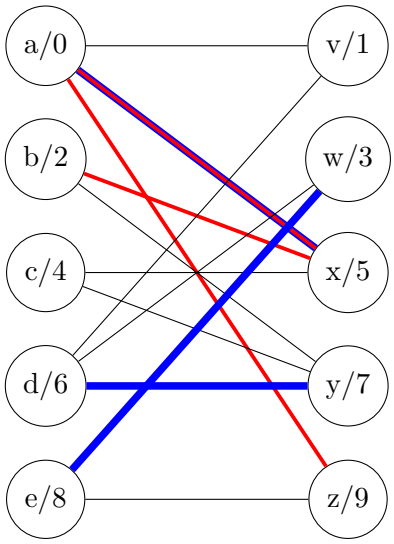


Problem 7 (Matching – Hopcroft and Karp)**.** Given a matching M in a bipartite graph G, we say that a simple path P in G is an 'augmenting path' with respect to M if it

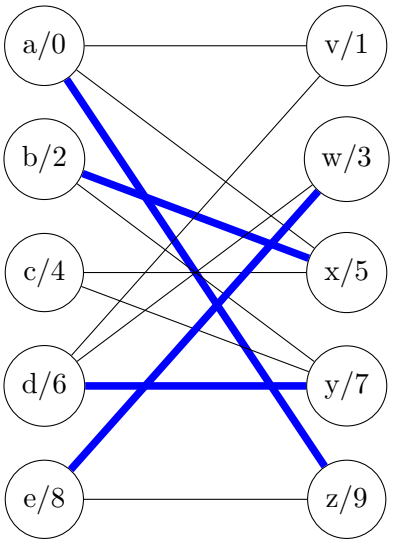
i. Starts at an unmatched vertex on one side and ends at an unmatched vertex on the other side.
ii. Its edges alternate between a non-matched edges and matched edges.

By identifying augmenting paths we can apply a symmetric difference/XOR to increase the size of the matching!

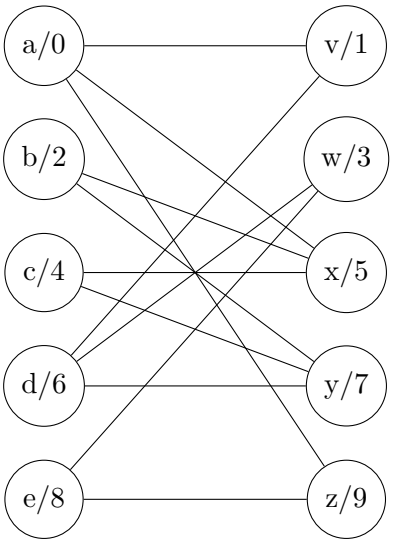
a. Consider the matching $M = \{05, 67, 83\}$ with size $|M| = 3$ (given in blue below). An augmenting path is $P = \{25, 50, 09\}$ (given in red). Thus, we can compute $M \oplus P$ to improve or original matching! Here $M' = M \oplus P = \{09, 25, 67, 83\}$ (All but the double counted edge 05). Using M' identify another augmenting path P' and use this to construct a maximal matching $M'' = M' \oplus P'$.



