Algorithmic Discrete Mathematics

March 21-23, 2017

Invariant Principle

The 15-puzzle

Here is a puzzle. There are 15 lettered tiles and a blank square arranged in a 4×4 grid. Any lettered tile adjacent to the blank square can be slid into the blank. For example, a sequence of two moves is illustrated below:

A	B	C	D
E	F	G	H
I	J	K	L
M	O	N	

A	B	C	D
E	F	G	H
I	J	K	L
M	O		N

A	B	C	D
E	F	G	H
I	J		L
M	O	K	N

In the leftmost configuration shown above, the N and O tiles are out of order. We can find a way of swapping N and O so that they are in the right order, but then other letters may be out of order. Can you find a sequence of moves that puts these two letters in the correct order, but returns every other tile to its original position? Some experimentation suggests that the answer is probably "no", so let's try to prove that.

We're going to take an approach that is frequently used in the analysis of software and systems. We'll look for an invariant, a property of the puzzle that is always maintained, no matter how you move the tiles around. If we can then show that putting the N and O tiles in the correct order would violate the invariant, then we can conclude that this is impossible.

Let's see how this game plan plays out. Here is the theorem we're trying to prove:

Theorem. No sequence of legal moves transforms the board below on the left into the board below on the right.

A	B	C	D
E	F	G	H
I	J	K	L
M	O	N	

A	B	C	D
E	F	G	H
I	J	K	L
M	N	O	

We'll build up a sequence of observations, stated as lemmas. Once we achieve a critical mass, we'll assemble these observations into a complete proof.

Define a row move as a move in which a tile slides horizontally and a column move as one in which a tile slides vertically. Assume that tiles are read top-to-bottom and left-to-right like English text, that is, the natural order, defined as follows:

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

So when we say two tiles are "out of order", we mean that the larger letter precedes the smaller letter in this natural order.

Our difficulty is that one pair of tiles (the N and O) is out of order initially. An immediate observation is that row moves alone are of little value in addressing this problem:

Lemma 1. A row move does not change the order of the tiles.

Proof. A row move moves a tile from cell i to cell $i + 1$ or vice versa. This tile does not change its order with respect to any other tile. Since no other tile moves, there is no change in the order of any of the other pairs of tiles.

Let's turn to column moves. This is the more interesting case, since here the order can change. For example, the column move shown below changes the relative order of the pairs (G, H), (G, I), and (G, J).

A	B	C	D
E	F	G	H
I	J		L
M	O	K	N

A	B	C	D
E	F		H
I	J	G	L
M	O	K	N

Lemma 2. A column move changes the relative order of exactly 3 pairs of tiles.

Proof. Sliding a tile down moves it after the next three tiles in the order. Sliding a tile up moves it before the previous three tiles in the order. Either way, the relative order changes between the moved tile and each of the three it crosses. The relative order between any other pair of tiles does not change.

These observations suggest that there are limitations on how tiles can be swapped. Some such limitation may lead to the invariant we need. In order to reason about swaps more precisely, let's define a term referring to a pair of items that are out of order:

Definition. A pair of letters L_1 and L_2 is an inversion if L_1 precedes L_2 in the alphabet, but L_1 appears after L_2 in the puzzle order.

For example, in the puzzle below, there are seven inversions: (G, H), (G, I), (G, J), (K, L), (K, M), (K, O), and (N, O).

A	B	C	D
E	F		H
I	J	G	L
M	O	K	N

There is exactly one inversion (N, O) in the start state and there are no inversions in the end state. Let's work out the effects of row and column moves in terms of inversions.

Lemma 3. A row move never changes the parity of the number of inversions.

Proof. This follows from Lemma 1.

Lemma 4. A column move always changes the parity of the number of inversions.

Proof. A column move changes the relative order of exactly 3 pairs of tiles. Thus, an inverted pair becomes uninverted and vice versa. Thus, one exchange flips the total number of inversions to the opposite parity, a second exchange flips it back to the original parity, and a third exchange flips it to the opposite parity again.

The previous observations imply that we must make an odd number of column moves in order to exchange just one pair of tiles. But this is problematic, because each column move also moves the blank square up or down one row. So after an odd number of column moves, the blank can not possibly be back in the last row, where it belongs! Now we can bundle up all these observations and state an invariant, a property of the puzzle that never changes, which holds no matter how you slide the tiles around.

Lemma 5. In every configuration reachable from the starting configuration, the parity of the number of inversions is different from the parity of the row containing the blank square.

Proof. We use induction. Let $P(n)$ be the proposition that after n moves, the parity of the number of inversions is different from the parity of the row containing the blank square.

Base case. After zero moves, exactly one pair of tiles is inverted (N and O), which is an odd number. And the blank square is in row 4, which is an even number. Therefore, $P(0)$ is true.

Inductive step. Now we must prove that $P(n)$ implies $P(n+1)$ for all $n \geqslant 0$. So assume that $P(n)$ is true; that is, after n moves the parity of the number of inversions is different from the parity of the row containing the blank square. There are two cases:

(1) Suppose move $n+1$ is a row move. Then the parity of the total number of inversions does not change. The parity of the row containing the blank square does not change either, since the blank remains in the same row. Therefore, these two parities are different after $n+1$ moves as well, so $P(n+1)$ is true.

(2) Suppose move $n+1$ is a column move. Then the parity of the total number of inversions changes. However, the parity of the row containing the blank square also changes, since the blank moves up or down one row. Thus, the parities remain different after $n+1$ moves, and so $P(n+1)$ is again true.

Thus, $P(n)$ implies $P(n+1)$ for all $n \geqslant 0$.

By the principle of induction, $P(n)$ is true for all $n \geqslant 0$.

With the invariant in hand, the proof of the theorem is simple. In the target configuration on the right, the total number of inversions is zero, which is even, and the blank square is in row 4, which is also even. Therefore, by Lemma 5, the target configuration is unreachable.

This kind of puzzle was originally invented by Sam Lloyd in 1874, and called the 15-puzzle. It was very popular in its day. Somewhat like the Rubik's cube, but here there is no solution!

If you ever played with Rubik's cube, you know that there is no way to rotate a single corner, swap two corners, or flip a single edge. All these facts are provable with invariant arguments like the one above. In the wider world, invariant arguments are used in the analysis of complex protocols and systems, often to show that you don't get into a really bad state.

