CS 70 Discrete Mathematics and Probability Theory
Spring 2020

HW 2

1 Gaussian Elimination

Gaussian Elimination for solving systems of linear equations is an algorithm that should be very familiar to you. We will have need of it later in the course, and so it is nice for you to actually prove some basic properties that you already know that Gaussian Elimination satisfies.

We will prove the termination of Gaussian Elimination Algorithm for input augmented matrix $A \in \mathbb{R}^{n\times(n+1)}$. Let $A[i][j]$ denote the entry of A in the ith row and jth column. Let $A[i]$ denote the ith row of A, where i can range from 1 to n.

The first equation of the system of linear equations is:

$$A[1][1]\cdot x_1 + A[1][2]\cdot x_2 + \ldots + A[1][n]\cdot x_n = A[1][n+1]$$

In general, for $1 \le i \le n$, the ith equation of the system is:

$$A[i][1]\cdot x_1 + A[i][2]\cdot x_2 + \ldots + A[i][n]\cdot x_n = A[i][n+1]$$

Here is the algorithm:

Initialize $r = 1$, $c = 1$.

While $c \le n$ and $r \le n$, repeat the following:

1. Pick a row $A[i]$ such that $i \ge r$ and $A[i][c] \ne 0$, and swap it with row $A[r]$. If no such $A[i]$ exists, increment c by 1 and skip the following three steps.
2. Scale row $A[r]$ by multiplying it by $\frac{1}{A[r][c]}$, so $A[r][c]$ becomes a 1.
3. For each value of $r < t \le n$, scale the updated row $A[r]$ by $-A[t][c]$ and add it to row $A[t]$ such that $A[t][c] = 0$. In other words $A[t] = A[t] - A[t][c]A[r]$.
4. Increment c by 1 and increment r by 1

At this point, the matrix is in an upper-triangular form (you will have to prove this later in this problem). There are many things that could happen.

If $r < n+1$, then we terminated by running out of rows to work on. In this case, it turns out you can figure out if there are no valid solutions or infinitely many solutions.

If $r = c = n+1$, then we continue onward to the backward pass of this algorithm, known as backsubstitution:

Initialize $r = n$.
Directly inspect the matrix to identify the value of x_n as $A[n][n+1]$.
While $r > 1$:

1. Use the values $x_r \dots x_n$ and plug them into the linear equation corresponding to $A[r-1]\vec{x} = A[r-1][n+1]$.
2. Solve the resulting linear equation for x_{r-1}
3. Decrement r by 1.

This gives us the unique solution if one exists.

(a) Prove using induction that the forward pass of this algorithm always terminates for augmented matrix $A \in \mathbb{R}^{n\times(n+1)}$.

(b) Prove using induction that the backward pass of the algorithm terminates if reached.

(c) Prove using induction that the forward pass of the algorithm terminates with an upper-triangular matrix — all the entries below the diagonal are zero.

(d) Prove that the algorithm is correct for augmented matrix $A \in \mathbb{R}^{n\times(n+1)}$ for the case when a unique solution exists. To do this, you should first prove a lemma that shows that any solution to the original system of equations remains a solution to the modified system of equations at all iterations of the forward pass of the algorithm, and vice-versa: all solutions of the modified system of equations at all iterations are valid solutions to the original system of equations.

Solution:

(a) We will show that c exceeds n in a finite number of steps. Proceed by induction on the value of c.
Base Case: If the algorithm reaches $c = n+1$, the algorithm must terminate.
Induction Hypothesis: Assume if the algorithm reaches $c = k$, the algorithm will terminate.
Inductive Step: Given that the algorithm terminates if it reaches $c = k$, we must show it will terminate from a state where $c = k-1$.
Indeed, from a state where $c = k-1$, the first step of the next iteration can increment c, and will result in a state where $c = k$. If the first step does not increment c, then it must be the case that the fourth step is reached. The fourth step increments c, so we can conclude that from a state where $c = k-1$, the algorithm can reach a state where $c = k$. Applying the Induction Hypothesis, we know that the algorithm will terminate from there. Therefore, the algorithm will terminate from a state where $c = k-1$.

By the principle of induction, we can conclude that the algorithm will proceed from $c = 1$ to $c = n+1$ in n iterations of the while loop. Upon reaching $c = n+1$, the algorithm will terminate per the condition of the while loop (base case).

(b) By the time the forward pass completes, the matrix is in upper triangular form. We can perform backsubstitution and are guaranteed it will complete. We can illustrate this by using strong induction on the known variables.
Base Case: x_n can be solved for by inspection of the bottom row. Either x_n can be anything, if the corresponding linear equation is trivially true, or $x_n = a_{n,n+1}/a_{n,n}$.
Inductive Hypothesis: Assume we have solved for $x_k, l \leq k \leq n$.
Inductive Step: Given the inductive hypothesis, we can calculate x_{l-1} as follows. First, observe since the system is in upper triangular form, the corresponding linear equation only depends on $x_k, l-1 \leq k \leq n$. Since we know $x_k, l \leq k \leq n$, we can rearrange this equation and isolate x_{l-1}.

By the principle of induction, we can conclude that the backward pass will determine the value for a new unknown in each step, and terminate since there are only n unknowns to calculate.

(c) We use induction on r to prove the following more concrete claim: " If r is incrementing from k to $k+1$, then the submatrix A[k+1:n][1:c] must be all 0s"
Base Case: $r = 1$ going to $r = 2$, the algorithm must have zeros in all columns to the left of A[1][c], and must also have zerod out all entries below $A[1][c]$. This means $A[1+1:n][1:c]$ must be all 0s, as desired.
Inductive Hypothesis: $r = k-1$ to $r = k$, assume that $A[k+1:n][1:c]$ must be all 0s.
Inductive Step: Let c' be the value of c at the beginning of the inductive step. By following the body of the while loop, we know that as we proceed to step $k+1$, we already have $A[k+1:n][1:c']$ as all zeros. $A[k+2:n][1:c']$ is a subpart of this submatrix which is therefore all 0s as well. Thus, to reach the proposition, we need only ensure that each column from c' to the final value of c before the row is incremented, must be set to 0. But this is precisely step 3 of the algorithm. Thus, we have $A[k+2:n][1:c]$ as all 0s, as desired.

Now that we have show this statement, observe the following. First, $c \geq r$, since r is never incremented without c and both start off at 1. Futhermore, once r is incremented to $r+1$, the row r, as well as the rows above, are not modified, and the same can be said for entries in columns to the left of c. Putting these facts together, the algorithm will terminate with the union of the zero submatrices, which means the matrix resulting from the forward pass will have a lower triangle of all 0s.