Matching: $M = \{05, 67, 83\}$
Path: $P = \{25, 50, 09\}$

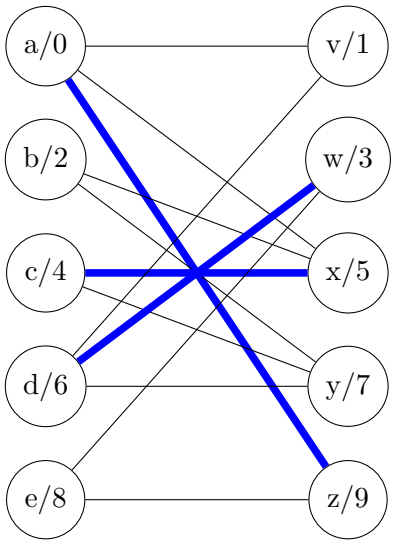


Matching $M' = M \oplus P$:
$M' = \{09, 25, 67, 83\}$
$P' =$

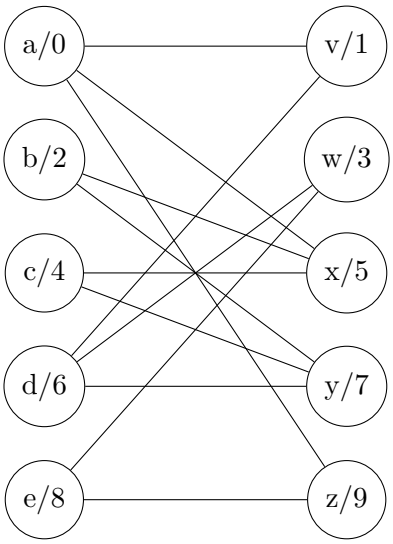


Matching $M'' = M' \oplus P'$:
$M'' =$

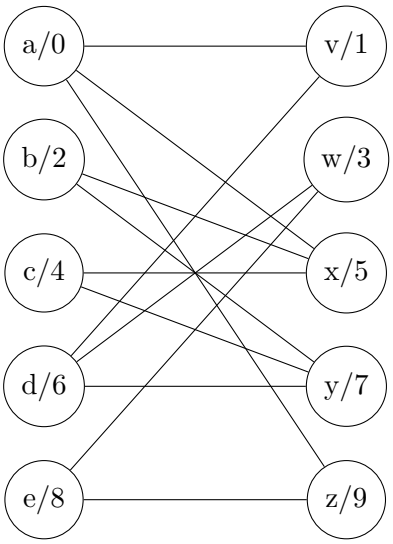
b. Consider the matching $M = \{09, 45, 63\}$ with size $|M| = 3$. Identify an augmenting path P and construct $M' = M \oplus P$. Then identify an augmenting path P' to M' and construct $M'' = M' \oplus P'$.



Matching: $M =$
Path: $P =$

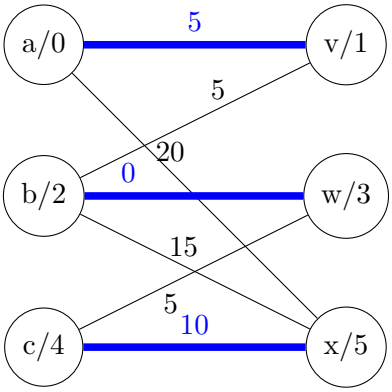


Matching $M' = M \oplus P$:
$M' =$
$P' =$

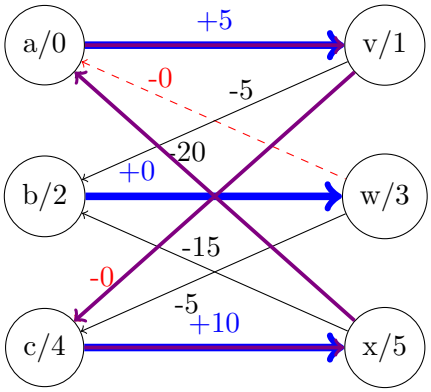


Matching $M'' = M' \oplus P'$:
$M'' =$

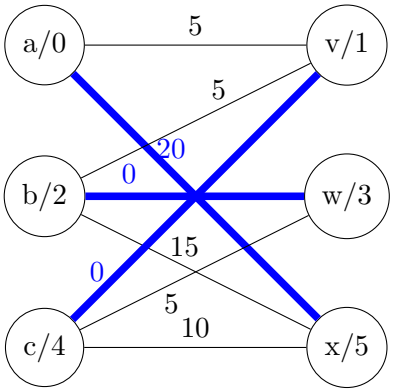
Problem 8 (Weighted Matching - Option 1!)**.** What happens if we weight our edges? Consider the following (Easter Bunny Vomit) bipartite graph with weights (the terribleness on the left). Here we assume every vertex on the left is connected to a vertex on the right - if there is no 'formal' connection, we can add an edge with a weight of zero.



Matching: $M = \{01, 23, 45\}$
Weight Sum: $5 + 0 + 10 = 15$



Directed Graph: D
Cycle: $C = \{01, 14, 45, 50\}$



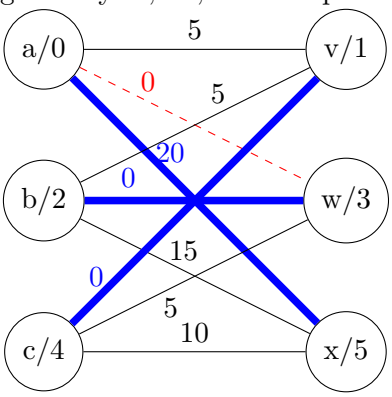
New Matching: $M' = \{05, 23, 41\}$
Weight Sum: $20 + 0 + 0 = 20$

Our approach is a neat blend of the two approaches to finding maximal matchings! Given a matching M we construct a directed graph with edges from the matching M as directed edges (from L to R) with positive weights and the unused edges, $E - M$, as directed edges (from R to L) with negative weights (middle graph). Then, find a negative cycle in D – that is, a path that starts and stops at the same place with the sum of all the weights negative. Call this cycle C. Then, $M' = M \oplus C$ is a matching with larger weight sum - Neat!