Die Hard

In the movie Die Hard 3: With a Vengeance, the characters played by Samuel L. Jackson and Bruce Willis have to disarm a bomb planted by the diabolical Simon Gruber. They are given a five-gallon jug and a three-gallon jug beside a fountain in a city park, and have to measure out exactly four gallons of water on a scale to keep the bomb from exploding. Fortunately, they find a solution in the nick of time. (No doubt reading the script helped.)

Unfortunately, Hollywood never lets go of a gimmick. Although there were no water jug tests in Die Hard 4: Live Free or Die Hard, rumor has it that the jugs will return in future sequels:

Die Hard 5: Die Hardest. Bruce goes on vacation and – shockingly – happens into a terrorist plot. To save the day, he must make 3 gallons using 21- and 26-gallon jugs.

Die Hard 6: Die of Old Age. Bruce must save his assisted living facility from a criminal mastermind by forming 2 gallons with 899- and 1147-gallon jugs.

Die Hard 7: Die Once and For All. Bruce has to make 4 gallons using 3- and 6-gallon jugs.

It would be nice if we could solve all these silly water jug questions at once. In particular, how can one form g gallons using jugs with capacities a and b?

Suppose that we have water jugs with capacities a and b. Let's carry out a few arbitrary operations and see what happens. The state of the system at each step is described below with a pair of numbers (x, y), where x is the amount of water in the jug with capacity a and y is the amount in the jug

with capacity b.

$$\begin{array}{rcll}
(0,0) & \to & (a,0) & \text{fill first jug} \\
 & \to & (0,a) & \text{pour first into second} \\
 & \to & (a,a) & \text{fill first jug} \\
 & \to & (2a-b,b) & \text{pour first into second} \\
 & \to & (2a-b,0) & \text{empty second jug} \\
 & \to & (0,2a-b) & \text{pour first into second} \\
 & \to & (a,2a-b) & \text{fill first} \\
 & \to & (3a-2b,b) & \text{pour first into second}
\end{array}$$

Of course, we're making some assumptions about the relative capacities of the two jugs here. But another point leaps out: at every step, the amount of water in each jug is of the form

$$s \cdot a + t \cdot b$$

for some integers s and t. An expression of this form is called a linear combination of a and b. This sounds like an invariant assertion that we might be able to prove by induction!

Lemma. Suppose that we have water jugs with capacities a and b. Then the amount of water in each jug is always a linear combination of a and b.

Proof. We use induction. Our invariant $P(n)$ is the proposition that after n steps, the amount of water in each jug is a linear combination of a and b.

Base case. $P(0)$ is true, because both jugs are initially empty, and

$$0 \cdot a + 0 \cdot b = 0.$$

Inductive step. Now we must show that $P(n)$ implies $P(n+1)$ for all $n \geqslant 0$. So assume that after n steps the amount of water in each jug is a linear combination of a and b. There are two cases:

(1) If we fill a jug from the fountain or empty a jug into the fountain, then that jug is empty or full. The amount in the other jug remains a linear combination of a and b. So $P(n+1)$ holds.

(2) Otherwise, we pour water from one jug to another until one is empty or the other is full. By our assumption, the amount in each jug is a linear combination of a and b before we begin pouring:

$$j_1 = s_1 \cdot a + t_1 \cdot b$$
$$j_2 = s_2 \cdot a + t_2 \cdot b$$

After pouring, one jug is either empty (contains 0 gallons) or full (contains a or b gallons). Thus, the other jug contains either $j_1 + j_2$ gallons, $j_1 + j_2 - a$, or $j_1 + j_2 - b$ gallons, all of which are linear combinations of a and b.

Thus, $P(n)$ implies $P(n+1)$ for all $n \geqslant 0$.

By the principle of induction, $P(n)$ is true for all $n \geqslant 0$.

This lemma isn't very satisfying. We've just managed to recast a pretty understandable question about water jugs into a complicated question about linear combinations. This might not seem like progress. Fortunately, linear combinations are closely related to something more familiar and that will help us solve the water jug problem.

The greatest common divisor of a and b is exactly what you'd guess: the largest number that is a divisor of both a and b. It is denoted $\gcd(a, b)$. For example, $\gcd(18, 24) = 6$.

Probably some junior high math teacher made you compute greatest common divisors for no apparent reason until you were blue in the face. But, amazingly, the greatest common divisor actually turns out to be quite useful for reasoning about the integers. Specifically, the quantity $\gcd(a, b)$ is a valuable piece of information about the relationship between the numbers a and b. The theorem below relates the greatest common divisor to linear combinations.

Theorem. The greatest common divisor of a and b is equal to the smallest positive linear combination of a and b.

For example, the greatest common divisor of 52 and 44 is 4. And, sure enough, 4 is a linear combination of 52 and 44:

$$6 \cdot 52 + (-7) \cdot 44 = 4.$$

Furthermore, no linear combination of 52 and 44 is equal to a smaller positive integer.

Note that every linear combination of a and b is a multiple of $\gcd(a, b)$. Conversely, since $\gcd(a, b)$ is a linear combination of a and b, every multiple of $\gcd(a, b)$ is as well. This establishes the following

Corollary 1. Every linear combination of a and b is a multiple of $\gcd(a, b)$ and vice versa.

Now we can restate the water jugs lemma in terms of the greatest common divisor:

Corollary 2. Suppose that we have water jugs with capacities a and b. Then the amount of water in each jug is always a multiple of $\gcd(a, b)$.

For example, there is no way to form 4 gallons using 3- and 6-gallon jugs, because 4 is not a multiple of $\gcd(3, 6) = 3$.

Let's see if one can form 3 gallons using 21- and 26-gallon jugs. Now 3 is a multiple of $\gcd(21, 26) = 1$, so we can't rule out the possibility that one can form 3 gallons. On the other hand, we don't know she can do it either.

Corollary 1 says that 3 can be written as a linear combination of 21 and 26, since 3 is a multiple of $\gcd(21, 26) = 1$. In other words, there exist integers s and t such that:

$$3 = s \cdot 21 + t \cdot 26.$$

Now the coefficient s could be either positive or negative. However, we can readily transform this linear combination into an equivalent linear combination

$$3 = s' \cdot 21 + t' \cdot 26$$

where the coefficient s' is positive. The trick is to notice that if we increase s by 26 in the original equation and decrease t by 21, then the value of the expression $s \cdot 21 + t \cdot 26$ is unchanged overall. Thus, by repeatedly increasing the value of s (by 26 at a time) and decreasing the value of t (by 21 at a time), we get a linear combination $s' \cdot 21 + t' \cdot 26 = 3$ where the coefficient s' is positive. Notice that t' must be negative; otherwise, this expression would be much greater than 3.