(d) Lemma: Solutions of the linear system are preserved at all iterations.
Proof: The only operations that appear to modify the system are in steps 2 and 3 of the forward pass. In step 2, we are scaling both sides of a single equation with a nonzero scalar. This does not change the solutions to the equation, and therefore does not change the solutions to the linear system. In step 3, we are adding nonzero multiples of equations to one another. This also does not change the solutions of the system of linear equations, since its a combination of

the previous case and the fact that if two equations share a solution, then adding one to another also has the same solution.

Since the solutions are preserved across iterations of the algorithm, we just need to make sure the algorithm computes a valid solution to the augmented matrix after the forward pass is complete. After the forward pass, if it is the case that there is a unique solution, then it must be the case that $r = c = n+1$. The backward pass then not only terminates, but is correct (same proof as in (b)), as we can read off x_n and use it to solve for x_{n-1} and so on. This means that the solution to the original system must also be correct, since the solution set is the same. Thus, the algorithm is correct in the case that a unique solution exists.

2 Strong Induction

Use strong induction to show that for all natural numbers n there exist natural numbers x, y, and z such that $6^n = x^2 + y^2 + z^2$. *Hint: You may want to write out a decomposition for n=1, 2, 3*

Solution: In this induction we need to go two steps back in the induction step, so we verify the validity of the claim for the first two cases. In other words, for proposition

$$P(n) = (\forall n \in \mathbb{N}, \exists x,y,z \in \mathbb{N})(6^n = x^2 + y^2 + z^2),$$

we show that for all natural numbers $k > 1$, $P(k+1)$ follows from $P(k-1)$, and that this requires us to prove both P(0) and P(1) as base cases (else, if we only proved P(0), the induction would only hold for even values of n).

- Base case : $n = 0, n = 1$:

  $n = 0$, we choose $x = 1$, $y = 0$ and $z = 0$.

  $$6^0 = 1 = 1^2 + 0^2 + 0^2 = 1.$$

  $P(0)$ is true.

  $n = 1$, we choose $x = 2$, $y = 1$ and $z = 1$.

  $$6^1 = 6 = 2^2 + 1^2 + 1^2 = 4 + 1 + 1 = 6.$$

  $P(1)$ is true.

- Inductive Hypothesis: For some arbitrary $k - 1 > 0$

  Assume there exist natural numbers x, y, and z such that: $6^{k-1} = x^2 + y^2 + z^2$.

  For this we show that $P(k-1)$ implies $P(k+1)$.

- Inductive Step: We show there exist natural numbers x', y', and z' such that: $6^{k+1} = x'^2 + y'^2 + z'^2$

$P(k+1)$:

$$\begin{aligned} 6^{k+1} &= 6^2 6^{k-1} \\ &= (x^2+y^2+z^2)6^2 \qquad \text{(Inductive Hypothesis)} \\ &= (6x)^2+(6y)^2+(6z)^2. \end{aligned}$$

We choose $x' = 6x$, $y' = 6y$, and $z' = 6z$.

Thus,

$$6^{k+1} = x'^2 + y'^2 + z'^2.$$

We showed that for all natural numbers $k > 1$, $P(k+1)$ follows from $P(k-1)$ and not from $P(k)$. This requires us to prove both P(0) and P(1) as base cases. If we only proved P(0), the induction would only hold for even values of n.

Hence, we proved by strong induction that for all natural numbers n there exist natural numbers x, y, and z such that $6^n = x^2+y^2+z^2$.

3 Induction on Reals

Induction is always done over objects like natural numbers, but in some cases we can leverage induction to prove things about real numbers (with the appropriate mapping). We will attempt to prove the following by leveraging induction and finding an appropriate mapping.

Bob the Bug is on a window extending to the ground, trying to escape Sally the Spider. Sally has built her web from the ground to 2 inches up the window. Every second, Bob jumps 1 inch vertically up the window, then loses grip and falls to half his vertical height.

Prove that no matter how high Bob starts up the window, he will always fall into Sally's net in a finite number of seconds.

Solution: The basic idea is: First, prove directly that Bob will be netted if he starts ≤ 3 inches above the ground. Then, prove the inductive step: Given he dies if he starts $\leq n$ inches, show he also dies if starts $\leq n+1$ inches above the ground.
When working with natural numbers, we use induction to prove certain claims for discrete values (natural numbers), $n, n+1, \ldots$ and so on. In this case, since we want our claim to hold true for a subset of the real numbers, we have assumed our claim to hold true for all real numbers $\leq n$ - thereby creating our mapping. Therefore, if we can prove that for every natural number n, falling from height $\leq n$ means Bob will fall into the net, we will prove our claim for the entire subset of real numbers we want.
Let Bob's height up the window at time i be $x_i \in \mathbb{R}$. He starts at height $x_0 > 0$, and we have $x_{i+1} = (x_i+1)/2$.

Proof: We prove this by induction on n.
Base Case: If he starts at height $x_0 \leq 3$, $x_1 = (x_0+1)/2 \leq 2$. So he will fall into the net in finite time (within the next second, in fact).

Inductive Hypothesis: Assume he falls into the net in finite time if he starts $x_0 \leq n$ inches above the ground.
Note: We are proceeding to do strong induction here as we are assuming our inductive hypothesis to hold true for all real numbers less than or equal to n which includes all the natural numbers less than or equal to n.

Inductive Step: We want to show: If he starts $x_0 \leq n+1$ inches above the ground, then he will also be netted in finite time.
If he starts at $x_0 \leq 3$ inches, the base case directly applies. Otherwise we have (for $x_0 > 3$):

$$\Delta = x_0 - x_1 = x_0 - (x_0+1)/2 = x_0/2 - 1/2 > 1$$

In other words, for all $x_0 > 3$, Bob falls more than 1 inch total in the first second. Therefore if $x_0 \leq n+1$, we have $x_1 = x_0 - \Delta \leq n+1-\Delta \leq n$. So Bob will be $x_1 \leq n$ inches high after 1 second after which point we know (by the inductive hypothesis) he dies in finite time.

4 Nothing Can Be Better Than Something

In the stable matching problem, suppose that some jobs and candidates have hard requirements and might not be able to just settle for anything. In other words, in addition to the preference orderings they have, they prefer being unmatched to being matched with some of the lower-ranked entities (in their own preference list). We will use the term entity to refer to a candidate/job. A matching could ultimately have to be partial, i.e., some entities would and should remain unmatched.

Consequently, the notion of stability here should be adjusted a little bit to capture the autonomy of both jobs to unilaterally fire employees and employees to just walk away. A matching is stable if

- there is no matched entity who prefers being unmatched over being with their current partner;
- there is no matched/filled job and unmatched candidate that would both prefer to be matched with each other over their current status;
- similarly, there is no unmatched job and matched candidate that would both prefer to be matched with each other over their current status;
- there is no matched job and matched candidate that would both prefer to be matched with each other over their current partners; and
- there is no unmatched job and unmatched candidate that would both prefer to be with each other over being unmatched.