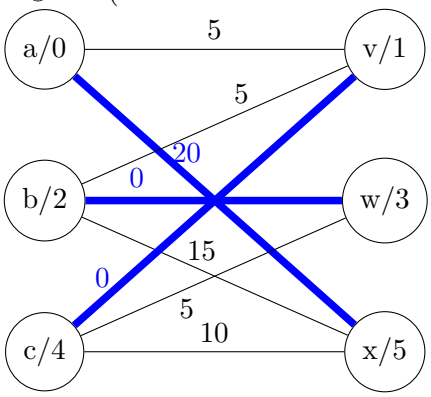
For the initial matching of $M = \{01, 23, 45\}$ we have a weight sum of $w_{01} + w_{23} + w_{45} = 5 + 0 + 10 = 15$. We then build the associated directed graph D (even more Easter Bunny Vomit). Within this graph there is a negative cycle! The cycle $C = \{01, 14, 45, 50\}$ has a weight sum of $5 - 0 + 10 - 20 = -5$. We then compute $M' = M \oplus C = \{05, 23, 41\}$ and see the weight sum of this matching is $20 + 0 + 0 = 20$ (our value went up by 5). Then we start the process again! That is, we construct a directed graph, look for negative cycles, compute the symmetric difference/XOR. We keep doing this until there are no negative cycles.

```
Maximum Weighted Bipartite Matching - Option 1
1.  M = Any Perfect Matching
2.  D = Directed Graph from M
3.  While there is a negative cycle C in D
4.      M = M ⊕ C
5.      Update D
6.  Return M
```

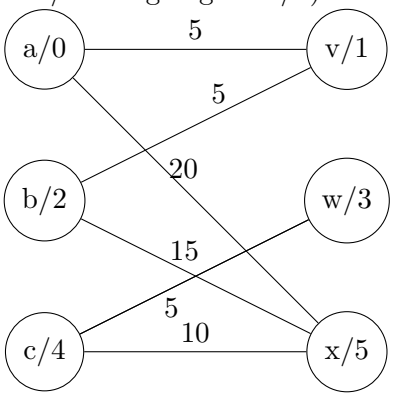
Finally, it is your turn to play! Using matching M' construct the directed graph D' from M' and identify a negative cycle, C', and compute $M'' = D' \oplus C'$. (I would recommend starting at $b/2$ and going to $w/3$)



Matching: $M' = \{05, 23, 41\}$
Weight Sum: $20 + 0 + 0 = 20$



Directed Graph: D'
Cycle: $C =$



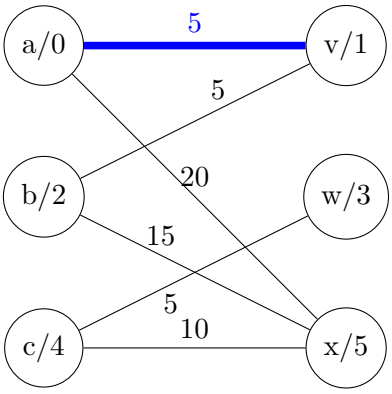
New Matching: $M'' =$
Weight Sum:

After this pass you should be at the maximal matching with weight sum of 30. If not, go crazy and run it again. I would recommend giving it another go from the start with a different first cycle (e.g., $C = \{23, 34, 45, 52\}$. If you are feeling extra excited, give it a full implementation and run it on some proper bipartite graphs (read: run it on some big ole systems) - you know you want to (well if it was not the end of the semester you would want to).