Now here's how to form 3 gallons using jugs with capacities 21 and 26:

- Repeat s' times:
  - Fill the 21-gallon jug.
  - Pour all the water in the 21-gallon jug into the 26-gallon jug. Whenever the 26-gallon jug becomes full, empty it out, and continue pouring from the smaller jug into the larger jug until the smaller jug is empty.

At the end of this process, there must be exactly 3 gallons in the 26-gallon jug! Here's why: we've taken $s' \cdot 21$ gallons of water from the fountain, we've

poured out some multiple of 26 gallons, and in the end the 26-gallon jug holds somewhere between 0 and 26 gallons. Furthermore, we know that

$$s' \cdot 21 + t' \cdot 26 = 3$$

Thus, we must have emptied the 26-gallon jug exactly $-t'$ times; if we had emptied it fewer times, then there would be more than 26 gallons left. And we did not withdraw enough water from the fountain to empty the 26-gallon jug more than $-t'$ times. Thus, by the equation above, there must be exactly 3 gallons left.

Remarkably, we don't even need to know the coefficients s' and t' in order to use this strategy! Instead of repeating the outer loop s' times, we could just repeat until we obtain 3 gallons, since that must happen eventually. Of course, we have to keep track of the amounts in the two jugs so we know when we're done.

The same approach works regardless of the jug capacities and even regardless the amount we're trying to produce! Simply proceed as follows:

- Repeat until the desired amount of water is obtained:
  - Fill the smaller jug.
  - Pour all the water in the smaller jug into the larger jug. Whenever the larger jug becomes full, empty it out.

By the same reasoning as before, this method eventually generates every multiple of the greatest common divisor of the jug capacities – all the quantities we can possibly produce. No ingenuity is needed at all!

Combinatorial Games

Unstacking

Here is another game that's surely about to sweep the nation! You begin with a stack of n boxes. Then you make a sequence of moves. In each move, you divide one stack of boxes into two nonempty stacks. The game ends when you have n stacks, each containing a single box. You earn points for each move; in particular, if you divide one stack of height $a + b$ into two stacks with heights a and b, then you score ab points for that move. Your overall score is the sum of the points that you earn for each move. What strategy should you use to maximize your total score?

As an example, suppose that we begin with a stack of $n = 10$ boxes. Then the game might proceed as follows:

10	
5 5	25
5 3 2	6
4 3 2 1	4
2 3 2 1 2	4
2 2 2 1 2 1	2
1 2 2 1 2 1 1	1
1 1 2 1 2 1 1 1	1
1 1 1 1 2 1 1 1 1	1
1 1 1 1 1 1 1 1 1 1	1
	45

Can you find a better strategy?

We prove that your score is determined entirely by the number of boxes; your strategy is irrelevant!

Theorem. Every way of unstacking n blocks gives a score of $n(n-1)/2$ points.

Proof. The proof is by strong induction. Let $P(n)$ be the proposition that every way of unstacking n blocks gives a score of $n(n-1)/2$.

Base case. If $n = 1$, then there is only one block. No moves are possible, and so the total score for the game is $1(1-1)/2 = 0$. Therefore, $P(1)$ is true.

Inductive step. Now we must show that $P(1), \ldots, P(n)$ imply $P(n+1)$ for all $n \geqslant 1$. So assume that $P(1), \ldots, P(n)$ are all true and that we have a stack of $n+1$ blocks. The first move must split this stack into substacks with sizes k and $n+1-k$ for some k strictly between 0 and $n+1$. Now the total score for the game is the sum of points for this first move plus points obtained by unstacking the two resulting substacks:

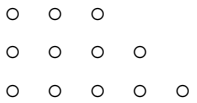
$$\begin{aligned}
\text{total score} &= (\text{score for the first move}) \\
&\quad + (\text{score for unstacking } k \text{ blocks}) \\
&\quad + (\text{score for unstacking } n+1-k \text{ blocks}) \\
&= k(n+1-k) + \frac{k(k-1)}{2} + \frac{(n+1-k)(n+1-k-1)}{2} \\
&= \frac{2kn + 2k - 2k^2 + k^2 - k + n^2 - nk + n - k - kn + k^2}{2} \\
&= \frac{n^2 + n}{2} \\
&= \frac{(n+1)n}{2}.
\end{aligned}$$

The second equation uses the assumptions $P(k)$ and $P(n+1-k)$ and the rest is simplification. This shows that $P(1), \ldots, P(n)$ imply $P(n+1)$.

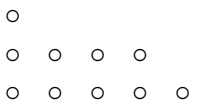
Therefore, the claim is true by strong induction.

Nim

Nim is a game involving two players and some pennies (and mathematical induction). The game begins with a bunch of pennies, arranged in one or more rows. For example, here we have three rows of pennies:



The two players take turns. On each turn, a player must remove one or more pennies, all from a single row. The player who takes the last penny wins. For example, suppose that the first player removes two pennies from the first row:



Now the second player removes all the pennies from the last row, leaving this configuration:

The first player then takes three pennies from the last row:

The second player is now in trouble; she must take exactly one of these two pennies. Either way, the first player takes the last penny and wins the game.

There is a compact way to describe a configuration in Nim: list the number of pennies in each row. For example, the starting configuration in the game above is $(3, 4, 5)$. The game then passed through the configurations $(1, 4, 5)$, $(1, 4, 0)$, and $(1, 1, 0)$.

A Harvard professor, Charles Bouton, discovered the optimal strategy for Nim in 1901. Bouton's strategy relies on a special mathematical operation called a Nimsum. The Nimsum of natural numbers $c_1, \ldots, c_k$ is itself a natural number that is computed as follows:

- Regard $c_1, \ldots, c_k$ as binary numbers.
- The i-th bit of the Nimsum is the xor of the i-th bits of $c_1, \ldots, c_k$.