(a) Prove that a stable pairing still exists in the case where we allow unmatched entities.

(HINT: You can approach this by introducing imaginary/virtual entities that jobs/candidates "match" if they are unmatched. How should you adjust the preference lists of jobs/candidates, including those of the newly introduced imaginary ones for this to work?)

(b) As you saw in the lecture, we may have different stable matchings. But interestingly, if an entity remains unmatched in one stable matching, it/she must remain unmatched in any other stable matching as well. Prove this fact by contradiction.

Solution:

(a) Following the hint, we introduce an imaginary mate (let's call it a robot) for each entity. Note that we introduce one robot for each entity, i.e. there are as many robots as there are candidates+jobs. For simplicity let us say each robot is owned by the entity we introduce it for.

Each robot prefers its owner, i.e. it puts its owner at the top of its preference list. The rest of its preference list can be arbitrary. The owner of a robot puts it in their preference list exactly after the last entity they are willing to match with. i.e. owners like their robots more than entities they are not willing to match, but less than entities they like to match. The ordering of entities who someone does not like to match as well as robots they do not own is irrelevant as long as they all come after their robot.

To illustrate, consider this simple example: there are three jobs $1,2,3$ and three candidates A,B,C. The preference lists for jobs is given below:

Job	Preference List
1	$A > B$
2	$B > A > C$
3	C

The following depicts the preference lists for candidates:

Candidate	Preference List
A	1
B	$3 > 2 > 1$
C	$2 > 3 > 1$

In this example, 1 is willing to match A and B and it likes A better than B, but it'd rather be single than to be with C. On the other side B has a low standard and does not like being single at all. She likes 3 first, then 2, then 1 and if there is no option left she is willing to be forced into singleness. On the other hand, A has pretty high standards. She either matches 1 or remains single.

According to our explanation we should introduce a robot for each entity. Let's name the robot owned by entity X as R_X. So we introduce job robots R_A, R_B, R_C and candidate robots R_1, R_2, R_3. Now we should modify the existing preference lists and also introduce the preference lists for robots.

According to our method, 1's preference list should begin with its original preference list, i.e. $A > B$. Then comes the robot owned by 1, i.e. R_1. The rest of the ordering, which should include C and R_2, R_3 does not matter, and can be arbitrary.

For B, the preference list should begin with $3 > 2 > 1$ and continue with R_B, but the ordering between the remaining robots (R_A and R_C) does not matter.

What about robots' preference lists? They should begin with their owners and the rest does not matter. So for example R_A's list should begin with A, but the rest of the entities/robots (B, C, R_1, R_2, and R_3) can come in any arbitrary order.

So the following is a list of preference lists that adhere to our method. There are arbitrary choices which are shown in bold (everything in bold can be reordered within the bold elements).

Job	Preference List
1	$A > B > R_1 > \mathbf{3} > \mathbf{R_3} > \mathbf{R_2}$
2	$B > A > C > R_2 > \mathbf{R_1} > \mathbf{R_3}$
3	$C > R_3 > \mathbf{R_1} > \mathbf{R_3} > \mathbf{A} > \mathbf{B}$
R_A	$A > \mathbf{B} > \mathbf{C} > \mathbf{R_1} > \mathbf{R_2} > \mathbf{R_3}$
R_B	$B > \mathbf{R_1} > \mathbf{R_2} > \mathbf{R_3} > \mathbf{A} > \mathbf{C}$
R_C	$C > \mathbf{A} > \mathbf{R_2} > \mathbf{B} > \mathbf{R_1} > \mathbf{R_3}$

and the following depicts the preference lists for candidates and job robots:

Candidate	Preference List
A	$1 > R_A > \mathbf{3} > \mathbf{R_B} > \mathbf{2} > \mathbf{R_C}$
B	$3 > 2 > 1 > R_B > \mathbf{R_C} > \mathbf{R_A}$
C	$2 > 3 > 1 > R_C > \mathbf{R_A} > \mathbf{R_B}$
R_1	$1 > \mathbf{R_B} > \mathbf{2} > \mathbf{R_C} > \mathbf{3} > \mathbf{R_A}$
R_2	$2 > \mathbf{R_A} > \mathbf{R_C} > \mathbf{1} > \mathbf{3} > \mathbf{R_B}$
R_3	$3 > \mathbf{2} > \mathbf{1} > \mathbf{R_A} > \mathbf{R_C} > \mathbf{R_B}$

Now let us prove that a stable pairing between robots and owners actually corresponds to a stable pairing (with singleness as an option). This will finish the proof, since we know that in the robots and owners case, the propose and reject algorithm will give us a stable matching.

It is obvious that to extract a pairing without robots, we should simply remove all pairs in which there is at least one robot (two robots can match each other, yes). Then each entity which is not matched is declared to be single. It remains to check that this is a stable matching (in the new, modified sense). Before we do that, notice that an entity will never be matched with another entity's robot, because if that were so it and its robot would form a rogue couple (the robot prefers its owner, and the owner actually prefers their robot more than other robots).

(a) No one who is paired would rather break out of their pairing and be single. This is because if that were so, that entity along with its robot would have formed a rogue couple in the original pairing. Remember, the robot prefers its owner more than anything, so if the owner likes it more than their mate too, they would be a rogue couple.

(b) There is no rogue couple. If a rogue couple j and c existed, they would also be a rogue couple in the pairing which includes robots. If neither j nor c is single, this is fairly obvious. If one or both of them are single, they prefer the other entity over being single, which in the robots scenario means they prefer being with each other over being with their robot(s) which is their actual match.

This shows that each stable pairing in the robots and entities setup gives us a stable pairing in the entities-only setup. It is noteworthy that the reverse direction also works. If there is a stable pairing in the entities-only setup, one can extend it to a pairing for robots and entities setup by first creating pairs of owners who are single and their robots, and then finding an arbitrary stable matching between the unmatched robots (i.e. we exclude everything other than the unmatched robots and find a stable pairing between them). To show why this works, we have to refute the possibility of a rogue pair. There are three cases:

(a) A entity-entity rogue pair. This would also be rogue pair in the entities-only setup. The entities prefer each other over their current matches. If their matches are robots, that translates to them preferring each other over being single in the entities-only setup.

(b) A entity-robot rogue pair. If the entity is matched to their robot, our pair won't be a rogue pair since a entity likes their robot more than any other robot. On the other hand if the entity is matched to another entity, they prefer being with that entity over being single which places that entity higher than any robot. Again this refutes the entity-robot pair being rogue.