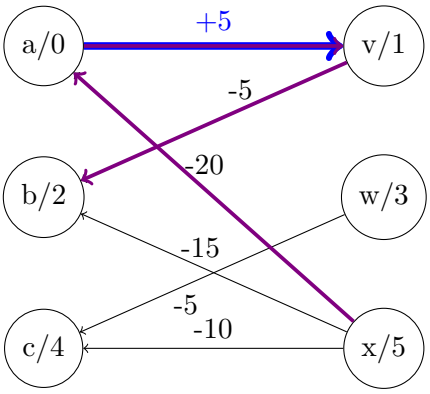
Problem 9 (Weighted Matching - Option 2!)**.** Here we continue this discussion! The following method, known as the Hungarian method, was first introduced by Kuhn and later improved by Munchers (showing polynomial running time) as well as Iri and Edmond/Karp.

Here we start with an empty matching $M = [\,]$ (or a single match) and in each iteration construct the digraph D with edges from the matching M as directed edges (from L to R) with positive weights and the unused edges, $E - M$, as directed edges (from R to L) with negative weights (middle graph).

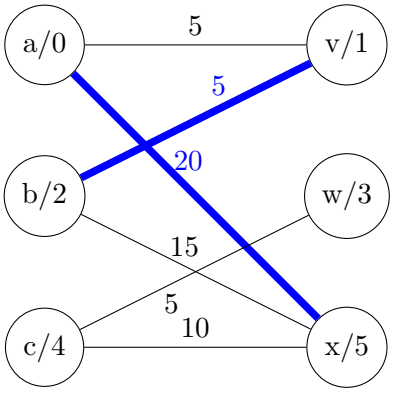
The idea is then to augment M with the shortest augmenting path – i.e., the negative length path with largest absolute value, in each iteration. To find the shortest augmenting path, we can use the Bellman-Ford algorithm!



Matching: $M = \{01\}$
Weight Sum: 5



Directed Graph: D
Path: $P = \{50, 01, 12\}$

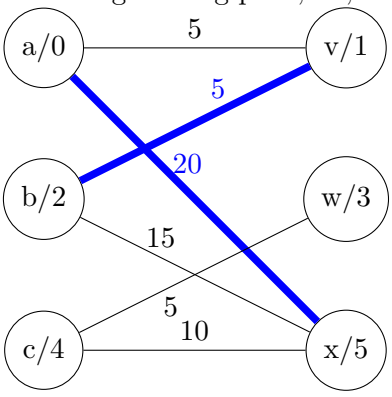


New Matching: $M' = \{05, 12\}$
Weight Sum: $5 + 20 = 25$

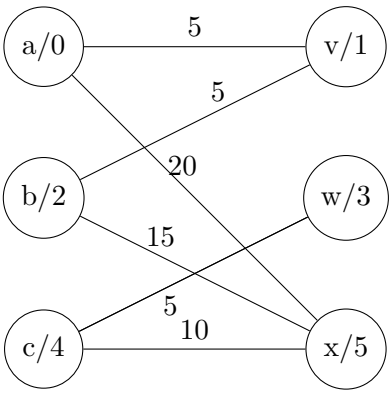
For the matching of $M = \{01\}$ we have a weight sum of $w_{01} = 5$. We then build the associated directed graph D (even more Easter Bunny Vomit). Within this graph there is a shortest path! The Path $C = \{50, 01, 12\}$ has a weight sum of $-20 + 5 - 5 = -20$. We then compute $M' = M \oplus P = \{05, 12\}$ and see the weight sum of this matching is $20 + 5 + 0 = 25$ (our value went up by 20). Then we start the process again! That is, we construct a directed graph, look for shortest path, compute the symmetric difference/XOR.

```
Maximum Weighted Bipartite Matching - Option 2
1.  M = [] (Or M can be initialized with a random edge)
2.  D = Directed Graph from M
3.  While |M| ≤ n/2
4.      Find shortest augmenting path P in D
4.      M = M ⊕ P
5.      Update D
6.  Return M
```

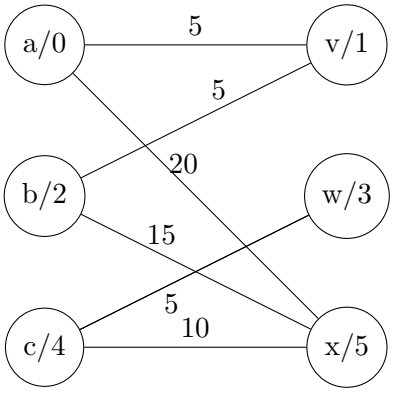
Finally, it is your turn to play! Using matching M' construct the directed graph D' from M' and identify the shortest augmenting path, P', and compute $M'' = D' \oplus C'$.



