I accuse! Part one

The Computer Science Games (CSGames) 2024 CO worked hard to create the challenges, schedules, logistics, etc. After their efforts, they decided to relax and reward themselves by playing a classic game, CLUE.

Unfortunately, Colonel de Lachevrotière forgot the game box at home. Determined not to play another board game, Mrs Prevot came up with an idea: she proposes a new game that recreates the spirit of the CLUE game.

Game Setup

It's a one-on-one game, but an additional person (the dealer) is needed to set up the initial state of the game. Other members of the CO can only be spectators.

The game is played with a small envelope and a reduced deck of cards (specifically, only the hearts from a standard deck, so 13 cards).

The dealer performs the following operations out of sight of the two players:

- He randomly shuffles the deck.
- He randomly selects two cards from the deck and places them in the envelope. These cards are called the guilty cards.
- From the remaining cards, he randomly selects 5 cards and gives them to player 1.
- He gives the remaining 6 cards to player 2.

Of course, as in the real Clue game, all cards have been face down during these operations, so neither player knows what is in the envelope or in the other player's hand.

Game Rules

Players take turns playing. Player 1 goes first. On their turn, a player chooses ONE of the following two actions:

- Draw: **The player randomly draws a card from their opponent's hand.** Of course, the opponent can shuffle their cards behind their back before drawing to ensure that the drawing player is indeed drawing randomly. **Then, the player adds the drawn card to their hand.**
- Accusation: **The player says "I accuse" by naming the two cards they believe are in the envelope.** At this point, the game ends. The envelope is opened. The player wins if their statement is correct; otherwise, the opponent wins.

There is no limit to the number of rounds the game can last. Indeed, if you have understood correctly, a player always draws from a hand containing 6 cards. They can be really unlucky and often draw a card that was just taken from them (which provides no new information).

Your Mission

Calculate the probability that player 1 wins the game, assuming both players have an excellent memory and play optimally (i.e., only make an accusation if it is the best choice).

Flag

The file 'flag-steps-fake.txt' serves as an example to guide you on how to create the FLAG. Except for one big detail, reproduce exactly the steps in 'flag-steps-fake.txt'.

The big detail is, of course, that you must use the correct probability to perform these steps, instead of the $\frac{175}{375}$ provided as an example.

I accuse! Part 2

It seems that the previous version of the game does not satisfy player 1. Due to the calculation you made earlier, he complains that his chances of winning are not close enough to 50% for his liking.

To get closer, he proposes the following parameter changes:

- Play with all hearts AND all spades, so 26 cards, twice as many as before!
- The dealer puts 3 random cards in the envelope instead of 2. Thus, a player who accuses must guess the 3 cards.
- Players receive 11 and 12 cards respectively instead of 5 and 6.
- All other rules remain unchanged.

However, with these new parameters, other CO members are afraid of falling asleep. They claim that they will fall asleep if both players play more than 25 turns each.

Your Mission

Similar to the previous question, players still play optimally. No accusing just to please the spectators who will fall asleep!

Calculate the probability that the number of draws reaches at least 50. A draw is the act of randomly taking a card from the opponent's hand. For example, if both players take 25 turns each without accusing, that makes 50 draws.

Flag

Same format as the previous flag.

Solution Guide

To understand this, let's call the current player the player whose turn it is at the moment. The other player is the opponent.

During the game, we can represent the current state by the ordered pair (i, j) where:

- i: The number of cards in the opponent's hand that are unknown to the current player
- j: The number of cards in the current player's hand that are unknown to the opponent

Firstly, draws are random. So, tactically, the only strategic element of the game is deciding when to make an accusation.

The current player, if they make an accusation right away, has to guess the two guilty cards among $2 + i$ cards because there are $2 + i$ cards they haven't seen. From their perspective, there are $C(2, i + 2)$ possible pairs of guilty cards, so if they accuse, they only have $\frac{1}{C(2,i+2)}$ chance of winning. This is at best $\frac{1}{3}$ if $i > 0$.

The opponent is in the same situation. If the current player does not accuse, the opponent will have $\frac{1}{C(2,j+2)}$ chance of winning if they accuse next.

In conclusion, the current player has no interest in making an accusation if $i > 0$ and $j > 0$ because the opponent would have less than $\frac{1}{2}$ chance of winning their accusation on their turn.

The only case where the current player, in state (i, j), has an interest in making an accusation is if $i = 0$ or $j = 0$. If $j = 0$, it means the opponent has just seen the last non-guilty card they hadn't seen yet; since they have a 100% chance of winning an accusation on their next turn, the accusation must be made immediately before they can take their turn. Furthermore, following this reasoning, since the opponent thinks the same way, the case $i = 0$ never occurs because the opponent will accuse before it's our turn.

Equations

Let's call $P_{i,j}$ the probability that the current player wins the game if the current state is (i, j), respectively.

As explained earlier, we have:

$$P_{i,0} = \frac{1}{C(2, i + 2)}$$

We also have, trivially:

$$P_{0,j} = 1$$

But this last equation is unnecessary because the opponent will never let us take our turn if we know the guilty cards.