(c) A robot-robot rogue pair. If both robots are matched to other robots, then by our construction, this won't be a rogue couple (we explicitly selected a stable matching between left-alone robots). On the other hand, if either robot is matched to a entity, that entity is its owner, and obviously a robot prefers its owner more than anything, including other robots. So again this cannot be a rogue pair.

This completes the proof.

(b) We will perform proof by contradiction. Assume that there exists some job j_1 who is paired with a candidate c_1 in stable pairing S and unpaired in stable pairing T. Note that this means j_1 and c_1 both prefer to be with each other over being single. Since T is a stable pairing and j_1 is unpaired, c_1 must be paired in T with a job j_2 whom she prefers over j_1. (If c_1 were unpaired or paired with a job she does not prefer over j_1, then (j_1, c_1) would be a rogue couple in T, which is a contradiction.)

Since j_2 is paired with c_1 in T, it must be paired in S with some candidate c_2 whom j_2 prefers over c_1. This process continues (c_2 must be paired with some j_3 in T, j_3 must be paired with some c_3 in S, etc.) until all entitys are paired. Indeed, the last candidate c_n needs a partner in T and cannot be single (for the same reasons that c_1, c_2, and all the candidates before her need partners in T who they like better than their partners in S, to maintain stability). At this point, j_1 will be the only unpaired job, but to maintain the stability of T, we require j_1 to be paired in T with c_n. Yet we assumed j_1 was single, so we have reached a contradiction. Therefore, our assumption must be false, and there cannot exist some job who is paired in a stable pairing S and unpaired in a stable pairing T. A similar argument can be used for candidates.

Since no job or candidate can be paired in one stable pairing and unpaired in another, every job or candidate must be either paired in all stable pairings or unpaired in all stable pairings.

Here is another possible proof:

We know that some job-optimal stable pairing exists. Call this pairing M. We first establish two lemmas.

Lemma 1. If a job is single in job-optimal pairing M, then it is single in all other stable pairings.

Proof. Assume there exists a job that is single in M but not single in some other stable pairing M'. Then M would not be a job-optimal pairing, so this is a contradiction.

Lemma 2. If a candidate is paired in job-optimal pairing M, she is paired in all other stable pairings.

Proof. Assume there exists a candidate that is paired in M but single in some other stable pairing M'. Then M would not be candidate-pessimal, so this is a contradiction.

Let there be k single jobs in M. Let M' be some other stable pairing. Then by Lemma 1, we know single jobs in M' will be greater than or equal to k. We also know that there are $n-k$ paired jobs and candidates in M. Then by Lemma 2, we know that the number of paired candidates in M' will be greater than or equal to $n-k$.

Now, we want to prove that if a job is paired in M, then it is paired in every other stable pairing. We prove this by contradiction. Assume that there exists a job m that is paired in M but is single in some other stable pairing M'. Then there must be strictly greater than k single jobs in M', and thus strictly greater than k single candidates in M'. Since there are strictly greater than k single candidates in M', there must be strictly less than $n-k$ paired candidates in M'. But this contradicts that the number of paired candidates in M' will be greater than or equal to $n-k$.

We also have to prove that if a candidate is single in M, then she must be single every other stable pairing. We again prove this by contradiction. Assume that there exists a candidate w that is single in M and paired in some other stable pairing M'. Then there are strictly greater than $n-k$ paired candidates in M', which means there are strictly greater than $n-k$ paired jobs in M'. This means there must be strictly less than k single jobs in M'. But this contradicts that the number of single jobs in M' will be greater than or equal to k.

Since we have proved both 1) If a job is single in M then it is single in every other stable pairing and 2) If a job is paired in M then it is paired in every other stable pairing (note that the contrapositive of this is if a job is single in any other stable pairing, then this job is single in M), we know that a job is single in M if and only if it is single in every other stable pairing.

Similarly, since we have proved both 1) If a candidate is single in M then she is single in every other stable pairing and 2) If a candidate is paired in M then she is paired in every other stable pairing, we know that a candidate is single in M if and only if she is single in every stable pairing. Thus we have proved that if a entity is single in one stable pairing, it is single in every stable pairing.

5 Inductive Charging Revisited

Recall the setup from the previous HW. There are n cars on a one-way circular track. Among all of them, they have exactly enough fuel (in total) for one car to exactly circle the track. Each car burns exactly the same amount of fuel per unit of distance. Two cars at the same location may transfer fuel between them.

(a) Prove that there exists one car that can circle the track, by gathering fuel from other cars along the way. (That is, one car moving and all others stopped). Hint: Use induction and the previous part of this problem from HW1.

(HINT: The problem as written is about these isolated points called "cars" that have fuel associated with them. These exist at the beginning of your thinking, however they're not what you have when you have already made some progress. It can be useful to think about generalizing points to segments that have a starting and end point associated with them, together with some remaining fuel. Inside each segment, there can be a story of how you got here.)

Solution:

(a) The proof is non-constructive, but can be converted into an algorithm as follows: First, orient all cars clockwise. Every time some car at point A can reach some car at point B, collapse them into one car at A (oriented clockwise), with the effective fuel of both cars (at A and B). Keep collapsing in this way until we have only car at some point P (why is this collapsing always possible?). The original car at P can complete the track.

The proof is non-constructive, but can be converted to an algorithm as follows: Every time some Car A can reach some Car B, collapse them into one car at A, with effective fuel A+B. Keep collapsing until we have only one car (why is this always possible?). This car can complete the track.

Since the initial orientation of cars is not fixed, we may orient all cars to point in the clockwise direction. Then, we will prove a stronger statement: For n cars oriented clockwise, which in total have enough fuel for one car to circle the track, there always exists one car that can circle the track, traveling clockwise.

The key idea is: Let Car A be a car that can reach the next clockwise car (Car B). Notice that since Car A can reach Car B, we can consider Car A to have the "effective fuel" of cars A+B, and remove Car B altogether (because Car A can travel to Car B, pick up Car B's fuel, and continue). This modified setup has $n-1$ cars oriented clockwise, with enough fuel in total to

complete the track (total fuel was unchanged in the transform).
Then by inductive hypothesis, there is one car (Car X) here that can complete the track, traveling clockwise. Simply use this car's clockwise path for the original setup (n cars), making sure to unwrap "effective Car A" into "Car A, then Car B". Notice that strengthening the hypothesis to prove a **clockwise** path exists was necessary, to ensure there exists a path for Car X that reaches Car A before Car B (otherwise, our "unwrapping" step would fail).

The formal proof that follows illustrates the concept of a **reduction**, which is used in many areas of EECS (both theory and practice). Reductions are like calling subroutines in programming: if you have a new problem, you first convert it to a problem you've already solved, call the subroutine for that, then convert the solution back.