New Matching: $M' = \{05, 12\}$
Weight Sum: $5 + 20 = 25$



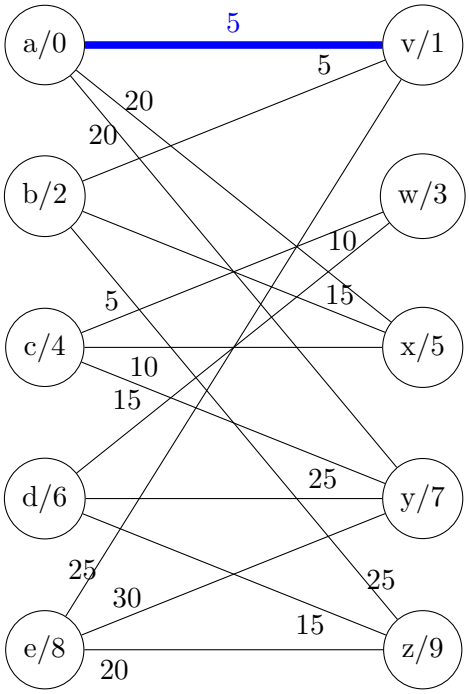
Directed Graph: D'
Path: $P' =$



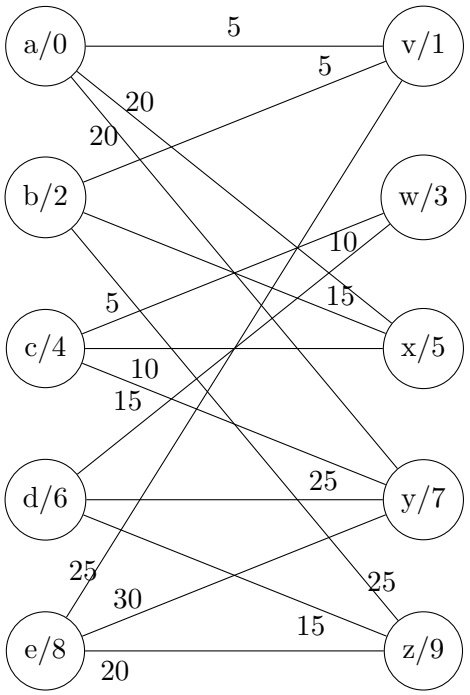
New Matching: $M'' =$
Weight Sum:

After this pass you should be at the maximal matching with weight sum of 30. If not, go crazy and run it again. I would recommend giving it another go from the start with a different first cycle (e.g., $C = \{23, 34, 45, 52\}$. If you are feeling extra excited, give it a full implementation and run it on some proper bipartite graphs (read: run it on some big ole systems) - you know you want to (well if it was not the end of the semester you would want to).

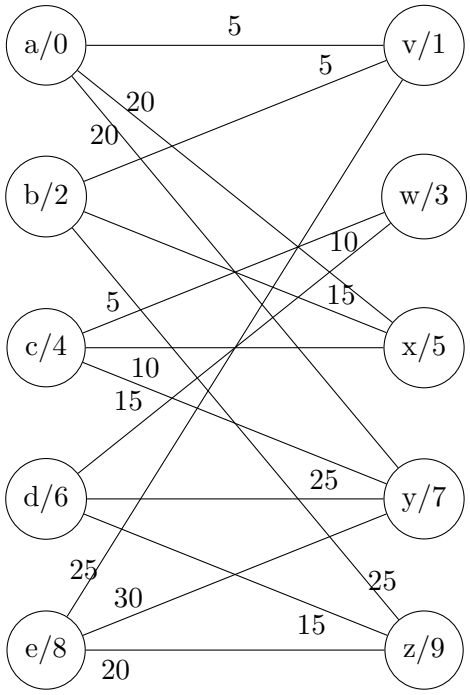
Problem 10. Run the Weighted Matching (Option 2) algorithm on the following bi-partite graph. Since the system is small, you may 'eyeball' the shortest negative augmenting paths - i.e., you do not need to run the Bellman-Ford Algorithm (but know that you could!). In the following I did not include edges with weights of 0 (but, know the 'edge' exists if you ever need it).