(Xor is pronounced "eks-or" and is short for "exclusive-or". The xor of bits b_1 and b_2 is denoted $b_1 \oplus b_2$ and defined as follows:

b_1	b_2	$b_1 \oplus b_2$
0	0	0
0	1	1
1	0	1
1	1	0

As a consequence of this definition, the xor of bits $b_1, \ldots, b_k$ is 0 if the sum of the bits is even and 1 if the sum is odd. For example, $1 \oplus 0 \oplus 1 \oplus 1 = 1$ because $1+0+1+1 = 3$ is odd, but $1 \oplus 1 \oplus 0 \oplus 0 = 0$ because $1+1+0+0 = 2$ is even.)

As an example, the Nimsum of 3, 4, and 5 is computed as follows:

$$\begin{array}{rcl} 3 & = & 011 \\ 4 & = & 100 \\ 5 & = & 101 \\ \hline & & 010 \quad = \quad 2 \end{array}$$

Here, we xor each column of bits to obtain the Nimsum 010, which is the binary representation of 2.

Suppose that $(c_1, \ldots, c_k)$ is a configuration in Nim; that is, there are a total of k rows, and the i-th row contains c_i pennies. This configuration is called:

- safe if the Nimsum of $c_1, \ldots, c_k$ is nonzero,
- unsafe if the Nimsum of $c_1, \ldots, c_k$ is zero.

In this way, all possible configurations in Nim are now partitioned into two groups: the safe configurations and the unsafe configurations. Bouton's strategy is as follows:

- If you see a safe configuration on your turn, then select a move that leaves your opponent with an unsafe configuration.
- If you see an unsafe configuration on your turn, you're doomed. Every possible move leaves your opponent with a safe configuration; select among them arbitrarily and prepare to lose.

Thus, if the first player sees a safe configuration, she selects a move that leaves the second player an unsafe configuration. Every move available to the second player then leaves the first player with another safe configuration. This cycle repeats until the first player wins the entire game.

Let's use Bouton's strategy in one example game before trying to prove a general theorem. The bizarre French movie "Last Year at Marienbad" features a Nim game beginning in the configuration $(1, 3, 5, 7)$. Let's analyze this configuration by computing the Nimsum:

$$\begin{array}{rcl} 1 & = & 001 \\ 3 & = & 011 \\ 5 & = & 101 \\ 7 & = & 111 \\ \hline & & 000 \quad = \quad 0 \end{array}$$

Since the Nimsum is zero, this is a hopeless situation for the first player. In particular, no matter what she does, the second player is left with a safe configuration; that is, one in which the Nimsum is nonzero. For example, suppose that the first player removes the entire fourth row. Then the Nimsum becomes:

$$\begin{array}{rcl} 1 & = & 001 \\ 3 & = & 011 \\ 5 & = & 101 \\ 0 & = & 000 \\ \hline & & 111 \quad = \quad 7 \end{array}$$

This is a safe configuration. To maintain her advantage, the second player needs to leave the first player in an unsafe configuration; that is, a configuration with Nimsum zero. Only one move accomplishes this: removing three pennies from the third row:

$$\begin{array}{rcl} 1 & = & 001 \\ 3 & = & 011 \\ 2 & = & 010 \\ 0 & = & 000 \\ \hline & & 000 \quad = \quad 0 \end{array}$$

Now the first player is again confronted with an unsafe configuration, one with a Nimsum of zero. No matter what she does, the second player will be left with a safe configuration. For example, suppose that the first player takes all three pennies from the second row. Then the second player is left with:

$$\begin{array}{rcl} 1 & = & 001 \\ 0 & = & 000 \\ 2 & = & 010 \\ 0 & = & 000 \\ \hline & & 011 \quad = \quad 3 \end{array}$$

Sure enough, this is a safe configuration. Once again, the second player needs to leave the first player in an unsafe configuration. Removing one penny from the third row accomplishes this:

$$\begin{array}{rcl} 1 & = & 001 \\ 0 & = & 000 \\ 1 & = & 001 \\ 0 & = & 000 \\ \hline & & 000 \quad = \quad 0 \end{array}$$

The first player is clearly going to lose. She must take one of the two remaining pennies, leaving the second player to take the other and win the game.

Now let's prove that Bouton's strategy works. We'll need two lemmas.

Lemma 1. If a player sees an unsafe configuration on her turn, then every possible move leaves her opponent with a safe configuration.

Proof. Suppose that the player sees an unsafe configuration; that is, a configuration $(c_1, \ldots, c_k)$ with Nimsum zero. Then the player must remove pennies from some row j, leaving $c'_j < c_j$ pennies behind. In binary, the numbers c_j and c'_j must differ in some bit position. But then the Nimsum of the resulting configuration has a 1 in that bit position. Therefore, the Nimsum is nonzero, meaning that the resulting configuration is safe.

Lemma 2. If a player sees a safe configuration on her turn, then she has a move that leaves her opponent with an unsafe configuration.

Proof. Suppose that the player sees a configuration $(c_1, \ldots, c_k)$ with Nimsum $s \neq 0$. Let i be the position of the most-significant 1 in the binary representation of s. Since the i-th position of s is nonzero, there must be some row size c_j that is nonzero in the i-th position as well. Suppose that the player removes all but c'_j pennies from the j-th row, where c'_j differs from c_j in exactly those positions where s has a 1.

We must check that c'_j, the number of pennies the player leaves behind, is less than c_j, the number of pennies originally in the j-th row. This is true because c_j has a 1 in the i-th position, c'_j has a 0 in this position, and the two numbers agree in all higher positions.

Modifying the game configuration by changing c_j by c'_j reduces the Nimsum to zero, because c_j and c'_j disagree in exactly those positions where the original Nimsum had a 1.

An example may clarify the preceding argument. Suppose that a player sees the configuration $(1, 2, 4, 4)$. We can compute the Nimsum as follows:

$$\begin{array}{rcl} 1 & = & 001 \\ 2 & = & 010 \\ 4 & = & 100 \\ 4 & = & 100 \\ \hline & & 011 \quad = \quad 3 \end{array}$$

The Nimsum is 011 in binary. The most significant 1 is in the second position from the right. The number of pennies in the second row, 010 in binary, also has a 1 in this position. We should leave 001 pennies in this row, since this differs from the number of pennies there now (010) in every position where the Nimsum (011) has a 1. Now we compute the Nimsum of the resulting configuration:

$$\begin{array}{rcl} 1 & = & 001 \\ 1 & = & 001 \\ 4 & = & 100 \\ 4 & = & 100 \\ \hline & & 000 \quad = \quad 0 \end{array}$$

As expected, this is an unsafe configuration.