Base Case: $n = 1$: If one car oriented clockwise has exactly enough fuel to circle the track... it has enough fuel to circle the track, clockwise.
Inductive Hypothesis: Assume the statement is true for k cars: If k cars oriented clockwise have exactly enough total fuel for one car to circle the track, then there exists one car that can circle the track, traveling clockwise (by gathering fuel from the other stopped cars).
Inductive Step: We want to prove the statement for $k+1$ cars. Let a particular set of $k+1$ cars be $C = \{c_1, c_2, \ldots, c_{k+1}\}$, ordered clockwise. We know from part (a) that there exists at least one car which can reach the next clockwise car along the track. Without loss of generality
[1]

, let c_1 be such a car (which can reach c_2). Let us define a new car $\widetilde{c}_1$ with the same initial location and orientation as c_1, but with the fuel of c_1 and c_2 combined. Then, construct a new set of only k cars: $\widetilde{C} = \{\widetilde{c}_1, c_3, \ldots c_{k+1}\}$ (that is, taking the original set C, removing c_2, and replacing c_1 with $\widetilde{c}_1$). Let us use $P(C)$ to denote "the problem setup with cars in C on the track". Then $P(\widetilde{C})$ is the new problem setup.

By construction, the total fuel of cars in $\widetilde{C}$ is the exact same as the total fuel of cars in C. In particular, since C has exactly enough total fuel for one car to circle the track, so does $\widetilde{C}$. Further, cars in $\widetilde{C}$ are also oriented clockwise, by construction. Therefore we may use the inductive hypothesis to conclude that of the k cars in $P(\widetilde{C})$, one of them can circle the track, traveling clockwise.
We are not done yet! We must show that if one of the cars in $P(\widetilde{C})$ can complete the track clockwise, then one of the cars in the original problem $P(C)$ can circle the track clockwise as well.

Say car c_i in $P(\widetilde{C})$ can complete the track, traveling clockwise. Consider its path, as it visits subsequent cars, picks up their fuel, and continues:

$$\widetilde{T} = c_i \to c_{i+1} \to c_{i+2} \to \ldots \to \widetilde{c}_1 \to c_3 \to \ldots$$

At some point, it must visit $\widetilde{c}_1$ (the modified car), pick up its fuel, and continue to c_3 (since it travels clockwise). To construct a valid path for the original problem, we unwrap this step,

[1] Meaning: In this case, we assumed that c_1 was a car that could reach the next car, but the following steps would equivalently hold if, for example, c_5 was such a car. We could simply re-label cars clockwise, starting at c_5 instead of c_1. In general, the phrase "without loss of generality" means that we appear to be considering a specific case, but some symmetry of the problem (in this case, the modular nature of clockwise labeling) allows us to generalize beyond this specific case.

turning $\widetilde{c}_1 \to c_3$ into $c_1 \to c_2 \to c_3$:

$$T = c_i \to c_{i+1} \to c_{i+2} \to \ldots \to c_1 \to c_2 \to c_3 \to \ldots$$

Notice that the same car c_i in $P(C)$ can certainly execute path T up to reaching car c_1, since all steps are the same as in path $\widetilde{T}$. (Actually, since c_i and $\widetilde{c}_1$ are not necessarily different, we may have $c_i = \widetilde{c}_1$. In this case, by "the same car in $P(C)$", we mean c_1, not $\widetilde{c}_1$.)

Then, executing the steps $\widetilde{c}_1 \to c_3$ in $P(\widetilde{C})$ and the steps $c_1 \to c_2 \to c_3$ in $P(C)$ have exactly the same effect on the car's final location and fuel, and is feasible, by construction. Since the steps of T and $\widetilde{T}$ are the same after c_3, it is possible for one car in $P(C)$ to circle the track traveling clockwise, by executing path T.

Now we have shown how to transform a solution of the smaller, modified problem into a solution of the original problem, and our inductive step is complete.

6 Stable Matching for Classes!

Let's consider the system for getting into classes. For simplicity, we will start with the problem assigning students to lab sections first, since it is clear that there are a finite number of seats. We are given n students and m lab sections. Each lab section ℓ has some number, q_ℓ of seats, and we assume that the total number of students is larger than the total number of seats (i.e. $\sum_{\ell=1}^{m} q_\ell < n$) and so some students are going to end up with no lab. Each student ranks the m lab sections in order of preference, and the instructor for each lab ranks the n students. Our goal is to find an assignment of students to seats (one student per seat) that is *stable* in the following sense:

- There is no student-lab pair (s,ℓ) such that s prefers ℓ to her allocated lab section and the instructor for ℓ prefers s to one of the students assigned to ℓ. (This is like the stability criterion you have seen for jobs: it says there is no student-lab pair that would induce that lab instructor to kick out an existing student and take this new student instead.)
- There is no lab section ℓ for which the instructor prefers some unassigned student s to one of the students assigned to ℓ. (This extends the stability criterion to take account of the fact that some students are not assigned to labs.)

Note that this problem is almost the same as the Stable Matching problem for jobs/internships presented in the lecture note, with two differences: (i) there are more students than seats; and (ii) each lab section can have more than one seat.

Perhaps you will agree that this extended model is more realistic, even for the jobs context!

(a) Explain how to modify the propose-and-reject algorithm so that it finds a stable assignment of students to seats. [*Hint*: What roles of students/instructors will be in the propose-and-reject algorithm? What does "candidates have a job offer in hand (on a string)" mean in this setting?]

(b) State a version of the Improvement Lemma (see the Stable Matchings Lecture Note) that applies to your algorithm, and prove that it holds.

(c) Use your Improvement Lemma to give a proof that your algorithm terminates, that every seat is filled, and that the assignment your algorithm returns is stable.

(d) Let us consider the potential of students to want to swap lab sections, subject to global stability (i.e. the swap can't make the matching as a whole unstable). Either prove that your algorithm will not have any such swap requests or modify it to have no such swap requests and prove that the modified one will not have any students wanting to stably-swap sections.

Solution:

(a) We will extend the propose-and-reject algorithm given in the lecture notes. Students will play the role of jobs and discussion section instructors will play the role of candidates. Instead of keeping a single person as in the original algorithm, each discussion section instructor will keep a *waitlist* of size equal to its quota.

Note that there are other valid ways to modify the propose-and-reject algorithm such that a stable assignment is produced. One way is to have the instructors playing the role of jobs and the students playing the role of candidates, with the difference that now we have jobs proposing to up to q_u candidates each day. It is also possible to "expand" each instructor by the size of his or her quota by making q_u copies of instructor u and adding empty discussions for the unassigned students.