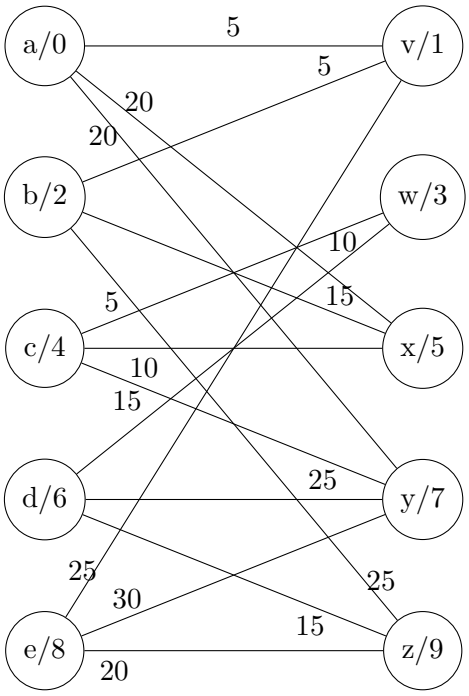
Matching: $M = \{01\}$
Weight Sum: 5



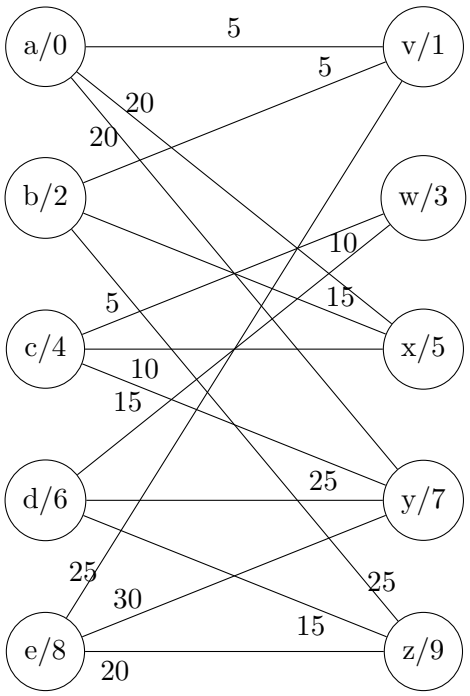
Direct Graph D:
Path P:



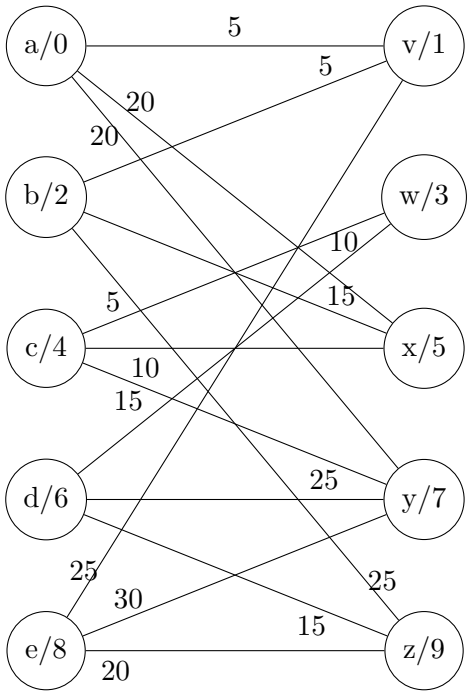
New Matching $M' = M \oplus P$
Weight Sum:



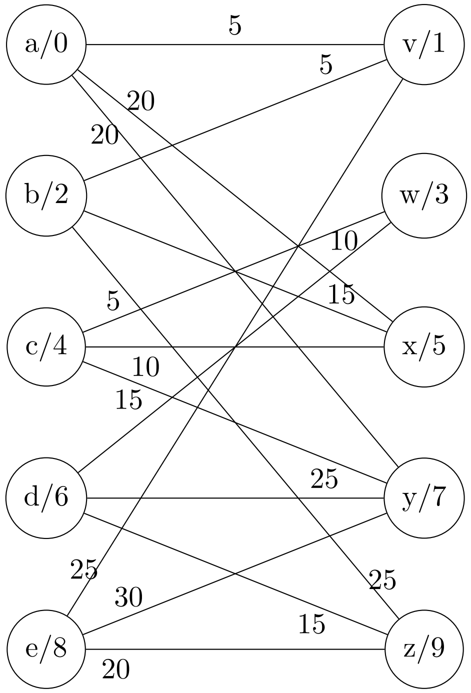
Matching:
Weight Sum:



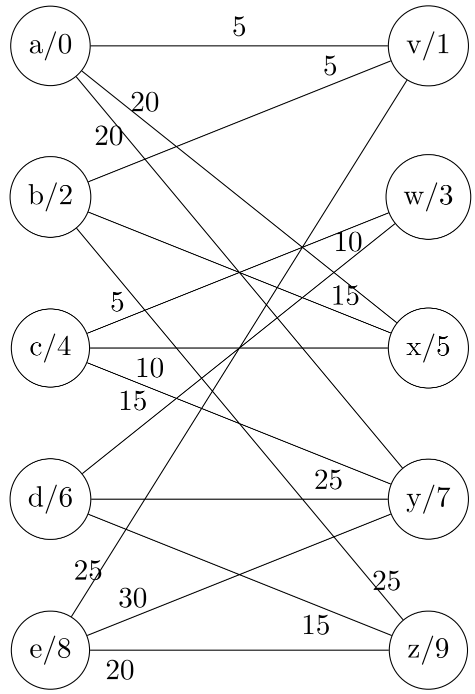
Direct Graph:
Path:



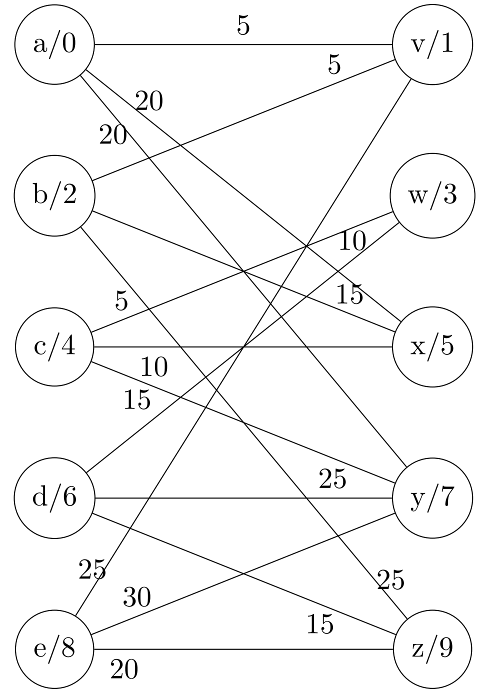
New Matching:
Weight Sum:



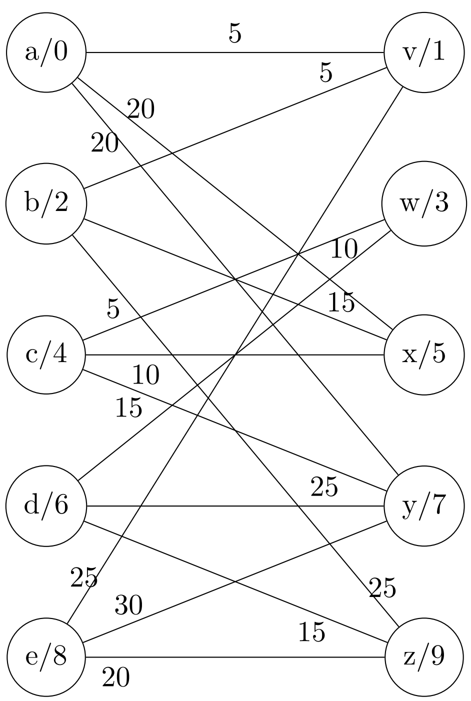
Matching:
Weight Sum:

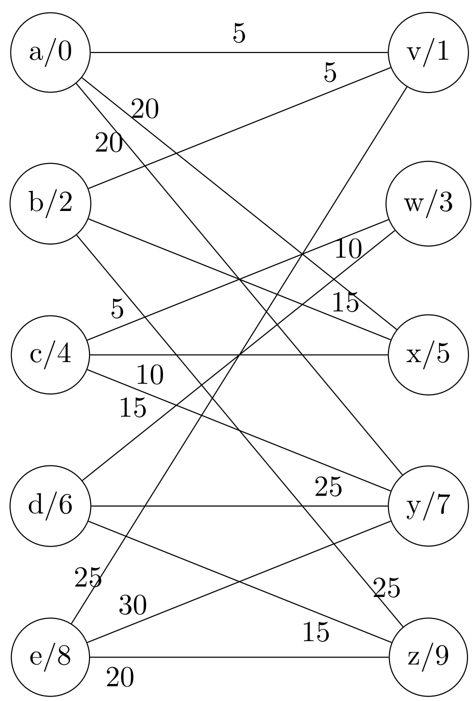


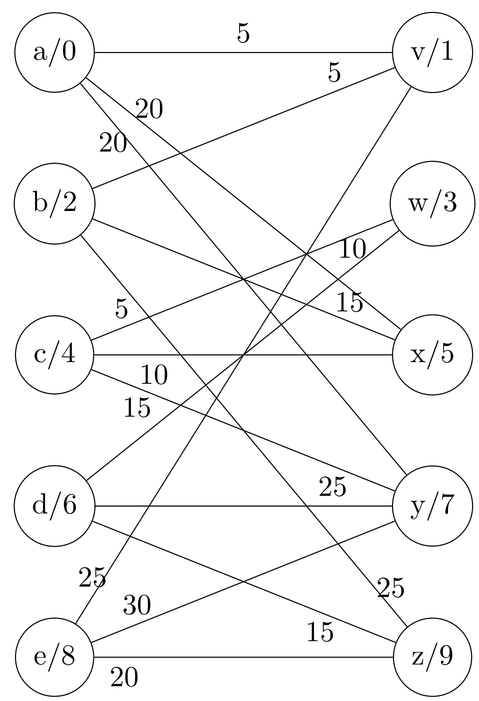
Direct Graph:
Path:



New Matching:
Weight Sum:







Problem 11. Thanks for a fun semester! (Thanks for your flexibility this semester and changes to keep us under Covid room capacity.) How are things going in the course? Provide some comments about how the course is going so far (e.g., Activities/Project, Homework/Exams, Notes, Videos, D2L).

Problem 12. (Bonus) So, you like Games, Randomizaton, and Weird Stories! Well, I have some goodies for you. Below is a collection of classic questions in game theory that can easily be framed in the context of procedures we have seen - wee! Some of the following are stolen from Wikipedia other are in my brain (and I do not know the official references)

a. Prisoner Hat Riddle: You and nine other individuals have been captured by super intelligent alien overlords. The aliens think humans look quite tasty, but their civilization forbids eating highly logical and cooperative beings. Unfortunately, they're not sure whether you qualify, so they decide to give you all a test. Through its universal translator, the alien guarding you tells you the following: You will be placed in a single-file line facing forward in size order so that each of you can see everyone lined up ahead of you. You will not be able to look behind you or step out of line. Each of you will have either a black or a white hat on your head assigned randomly, and I won't tell you how many of each color there are.

   When I say to begin, each of you must guess the color of your hat starting with the person in the back and moving up the line. And don't even try saying words other than black or white or signaling some other way, like intonation or volume; you'll all be eaten immediately. If at least nine of you guess correctly, you'll all be spared. You have five minutes to discuss and come up with a plan, and then I'll line you up, assign your hats, and we'll begin. Can you think of a strategy guaranteed to save 9 people (With a 50/50 chance of saving the 10th)?

b. Pirate Game: There are five rational pirates (in strict order of seniority A, B, C, D and E) who found 100 gold coins. They must decide how to distribute them.

   The pirate world's rules of distribution say that the most senior pirate first proposes a plan of distribution. The pirates, including the proposer, then vote on whether to accept this distribution. If the majority accepts the plan, the coins are dispersed and the game ends. In case of a tie vote, the proposer has the casting vote. If the majority rejects the plan, the proposer is thrown overboard from the pirate ship and dies, and the next most senior pirate makes a new proposal to begin the system again. The process repeats until a plan is accepted or if there is one pirate left.

   Pirates base their decisions on four factors. First of all, each pirate wants to survive. Second, given survival, each pirate wants to maximize the number of gold coins he receives. Third, each pirate would prefer to throw another overboard, if all other results would otherwise be equal. And finally, the pirates do not trust each other, and will neither make nor honor any promises between pirates apart from a proposed distribution plan that gives a whole number of gold coins to each pirate.

Problem 13. (Bonus) So, you like everything! Well, I have some goodies for you! Since most of my summer travels/conferences have been canceled for the summer, I will be around doing research most of the summer. If you want to dig into some cool problems (or just chat computation/cryptography/CS/Math) shoot me a message and we can get something started.