We're ready to prove the main result.

Theorem. If the current player has a safe configuration, then she can guarantee a win. Otherwise, the other player can guarantee a win.

The proof uses induction, with the statement of the theorem as the induction hypothesis. This hypothesis is quite complicated! Suppose that we define three smaller propositions:

$$\begin{aligned} A &= \text{current player has a safe configuration,} \\ B &= \text{current player can guarantee a win,} \\ C &= \text{next player can guarantee a win.} \end{aligned}$$

In these terms, the induction hypothesis is $(A \to B) \wedge (\bar{A} \to C)$. Induction hypotheses containing implications are always a bit tricky, but this one has two implications! Some square-bracketed comments are included in the proof to clairfy the structure.

Proof. The proof is by strong induction on n, the total number of pennies remaining. Let $P(n)$ be the proposition that in all Nim configurations with n pennies, if the current player has a safe configuration, then she can guarantee a win and, otherwise, the other player can guarantee a win.

Base case. Suppose that $n = 0$; that is, no pennies remain. This is an unsafe configuration. Therefore, the first implication is trivially true because the if-part is false. The second implication is also true, because when the current player is left with no pennies, the other player has just won the game. [In other words, the proposition $(A \to B) \wedge (\bar{A} \to C)$ is true because A is false and C is true.]

Inductive step. We must show that for all $n \geqslant 0$, the propositions $P(0)$, $\ldots, P(n)$ imply $P(n+1)$. Assume $P(0), \ldots, P(n)$ and consider a configuration with $n+1$ pennies remaining. Now we must show two things:

(1) If the configuration is safe, the current player can guarantee a win. Suppose that the configuration is safe. By Lemma 2, the current player has a move that leaves the next player with an unsafe configuration involving fewer than $n+1$ pennies. If $m < n+1$ pennies remain, then assumption $P(m)$ implies that the player after the next player (which is the current player) can guarantee a win. [This shows $A \to B$.]

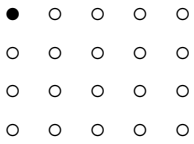
(2) If the configuration is unsafe, the next player can guarantee a win. Suppose that the configuration is unsafe. By Lemma 1, every move leaves the next player with a safe configuration involving $m < n+1$ pennies. The assumption $P(m)$ then implies that the next player can guarantee a win. [This shows $\bar{A} \to C$.]

Putting these arguments together gives $P(n+1)$. [That is, $A \to B$ and $\bar{A} \to C$ together imply $(A \to B) \wedge (\bar{A} \to C)$.]

Therefore, the theorem follows by the principle of induction.

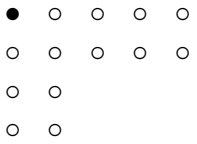
Chomp

Chomp is a game played by two players. In this game, cookies are laid out on a rectangular grid. The cookie in the top left position is poisoned:



The two players take turns making moves; at each move, a player is required to eat a remaining cookie, together with all cookies to the right and/or below

it:



The loser is the player who has no choice but to eat the poisoned cookie. We ask whether one of the two players has a winning strategy. That is, can one of the players always make moves that are guaranteed to lead to a win?

We will give a nonconstructive existence proof of a winning strategy for the first player. That is, we will show that the first player always has a winning strategy without explicitly describing the moves this player must follow.

First, note that the game ends and cannot finish in a draw because with each move at least one cookie is eaten, so after no more than mn moves the game ends, where the initial grid is $m \times n$.

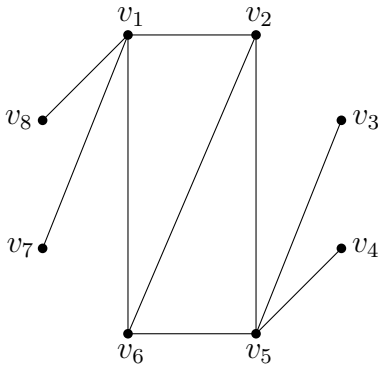
Now, suppose that the first player begins the game by eating just the cookie in the bottom right corner. There are two possibilities, this is the first move of a winning strategy for the first player, or the second player can make a move that is the first move of a winning strategy for the second player. In this second case, instead of eating just the cookie in the bottom right corner, the first player could have made the same move that the second player made as the first move of a winning strategy (and then continued to follow that winning strategy). This would guarantee a win for the first player.

Note that we showed that a winning strategy exists, but we did not specify an actual winning strategy. Consequently, the proof is a nonconstructive existence proof. In fact, no one has been able to describe a winning strategy for that Chomp that applies for all rectangular grids by describing the moves that the first player should follow. However, winning strategies can be described for certain special cases, such as when the grid is square and when the grid only has two rows of cookies.

Stable Marriage

Today, we are going to talk about matching problems. Matching problems arise in numerous applications. For example, dating services want to pair up compatible couples. Interns need to be matched to hospital residency programs. Other assignment problems involving resource allocation arise frequently, including balancing the traffic load among servers on the Internet. In the simplest form of a matching problem, you are given a graph where the edges represent compatibility and the goal is to create the maximum number of compatible pairs.

Definition. Given a graph $G = (V, E)$, a matching is a subgraph of G where every node has degree 1. In particular, the matching consists of edges that do not share nodes.

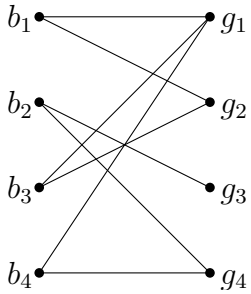


In this graph, $\{\{v_1, v_6\}, \{v_2, v_5\}\}$ is a matching of size two. But there is a larger matching – namely, $\{\{v_1, v_8\}, \{v_2, v_6\}, \{v_4, v_5\}\}$ is a matching of size three. Can there be a larger matching? Well, that would mean that every node is paired. But each of v_7 and v_8 can only be paired with v_1, and v_1 can only be paired with one other node in a matching. So, the answer is no!

Let's now define a matching that includes every node:

Definition. A matching of a graph $G = (V, E)$ is perfect if it has $|V|/2$ edges.