The extended procedure for students proposing and instructors keeping a waitlist works as follows:

- All students apply to their first-choice discussion section.
- Each discussion section u with a quota of q_u, then places on its waitlist the q_u applicants who rank highest (or all the applicants if there are fewer than q_u of them) and rejects all the rest.
- Rejected applicants then apply to their second-choice discussion section, and again each discussion section instructor u selects the top q_u students from among the new applicants AND those on its waitlist; it puts the selected students on its new waitlist, and rejects the rest of its applicants (including those who were previously on its waitlist but now are not).
- The above procedure is repeated until every applicant is either on a waitlist or has been rejected from every discussion section. At this point, each discussion section admits everyone on its waitlist.

(b) Improvement Lemma: Assume that discussion sections maintain their waitlist in decreasing order of their preference for the students on the lists. Let $\oplus$ be the "null" element, which we will use as a placeholder in waitlist positions not yet assigned to a student. For any discussion section u, let q_u be its quota and let $s_i^k \in \{\text{Students}\} \cup \{\oplus\}$ be the student in the i'th position on the waitlist after the k'th round of the algorithm. (If there are fewer than q_u students on the list, we fill the bottom of the list with $\oplus$'s.) Then for all i and k, the discussion section instructor likes s_i^{k+1} at least as much as s_i^k. (Here we assume that the discussion section instructor prefers any student to no student.)

In short, the lemma says that no position in the list ever gets worse for the discussion section instructor as the algorithm proceeds. As in the Lecture Notes, we use the Well-Ordering Principle and prove the lemma by contradiction:

Suppose that the jth day, where $j > k$, is the first counterexample where, for index $i \leq q_u$, discussion section u, has either nobody or some student $\hat{s}$ inferior to s_i^k. Then on day $j-1$, the instructor has some student $\tilde{s}$ on a string that they like at least as much as s_i^k. Following the algorithm, $\tilde{s}$ still "proposes" to the instructor of section u on day j since they said "maybe" the previous day. Therefore, the instructor has the choice of at least one student on the jth day, and his or her best option is at least as good as $\tilde{s}$, so the instructor would have chosen $\tilde{s}$ over $\hat{s}$. Therefore on day j, the instructor *does* have a student that they like at least as much as s_i^k in waitlist position $i \leq q_u$. This contradicts our initial assumption.

(c) First, the algorithm terminates. This follows by similar reasoning to the original propose-and-reject algorithm: in each round (except the last), at least one discussion section is crossed off the list of some rejected students.

Second, every seat is filled. Suppose some discussion section u has an unfilled seat at the end. Then the total number of students who applied to u must have been fewer than its quota q_u (since the Improvement Lemma in Part (b) ensures that a waitlist slot, once filled, will never later be unfilled). But the only students who do *not* apply to u are those who find a slot in some other discussion section. And since we are told that there are more students than seats, the number of students applying to u must be at least q_u.

Finally, the assignment is stable:

- Suppose there is a student-section pair (s,u) such that s prefers u to her discussion section u' in the final allocation. Then s must have proposed to u prior to proposing to u', and was rejected by u. Thus immediately after rejecting s, u must have had a full waitlist in which every student was preferred to s. By the Improvement Lemma, the same holds at all future times and hence at the end. Thus u does not prefer s to any of its assigned students, as required.
- Suppose s is a student left unassigned at the end. Consider any discussion section u. Since s must have applied to and have been rejected from all discussion sections, this holds in particular for u. Reasoning similar as in the previous case, using the Improvement Lemma, we see that in the final allocation, u prefers all its students to s. Therefore u does not prefer s to any of its assigned students, as required.

This concludes the proof that our algorithm finds a stable assignment when terminates.

(d) The intuition here is that the group proposing in the Propose-and-Reject algorithm is the group for which the resulting stable matchings are optimal. Thus, if we are using a Propose-and-Reject based algorithm with the students applying/proposing, then it is student-optimal and there should be no swap requests.
Assume toward contradiction that this were not the case. Then there is a pair (s,u) and a pair (s',u') such that s' prefers u to u' and s prefers u' to u. If this is the case, then s must have proposed to u' before u, and gotten rejected, and s' must have proposed to u before u', and

gotten rejected.

One of these must have happened first. WLOG let's say that s was rejected first. If s was rejected, that means that every single person in u''s waitlist must be preferred more by u' than s (by the Improvement Lemma). Similar logic applies for s' and u.
Now we must prove that u' has not rejected any student that it prefers to s throughout the algorithm, because if it has rejected some student s" that it prefers to s', then if s and s' swap, s" and u' would become a rogue pair. This implies that when s was rejected by u', the waitlist for u' on that day must be the exact students that are in u' at the end of the algorithm. This is because every student on the waitlist was preferred over s (which is why s was rejected in the first place), so none of them can be rejected by u'. That means that s' must have been on the waitlist for u' on the day s was rejected by u'.
However, we assumed earlier that s was rejected by u' before s' was rejected by u. u' is after u on the preference list for s', so there is no way for s' to have been on the waitlist for u' on the day that s was rejected by u'. Contradiction.

7 Grid Induction

Pacman is walking on an infinite 2D grid. He starts at some location $(i, j) \in \mathbb{N}^2$ in the first quadrant, and is constrained to stay in the first quadrant (say, by walls along the x and y axes). Every second he does one of the following (if possible):

(i) Walk one step down, to $(i, j-1)$.

(ii) Walk one step left, to $(i-1, j)$.

For example, if he is at $(5,0)$, his only option is to walk left to $(4,0)$; if Pacman is instead at $(3,2)$, he could walk either to $(2,2)$ or $(3,1)$.

Prove by induction that no matter how he walks, he will always reach $(0,0)$ in finite time. (*Hint*: Try starting Pacman at a few small points like $(2,1)$ and looking all the different paths he could take to reach $(0,0)$. Do you notice a pattern?)

Solution:

Following the hint, we notice that it seems as though Pacman takes $i+j$ seconds to reach $(0,0)$ if he starts in position (i, j), regardless of what path he takes. This would imply that he reaches $(0,0)$ in a finite amount of time since $i+j$ is a finite number. Thus, if we can prove this stronger statement, we'll also have proved that Pacman reaches $(0,0)$ in finite time. In order to simplify the induction, we will induct on the quantity $i+j$ rather than inducting on i and j separately.

Base Case: If $i+j=0$, we know that $i=j=0$, since i and j must be non-negative. Hence, we have that Pacman is already at position $(0,0)$ and so will take $0=i+j$ steps to get there.

Inductive Hypothesis: Suppose that if Pacman starts at position (i, j) such that $i+j=n$, he will reach $(0,0)$ in n seconds regardless of his path.