The equations become more complex when $i > 0$ and $j > 0$. Let $N = 6$, the number of cards in the opponent's hand before the current player draws from it. In this case, the current player draws a random card from their opponent's hand. They have $\frac{i}{N}$ chance of drawing a card they haven't seen before. In this case, the opponent, in turn, will be in state $(j, i - 1)$ and will have $P_{j,i-1}$ chance of winning. On the other hand, the current player has $\frac{N-i}{N}$ chance of drawing a card they have already seen. In this case, the opponent, in turn, will be in state $(j, i - 1)$ and will have $P_{j,i-1}$ chance of winning. So we have:

$$P_{i,j} = 1 - (\frac{i}{N} P_{j,i-1} + \frac{N - i}{N} P_{j,i})$$

Great! Let's code a program that recursively calculates all $P_{i,j}$! Not so fast... to call a recursive function without looping, we need to avoid circular references. The problem is that $P_{j,i}$ causes a problem in the equation because if we expand it:

$$P_{j,i} = 1 - (\frac{j}{N} P_{i,j-1} + \frac{N-j}{N} P_{i,j})$$

In short, the calculation of $P_{i,j}$ refers to $P_{j,i}$ which refers to $P_{i,j}$... we loop, so we need to expand and isolate $P_{i,j}$:

$$\begin{aligned}
P_{i,j} =& 1 - (\frac{i}{N} P_{j,i-1} + \frac{N-i}{N} P_{j,i}) \\
=& 1 - (\frac{i}{N} P_{j,i-1} + \frac{N-i}{N}(1 - (\frac{j}{N} P_{i,j-1} + \frac{N-j}{N} P_{i,j})) \\
=& 1 - \frac{i}{N} P_{j,i-1} - \frac{N-i}{N}(1 - (\frac{j}{N} P_{i,j-1} + \frac{N-j}{N} P_{i,j})) \\
=& 1 - \frac{i}{N} P_{j,i-1} - \frac{N-i}{N} + \frac{j(N-i)}{N^2} P_{i,j-1} + \frac{(N-j)(N-i)}{N^2} P_{i,j} \\
N^2 P_{i,j} =& N^2 - N P_{j,i-1} - N(N-i) + j(N-i) P_{i,j-1} + (N-j)(N-i) P_{i,j} \\
=& Ni - N P_{j,i-1} + j(N-i) P_{i,j-1} + (N^2 - Ni - Nj + ij) P_{i,j} \\
(Ni + Nj - ij) P_{i,j} =& = Ni - N P_{j,i-1} + j(N-i) P_{i,j-1}
\end{aligned}$$

So, we have:

$$P_{i,j} = \frac{Ni - N P_{j,i-1} + j(N-i) P_{i,j-1}}{(Ni + Nj - ij)}$$

This equation is ready for recursive programming because there is no risk of circular references. Indeed, the calculation of $P_{j,i-1}$ or $P_{i,j-1}$ cannot eventually call $P_{i,j}$ because the total number of cards not seen by both players cannot increase along the way.

The value we are looking for, in the end, is $P_{N,N-1} = P_{6,5}$.

We obtain $P_{6,5} = \frac{6557099}{13634544}$.

Code

It is sufficient to implement the recursive equation above, either by a recursive function compute_p(i,j), or by using dynamic programming to gradually fill a 2D array of all $P_{i,j}$. Dynamic programming is much more efficient because recursive calls will very often repeat the same calculations. However, the parameters are small enough that using a recursive function without dynamic programming works in less than 5 seconds.

For dynamic programming, a 2D array is used to store $P_{i,j}$ for each possible pair (i, j). The array is filled by iterating from the smallest values of $i + j$ to the largest.

Flag

FLAG{6557099/13634544}

Part 2

This question is conceptually easier, as there is no need for algebraic acrobatics to eliminate the risk of circular references.

Recall that for this problem, $N = 12$ since the parameters have changed. Each player draws from a hand that contains 12 cards, not 6.

Let's call $P_{k,i,j}$ the probability that, if we are in state (i, j), there are exactly k draws left in the rest of the game. The game is in state $(N, N-1)$ at the beginning. Thus, the probability that the game has at least 50 draws is

$$1 - \sum_{k=0}^{49} P_{k,N,N-1}$$

Equations of recurrence will again need to be established to find the value of each of these terms, as in the previous question.

Firstly, for tactical reasons explained in the first question, there will be no more draws if $i = 0$ or $j = 0$. Therefore, if $i = 0$ or $j = 0$, $P_{k,i,j} = 1$ for $k = 0$ and $P_{k,i,j} = 0$ for $k > 0$.

However, if $i \geq 1$ and $j \geq 1$, then a draw must be made. After the draw, the current player and the opponent are swapped, so we either end up in state $(j, i-1)$ with probability $\frac{i}{N}$ or in state (j, i) with probability $\frac{N-i}{N}$. In both cases, the rest of the game must end in exactly $k-1$ draws. Thus,

$$P_{k,i,j} = \frac{i}{N} P_{k-1,j,i-1} + \frac{N-i}{N} P_{k-1,j,i}$$

Code

If we directly implement the above recurrence equation, there is no risk of circular references.

However, using only a recursive function to calculate each of the terms in $1 - \sum_{k=0}^{49} P_{k,N,N-1}$ is endless, so dynamic programming is a necessity.

In the provided code, a 3D array is used to calculate the values of $P_{k,i,j}$ for each triplet (k, i, j). The array is filled by iterating from the smallest values of k to the largest.

We find out that the probability of reaching 50 draws is close to 0.67, see exact value below.

Flag

FLAG{1897575782773925539190703878422815958489/28352074999403673815253772280382766448 64}

Code link

https://github.com/desharnc27/incoming-csg-2024-python