There is no perfect matching for the previous graph. Matching problems often arise in the context of the bipartite graphs – for example, the scenario where you want to pair boys with girls.



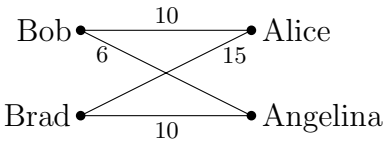
For example, the above graph has a perfect matching, namely $\{\{b_1, g_2\}, \{b_2, g_3\}, \{b_3, g_1\}, \{b_4, g_4\}\}$.

In many applications, not all matchings are equally desirable. For example, maybe b_1 and g_2 like each other a lot more than b_1 and g_1. Often, we can represent the desirability of a matching with a weight on the edge. For example b_1 and g_2 get weight 5 while b_1 and g_1 get weight 10.

The goal then is to find the perfect matching with the minimum weight.

Definition. The weight of matching M is the sum of the weights on the edges in M. The min-weight matching for a graph G is the perfect matching for G with minimum weight. (If it exists.)

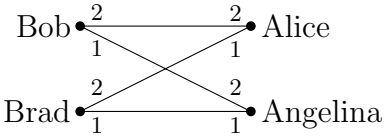
For example, the min-weight matching for the following graph is 20 (Bob gets matched with Alice, and Brad with Angeline).



It turns out that there are fast algorithms for finding maximum matchings in unweighted graphs and min-weight matchings in weighted graphs, but they are complicated and we don't cover them in this course (in particular, the greedy algorithm doesn't work in general).

Instead, we are going to talk about a different variant of the matching problem that does have an elegant solution and that is frequently used in practice. In this version of the problem, every node has a preference order of the possible mates. The preferences don't have to be symmetric. For

example, maybe Alice really likes Brad but Brad has the hots for Angelina. Suppose Angelina also likes Brad more than Bob but that Bob really likes Angelina.



In the above figure, suppose we were to pair Brad with Alice and Bob with Angelina. Well, that would lead to a very dicey situation! Suffice it to say that pretty soon, Brad and Angelina are likely to start spending late nights doing discrete mathematics homework together. The main problem is that Brad and Angelina each prefer each other to their mates in the matching. In such a circumstance we say that Brad and Angelina form a rogue couple. More precisely, we'll say that given a matching M, boy b and girl g are a rogue couple for M if b and g prefer each other to their mates in M.

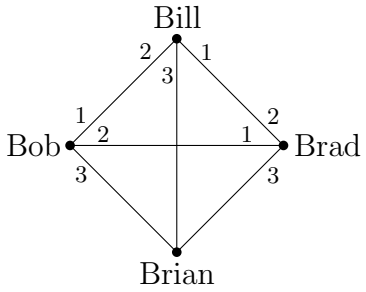
Obviously, the existence of rogue couples is not a good thing if you are making matchings, since they lead to instability. So, we'll say that a matching is stable if there are no rogue couples.

We are going to assume that preferences do not change with time. So we are not modeling the situation where you get tired of your mate or want to "play the field". Preferences are known at the start and never change.

Our main goal is to find a perfect matching that is stable. In this example, a possible stable perfect matching is to pair Brad with Angelina, and Bob with Alice. Bob and Alice may not be so happy, but no rogue couple is possible and so it is a stable matching. That's because neither Brad nor Angelina like anyone better than each other, so even though Alice and Bob are not happy with each other, no one else will form a rogue couple with either of them.

In general, it isn't so clear that there is always a stable matching for any number of people and set of preference orders. In fact, the answer is kind of complicated. If you allow boys to prefer boys and girls to prefer girls, then there are examples where there is no stable matching. But the strange thing is that in the special case where boys only get pairs with girls, then you can always find a stable matching.

We're going to show how to find such a stable matching shortly. But first, let's look at a unisex example where a stable matching is not possible. The idea is to create a love triangle with a fourth person who is everyone's last choice:



It turns out the fourth person's preferences don't even matter. Let's see why there is no stable matching. We'll prove this by contradiction. Assume, for contradiction, that there is a stable matching. Then there are two members of the love triangle that are matched. Without loss of generality, (by symmetry) assume that Bill is matched to Bob. Then the other pair must be Brad matched with Brian. But then there is a rogue couple since Bill likes Brad best and Brad prefers Bill to Brian. So, Bill and Brian are a rogue couple. Thus, there cannot be a stable matching.

This proposition is not very surprising. Getting a stable matching is a hard thing to do. What is surprising is that you can always do it in bipartite graphs – that is, where boys are only allowed to pair with girls and vice versa.

Let's formalize the statement of the problem that we are discussing here.

The setting:

- There are n boys and n girls. We assume the number of boys and girls is the same.
- Each boy has his own ranked preference list of girls.
- Each girl has her own ranked preference list of boys.
- The lists are complete and have no ties. Each boy ranks every girl and vice versa.

The goal:

Pair each boy with a unique girl so that there are no rogue couples. That is, find a perfect matching so that every boy and girl are paired up one to one, with no potential funny business.

Let's see if we can figure out a method for finding a stable matching by looking at an example:

$$b_1 \to (g_3, g_2, g_5, g_1, g_4)$$
$$b_2 \to (g_1, g_2, g_5, g_3, g_4)$$
$$b_3 \to (g_4, g_3, g_2, g_1, g_5)$$
$$b_4 \to (g_1, g_3, g_4, g_2, g_5)$$
$$b_5 \to (g_1, g_2, g_4, g_5, g_3)$$

$$g_1 \to (b_3, b_5, b_2, b_1, b_4)$$
$$g_2 \to (b_5, b_2, b_1, b_4, b_3)$$
$$g_3 \to (b_4, b_3, b_5, b_1, b_2)$$
$$g_4 \to (b_1, b_2, b_3, b_4, b_5)$$
$$g_5 \to (b_2, b_3, b_4, b_1, b_5)$$

Let's try to use a greedy algorithm to find the matching. In this case, the greedy algorithm will have each boy pick his favorite girl that remains by the time his turn comes up.

Running the greedy algorithm on our example, boy b_1 picks his favorite, which is g_3, boy b_2 picks his favorite, which is g_1, boy b_3 picks his favorite, which is g_4, boy b_4 picks his favorite remaining girl, which is g_2 (since his top 3 choices are already taken), and finally, boy b_5 picks his favorite remaining girl (which at this point, is the only remaining girl), which is g_5.