Inductive Step: Now suppose Pacman starts at position (i, j) such that $i+j=n+1$. If Pacman's first move is to position $(i-1, j)$, the sum of his x and y positions will be $i-1+j=(i+j)-1=n$. Thus, our inductive hypothesis tells us that it will take him n further seconds to get to $(0,0)$ no matter what path he takes. If Pacman's first move isn't to $(i-1, j)$, then it must be to $(i, j-1)$. Again in this case, the inductive hypothesis will tell us that Pacman will use n more moves to get to $(0,0)$ no matter what path he takes. Thus, in either case, we have that Pacman will take a total of $n+1$ seconds (one for the first move and n for the remainder) in order to reach $(0,0)$, proving the claim for $n+1$.

One can also prove this statement without strengthening the inductive hypothesis. The proof isn't quite as elegant, but is included here anyways for reference. We first prove by induction on i that if Pacman starts from position $(i,0)$, he will reach $(0,0)$ in finite time.

Base Case: If $i=0$, Pacman starts at position $(0,0)$, so he doesn't need any more steps. Thus, it takes Pacman 0 steps to reach the origin, where 0 is a finite number.

Inductive Hypothesis: Suppose that if $i=n$ (that is, if Pacman starts at position $(n,0)$), he will reach $(0,0)$ in finite time.

Inductive Step: Now say Pacman starts at position $(n+1,0)$. Since he is on the x-axis, he has only one move: he has to move to $(n,0)$. From the inductive hypothesis, we know he will only take finite time to get to $(0,0)$ once he's gotten to $(n,0)$, so he'll only take a finite amount of time plus one second to get there from $(n+1,0)$. A finite amount of time plus one second is still a finite amount of time, so we've proved the claim for $i=n+1$.

We can now use this statement as the base case to prove our original claim by induction on j.

Base Case: If $j=0$, Pacman starts at position $(i,0)$ for some $i \in \mathbb{N}$. We proved above that Pacman must reach $(0,0)$ in finite time starting from here.

Inductive Hypothesis: Suppose that if Pacman starts in position (i,n), he'll reach $(0,0)$ in finite time no matter what i is.

Inductive Step: We now consider what happens if Pacman starts from position $(i,n+1)$, where i can be any natural number. If Pacman starts by moving down, we can immediately apply the inductive hypothesis, since Pacman will be in position (i,n). However, if Pacman moves to the left, he'll be in position $(i-1,n+1)$, so we can't yet apply the inductive hypothesis. But note that Pacman can't keep moving left forever: after i such moves, he'll hit the wall on the y-axis and be forced to move down. Thus, Pacman must make a vertical move after only finitely many horizontal moves–and once he makes that vertical move, he'll be in position (k,n) for some $0 \leq k \leq i$, so the inductive hypothesis tells us that it will only take him a finite amount of time to reach $(0,0)$ from there. This means that Pacman can only take a finite amount of time moving to the left, one second making his first move down, then a finite amount of additional time after his first vertical move. Since a finite number plus one plus another finite number is still finite, this gives us our desired claim: Pacman must reach $(0,0)$ in finite time if he starts from position $(i,n+1)$ for any $i \in \mathbb{N}$.

8 Well-Ordered Grid

Consider an infinite sheet of graph paper such that each square contains a natural number. Suppose that the number in each square is equal to the average of the numbers in the four neighboring squares.

(a) By the Well-Ordering Principle, there must be some smallest number in the grid (call it n). Prove that for any square containing n, the four squares adjacent to it must also contain n.

(b) Prove that each square in the infinite grid contains the same number.

Solution:

(a) The basic idea is, the numbers around the minimum number n can't be larger, because that would raise the average above the n.
Formally: Suppose the four adjacent squares contain the numbers w, x, y, and z. For the sake of contradiction, assume that one of the four numbers (say w) is not equal to n. By our choice of n as the smallest number, we have $n \leq w, x, y, z$ and in fact we have $n < w$. Then

$$\frac{w+x+y+z}{4} > \frac{n+x+y+z}{4} \geq \frac{n+n+n+n}{4} = n.$$

But since each square is equal to the average of the squares around it, the LHS is equal to n, a contradiction.

(b) Introduce this problem by drawing out a finite grid, with the minimum number n in the center. Follow the chain of implications: First the 4 immediately adjacent cells must be equal to n. Then, the cells around those must also be equal to n, etc. Then move onto formalizing this by induction:

Let S be a square which contains the number n. Let us introduce the following definition: we say that a square T is of distance d from S if we can find a chain of $d+1$ adjacent squares which begins at S and ends at T. Now let T be any square in the grid. We must show that T contains the number n. The proof will be by induction on d, the distance from T to S.
Base Case: The base case occurs when $d = 0$, namely when $T = S$. Then there is nothing to prove, since S contains the number n by definition.
Inductive hypothesis: Assume that each square of distance d from S contains the number n.
Inductive Step: Suppose T is of distance $d+1$ from S. Then one of the four squares adjacent to T, say U, is of distance d from S. By the inductive hypothesis, U contains the number n. Now by part (a), all squares neighboring U must also contain n, so in particular T must contain n.

9 Losing Marbles

Two EECS70 GSIs are playing a game, where there is an urn that contains some number of red marbles (R), green marbles (G), and blue marbles (B). There is also an infinite supply of marbles outside the urn. When it is a player's turn, the player may either:

(i) Remove one red marble from the urn, and add 3 green marbles.

(ii) Remove two green marbles from the urn, and add 7 blue marbles.

(iii) Remove one blue marble from the urn.

These are the only legal moves. The last player that can make a legal move wins. We play optimally, of course – meaning we always play one of the best possible legal moves.

(a) Prove by induction that, if the urn initially contains a finite number of marbles at the start of the game, then the game will end after a finite number of moves.

(b) If the urn contains 2 green marbles and B blue marbles initially, then who will win the game? Prove it. In this case, does it matter what strategy the players use?

(c) If the urn contains (R,G,B) red, green, and blue marbles initially, then who will win the game? Prove it. In this case, does it matter what strategy the players use?

Solution:

(a) Imagine placing 1kg marbles on 3 pedestals: Red marbles on a 100 meter pedestal, Green marbles on a 10m one, and Blue marbles on a 1m one. Then every move corresponds to a downward flow of marbles (for example, move (i) converts 1 marble at 100m to 3 marbles at 10m). If we consider the potential energy $\Phi(R,G,B) = 100R+10G+B$, we find Φ decreases by at least 1 every turn.

Lemma: The value of Φ will decrease by at least 1 after every legal move.

Proof. By cases:

- Move (i): the value of Φ will change from $(100R+10G+B)$ to $(100(R-1)+10(G+3)+B) = (100R+10G+B)-70$, decreasing by 70.
- Move (ii): the value of Φ will change from $(100R+10G+B)$ to $(100R+10(G-2)+(B+7))$, decreasing by 13.
- Move (iii): the value of Φ will change from $(100R+10G+B)$ to $(100R+10G+(B-1))$, decreasing by 1.

□

Since Φ is never negative, we can't play this game forever. This argument is logically correct, but we must formalize it through induction.

Claim: Let Φ_0 be the value of the initial configuration and Φ_n be the value of the configuration after n moves where n is a natural number. Then $\Phi_n \leq \Phi_0 - n$.

Proof. Proof by simple induction over n, the number of legal moves made.

Base case: Let $n = 0$, then $\Phi_0 - n$ becomes Φ_0 and Φ_n becomes Φ_0. Trivially we see that $\Phi_0 \leq \Phi_0$.

Inductive hypothesis: Assume that for some $k \in \mathbb{N}$, $\Phi_k \leq \Phi_0 - k$.

Inductive step: Now consider the $k+1$-th turn. By the lemma, we know that the step will decrease the value of Φ_{k+1} by at least 1, so we conclude that $\Phi_{k+1} \leq \Phi_k - 1$. Hence $\Phi_{k+1} \leq \Phi_k - 1 \leq (\Phi_0 - k) - 1 = \Phi_0 - (k+1)$, where in the second step we used the inductive hypothesis. The claim follows by induction. □

Since $\Phi_n \leq \Phi_0 - n$, we know $n \leq \Phi_0 - \Phi_n$. Since $\Phi_n \geq 0$ by construction (the value can never be negative since it is the sum of natural numbers multiplied by other natural numbers), we can conclude $n \leq \Phi_0$. Since Φ_0 is clearly finite, this means that the game can have only a finite number of legal moves played and must therefore end.

In the solution above we assign weights 100, 10, and 1 for red, green, and blue marbles, respectively. However, there are other weight assignments that make this proof idea work. Another workable assignment would be 13, 4, and 1 for red, green, and blue. *i.e.*, $\Phi = 13R + 4G + B$ since that also ensures that each legal move reduces the value of Φ by at least 1. On the other hand, weighing all marbles equally would not work, because some moves would increase the value of Φ, violating the intended invariant.

What you should get out of this problem: The main idea is to associate each state of the game with a number in a way that lets us establish a useful invariant. This is a common strategy in solving these kinds of problems, but it need not be the only way. Notice also that when saying what you're inducting over, you should provide more detail than just saying "We induct over k" or "Proof by induction over n". Unless it is very clear what n means, specify what quantity n represents.

(b) First, consider a smaller case: If we started with 0 green marbles, then clearly P1 ("Player 1", the first player) wins if and only if B is odd. (Because each player must remove one blue marble every turn, the total number of turns is the number of initial blue marbles. And P1 wins if and only if the total number of turns is odd).

With 2 green marbles, the idea is essentially: At some point, someone will convert the 2 greens to 7 blues. Now we have only blues, so the next player wins if and only if the number of blue marbles **at this point** is odd. Following this through:

Claim: With 2 green marbles and B blue marbles initially, P1 wins if and only if B is odd (regardless of strategy).

Proof. Since we know from part (a) that the game ends, at some point someone must convert the 2 greens to 7 blues (otherwise, there would still be a legal move available). Let the number of moves **before** this conversion occurs be x. Let the number of moves **after** conversion, until the game ends, be y. Each move before or after conversion must have removed a blue marble, so in total the game lasted $x+y+1$ moves, and removed a total of $x+y$ blue marbles. Since the game started with B blue marbles, ends with 0 blue marbles, and the conversion introduced 7 new blue marbles, we must have $B+7=x+y$ (blue marbles initially + blue marbles added = total blue marbles removed). Since the game lasted $(x+y)+1=(B+7)+1=B+8$ moves, P1 wins if and only if this total number of moves is odd, or equivalently if and only if B is odd. □

(c) c

We can try generalizing the previous logic, first by considering arbitrary numbers of green and blue marbles, then also allowing red marbles.

However, we can simplify the argument by using a potential method. This will let us prove both the strategy-invariance and the winner in one sweep.

From the proof of part (b), we notice that the total number of moves in the game was useful. Let us try to find a function $M(R,G,B)$ for this. The following are plausible steps we may take in coming up with such a function. The formal proof follows.

If $R=G=0$, then trivially $M(0,0,B)=B$ since the only legal move is to remove a blue marble. Similarly, $M(0,1,B)=B$. From part (b), we know $M(0,2,B)=8+B$, essentially because the greens must be converted to blues (1 move), then the additional blues must be removed (7 more moves). We may postulate that for even G, $M(0,G,B)=4G+B$ for similar reasons (and in fact, could prove this by induction if we wish). Patching M slightly to deal with odd G, we may try: $M(0,G,B)=8\lfloor\frac{G}{2}\rfloor+B$

To deal with reds, we notice that at some point every red marble is converted to 3 green marbles. If all these R conversions happen at the beginning, we would have

$$M(R,G,B)=R+M(0,G+3R,B)=R+8\lfloor\frac{G+3R}{2}\rfloor+B.$$

Now, let us prove that this formula, which we arrived at by a series of intuitive steps and loose assumptions, actually works.

Claim: The total number of legal moves to complete a game, regardless of strategies, is always:

$$M(R,G,B)=R+8\lfloor\frac{G+3R}{2}\rfloor+B.$$

Where R,G,B is the initial number of red, green, and blue marbles, respectively.

Proof. First, prove by cases (as in part (a)) that M decreases by **exactly one** with every legal move. Further, M has the additional property:

Lemma: $M(R,G,B)$ is 0 if and only if there are no legal moves remaining.
Proof: If there are no legal moves, then there must be: 0 reds, 0 blues, and either 0 or 1 greens (any other case would allow a legal move). And $M(0,0,0) = M(0,1,0) = 0$. If $0 = M(R,G,B) = R + 8\lfloor\frac{G+3R}{2}\rfloor + B$, then each term in the sum on the right-hand side must be 0 (since they are all non-negative). Therefore: $R = B = 0$ and $8\lfloor\frac{G+3R}{2}\rfloor = 0 \implies \lfloor\frac{G}{2}\rfloor = 0 \implies G = 0$ or 1. □

Now we have a function $M(R,G,B)$ which decreases by exactly one with every legal move, and is zero if and only if the game is over.

Let s_0 be the initial state of the game (number of R,G,B marbles), and $s_1, s_2, \ldots, s_q$ be the states of the game after $1,2,\ldots,q$ moves, for some sequence of moves that ends the game. Since each move decreases M by exactly one, we must have $M(s_q) = M(s_0) - q$. But since the game ends at state s_q, we have $M(s_q) = 0$. Therefore, all sequences of moves that end the game in q moves have $q = M(s_0)$.

□

Therefore P1 wins if and only if $M(R,G,B)$ is odd.

10 Make Your Own Question

You must make your own question on this week's material and solve it.

11 Homework Process and Study Group

You must describe your homework process and study group in order to receive credit for this question.