Let's see – is there a rogue couple? Well, boys b_1, b_2, and b_3 are matched up with the loves of their lives, so they are too happy to be thinking of running off. However, boy b_4 is not so happy with g_2, who is his fourth choice. He approaches g_1, but she isn't interested in him, since she prefers boy b_2 – in fact, she ranked b_4 dead last so she wouldn't be caught dead in an affair with him. However, he runs into his love of his life, that is g_3, and she definitely prefers him to b_1, who is way down on her list. So we have a situation here. Both b_4 and g_3 prefer each other to their own mates. We could try to patch things up and pair b_4 with g_3 and then g_2 with b_1, but it's not clear that we would reduce the number of rogue couples, since for all we know, b_4 and g_3 could still be in rogue couple. It happens that in this case pairing up b_4 and g_3 is an ok thing to do. But, this is getting more and more complicated.

How about using an algorithm that is based on induction (or recursion)? Pair boy b_1 with girl g_3 and solve the rest by induction. By the induction hypothesis, the only rogue couples would involve b_1 or g_3. But, they can't

involve b_1, since he got his first choice. On the other hand, they might well involve g_3 since b_1 might be her last but one choice! Induction would work if there were some boy and some girl who each ranked the other first. If there were such a boy and girl, then they have to get paired to each other, or they would be a rogue couple. But, there might not be such a boy and girl. Too often people do not like those that like them!

Turns out that finding a good way of pairing up the boys and girls is a tricky problem. The best approach is to use the Gale-Shapley algorithm named after the mathematicians David Gale and Lloyd Shapley who devised it in 1962. Note that, in 2012, the Nobel Prize in Economics was awarded to Lloyd Shapley and Alvin Roth "for the theory of stable allocations and the practice of market design."

Here is the method for getting everyone paired up. The mating ritual takes place over several days. The idea is that each of the boys go after the girls one by one, in order of preference, crossing off girls from their list as they get rejected. Here is a more detailed specification:

Initial Condition:

Each of the n boys has an ordered list of the n girls according to his preferences. Each of the girls has an ordered list of the boys according to her preferences.

Each Day

- Morning:
  - Each girl stands on her balcony.
  - Each boy stands under the balcony of his favorite girl whom he has not yet crossed off his list and serenades. If there are no girls left on his list, he stays home and does discrete mathematics homework.
- Afternoon:
  - Girls who have at least one suitor say to their favorite from among the suitors that day: "Maybe, come back tomorrow."
  - To the others, they say "No, I will never marry you!"
- Evening:
  - Any boy who hears "No" crosses that girl off his list.

Termination Condition:

If there is a day when every girl has at most one suitor, we stop and each girl marries her current suitor (if any).

Let's run the Gale-Shapley algorithm on the example from before. On the first morning, boy b_1 serenades girl g_3, boy b_2 serenades girl g_1, boy b_3 serenades girl g_4, boy b_4 serenades girl g_1, and boy b_5 serenades girl g_1. In the afternoon, girls g_1, g_3, and g_4 say "Maybe, come back tomorrow" to boys b_5, b_1, and b_3, respectively. Girl g_1 says "No!" to boys b_2 and b_4, who cross g_1 off their lists that evening.

On the second morning, boy b_1 serenades girl g_3, boy b_2 serenades girl g_2, boy b_3 serenades girl g_4, boy b_4 serenades girl g_3, and boy b_5 serenades girl g_1. In the afternoon, girls g_1, g_2, g_3, and g_4 say "Maybe, come back tomorrow" to boys b_5, b_2, b_4, and b_3, respectively. Girl g_3 says "No!" to boy b_1, who crosses g_3 off his list in the evening.

On the third morning, boy b_1 serenades girl g_2, boy b_2 serenades girl g_2, boy b_3 serenades girl g_4, boy b_4 serenades girl g_3, and boy b_5 serenades girl g_1. In the afternoon, girls g_1, g_2, g_3, and g_4 say "Maybe, come back tomorrow" to boys b_5, b_2, b_4, and b_3, respectively. Girl g_2 says "No!" to boy b_1, who crosses g_2 off his list in the evening.

On the fourth morning, boy b_1 serenades girl g_5, boy b_2 serenades girl g_2, boy b_3 serenades girl g_4, boy b_4 serenades girl g_3, and boy b_5 serenades girl g_1. In the afternoon, the girls realize that each girl has at most one suitor, so all five couples start planning their weddings.

Now let's show that the algorithm works. We need to show that

- The Gale-Shapley algorithm terminates. – We don't want these poor guys singing forever.
- The Gale-Shapley algorithm terminates quickly. – In fact, their singing is pretty bad, so we'd like them to finish as quickly as possible.
- At termination, there are no rogue couples. – This is in fact the main goal. It would be a shame if after all this work, a rogue couple spoiled everything.
- Everyone is married. – Stability is very easy to show if there are no marriages!

We will also analyze the fairness of the protocol. It is better for boys or for girls?

Let's start by showing that this algorithm terminates:

Theorem 1. The Gale-Shapley algorithm terminates within $n^2 + 1$ days.

Proof. We'll prove this theorem by contradiction. Suppose, for contradiction, that the Gale-Shapley algorithm does not terminate in $n^2 + 1$ days. Well, let's notice something that must happen on a day in which the Gale-Shapley algorithm doesn't terminate – it must be that some boy crosses a girl off his list that evening! Why is this? If the Gale-Shapley algorithm doesn't terminate, then some girl must have had at least 2 suitors. If a girl has at least 2 suitors then at least one gets rejected, and that boy crosses that girl off his list. So if the Gale-Shapley algorithm doesn't terminate in n^2+1 days, there are at least n^2+1 names crossed off in total. But at the start, each list is of size n, so the total size of all the lists put together is n^2. So we couldn't have crossed off $n^2 + 1$ names, and thus we have our contradiction.

This is a typical proof technique in Computer Science used to bound the running time of an algorithm. We show that the algorithm is always making progress by some measure. Then, since there is only a finite amount of progress to make, it must eventually terminate. Here the measure is the number of names on the union of the lists.

Next, we'll prove that everyone gets married by the Gale-Shapley algorithm, but first we'll need a couple of lemmas.

Lemma 1. If a boy marries, then he courted every girl he liked better.

Proof. In the Gale-Shapley algorithm, boys cross girls off their lists one at a time in preference order, starting with the girl he likes most. A boy marries the girl (if any) that he is courting at termination. So if a boy marries, he marries his least favorite girl among those he courted. Tough luck for the boy, but at least he likes her better than all the girls he never courted.

Lemma 2. If a boy never marries, then he courted every girl.

Proof. By Theorem 1, the Gale-Shapley algorithm terminates. At the time of termination, the boy is not courting. How can that be? If he is home doing his discrete mathematics homework, then he must have crossed off every girl on his list. So, he has courted every girl.

Lemma 3. A girl marries her favorite among her suitors.

Proof. A girl only rejects a boy when a better one comes along and always keeps stringing along her favorite among those seen so far.

This also means that:

Lemma 4. If a girl is ever courted, she gets married.

Proof. Once a girl has a suitor, she keeps him until she trades up.

Now we can prove that everyone gets married:

Theorem 2. Everyone is married in the Gale-Shapley algorithm.

Proof. We'll show this one by contradiction. Assume, for contradiction, that some boy b is not married. But then, by Lemma 2, boy b has courted every girl. So, every girl has been courted. But then, every girl is married by Lemma 4. But since there are an equal number of boys and girls, it must be the case that every boy (including b) is married. So the theorem is true by contradiction.

Next we'll prove the main result, namely that the Gale-Shapley algorithm always produces stable marriages.

Theorem 3. The Gale-Shapley algorithm produces stable marriages.

Proof. Assume, for contradiction, that there is a rogue couple $\{b, g\}$. Suppose b married g' and g married b' in the Gale-Shapley algorithm. If b married g', but likes g better, then b visited g first by Lemma 1, and g said "no" to b. But then, g must have married someone that she likes better than b by Lemma 3. So, g likes b' better than b, which means that $\{b, g\}$ is not a rogue couple.

Well, who do you think is better off in the Gale-Shapley algorithm? In other words, who has the power, the proposers or the acceptors? Since the girls marry their favorite from among their suitors, and the boys get the worst girls that they court, it seems reasonable to assume that the girls do best. It seems hard to answer this question formally, especially since it isn't even clear what we mean by "doing better". But, in fact, we can show in a very precise and formal way that the algorithm is heavily biased toward the boys. To formalize this, we need to define the set of realistic potential mates.

Let S be the set of all stable matchings. Since the Gale-Shapley algorithm gives a matching, we know that $S \neq \emptyset$. For each person p, we define the realm of possibility for p to be $\{q \mid \exists M \in S,\ \{p, q\} \in M\}$. That is, q is within the realm of possibility for p if and only if there is a stable matching where p marries q.

Some mates just might be out of the question, since no stable pairings are possible if you married them. For example, Brad is just not realistic for Alice since if you ever pair them, Brad and Angelina will form a rogue couple – so there is no stable matching with Brad paired to Alice.

Definition. A person's optimal mate is his/her favorite from the realm of possibility.

An optimal mate must exist, since we know there is at least one stable matching, namely the one produced by the Gale-Shapley algorithm.

Definition. A person's pessimal mate is his/her least favorite from the realm of possibility.

Ok, now here is a pair of shocking results:

Theorem 4. The Gale-Shapley algorithm pairs every boy with his optimal mate!

Theorem 5. The Gale-Shapley algorithm pairs every girl with her pessimal mate!

This is too hard to believe, so let's do the proof.

Proof of Theorem 4. Assume, for contradiction, that some boy does not get his optimal girl (that is, his favorite girl within the realm of possibility). Let b be the first (in time) boy that gets rejected by his optimal girl g (resolving ties arbitrarily). Define b' to be the boy that caused g to reject b in the Gale-Shapley algorithm. Then g prefers b' to b.

Since b is the first to be rejected by the optimal mate in the Gale-Shapley algorithm, b' has not (yet) been rejected by the optimal mate when he is courting g. So, b' likes g at least as much as he likes his optimal mate g^* (g and g^* might be the same person). Let M be a stable matching where b marries g. M exists since g is in the realm of possibility of b. M is not produced by the Gale-Shapley algorithm by assumption. Let g' be the spouse of b' in M. By definition, b' likes g^* at least as much as g' (again, they might be the same person), so b' prefers g to g' since b' likes g at least as much as g^*, whom he likes at least as much as g'. (Note that g can't be the same person as g'.) So b' and g are a rogue couple, which contradicts the fact that M is a stable matching!

Now let's show that the girls get their pessimal mate.

Proof of Theorem 5. Suppose, for contradiction, that there is a stable matching M where there is a girl g who fares worse than in the Gale-Shapley algorithm. Let b be the mate of g in the Gale-Shapley algorithm. Let b' be the mate of g in M. Then g likes b better than b' since she fared worse in M than in the Gale-Shapley algorithm. Let g' be the mate of b in M.

We know that b likes g better than g' since (by Theorem 4) the Gale-Shapley algorithm gives an optimal mate for b. Then b and g form a rogue couple in M, which is a contradiction.

The Gale-Shapley algorithm arises in all sorts of applications. Perhaps the most famous application is in matching fresh MDs to residency programs. Fourth year medical students have to fill out a form with their top 20 choices for residency programs. Teaching hospitals do the same thing with their top choices for doctors. Then the data is fed to the algorithm which matches doctors to hospitals. The doctors find out their assignments on match day, which is a huge event.

The algorithm used is a variant of the Gale-Shapley algorithm, where the hospitals are boys and the doctors are girls, but in this case there are multiple girls for every boy. We still want a solution that is stable so that no swapping will occur – rogue couples can cause chaos and instability in the medical community as well! Actually, the Gale-Shapley algorithm is very civilized. There is no waiting list or delay. Of course, the hospitals get their optimal choices and the doctors get their pessimal choices.

Not surprisingly, the Gale-Shapley algorithm is also used by large dating agencies.

Computer Projects

Choose one from the following list, and write a program with the specified input and output.

The 15-puzzle

Given an initial configuration, solve the 15-puzzle, if possible!

Nim

Given an initial configuration, interactively play the game Nim!

Chomp

Given positive integers m and n, interactively play the game Chomp!

Gale-Shapley algorithm

Implement the Gale-Shapley algorithm! Think about a generalization of the Stable Marriage problem in which certain boy-girl pairs are explicitly forbidden!