Basic Algorithm Project – Game

May 2, 2020

Frank Zhou

Layla Liu

Lily Wang

**Basic Information about the Game**

There are n moves, alternating between Paul (first) and Carole. At each move, the player selects a bit, zero or one. The starting node, denoted e, is the empty string. After u moves the intermediate node will be a binary string of length u. At the end of the game, the leaves are the 2n strings of length n. The values VALUE(x) for the leaves x are set in advance as randomly picked value from the interval [-1, +1]. The result of the game is VALUE(e), the value of the game to Paul.

**General Idea of the Algorithm**

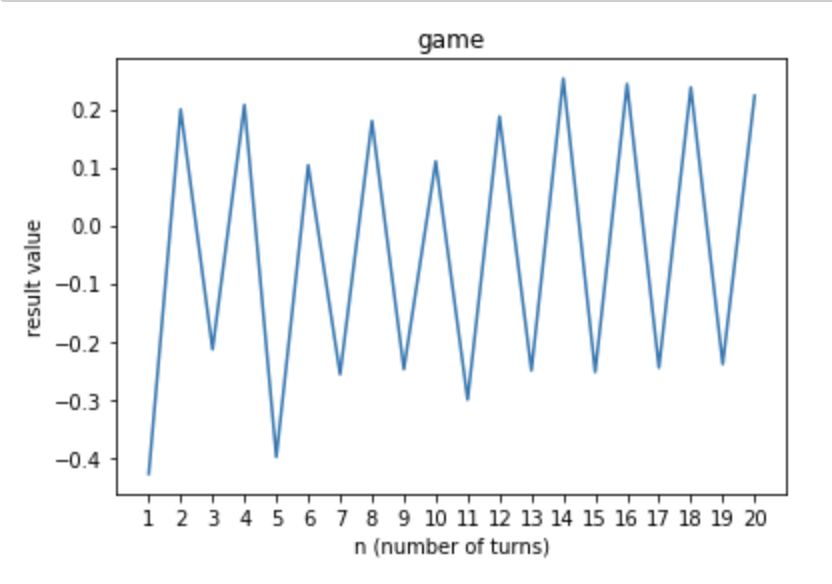
Since in each turn, the player who moves can only choose 0 or 1, then the whole game can be understood as a full binary tree with n+1 levels where n is the total # of moves.

So, we apply a simplified version of DFS – the Backward Introduction in Game Theory: Moving from the bottom to top, given the kth level, we calculate the value of each node in (k-1)th level by choose the larger (if it’s Paul’s turn) or the smaller (otherwise) value of its two children. Repeat doing this in each level until we reach the root, and now the root contains the result of the game.

In practice, instead of building a whole tree, we use arrays to represent each level and free the memory when one level is used, so that the RAM doesn’t explode in running the algorithm. For more information, one can check the Jupyter notebook attached in the zip.

**General Case (Paul goes first, n changes by stride 1)**

(莉莉你从这开始)



Is there an advantage to playing last? Is there an advantage to playing first?

First, when Paul plays first, if n is an odd number, where Paul always finishes, the result is always positive. Interestingly, the peak of the outcome (which satisfies Paul’s wish) appeared when n = 5, and it converges to 0.23 when n is getting large.

When n is an even number, where Carole always finishes, the result is always negative.

The result reached the bottom (which satisfies Carole’s wish) when n = 6, and we can

infer that it will converge to 0.23 when n is really large.

Similarly, when Carole plays first, the player who played last also got the result that

catered his/her benefit.

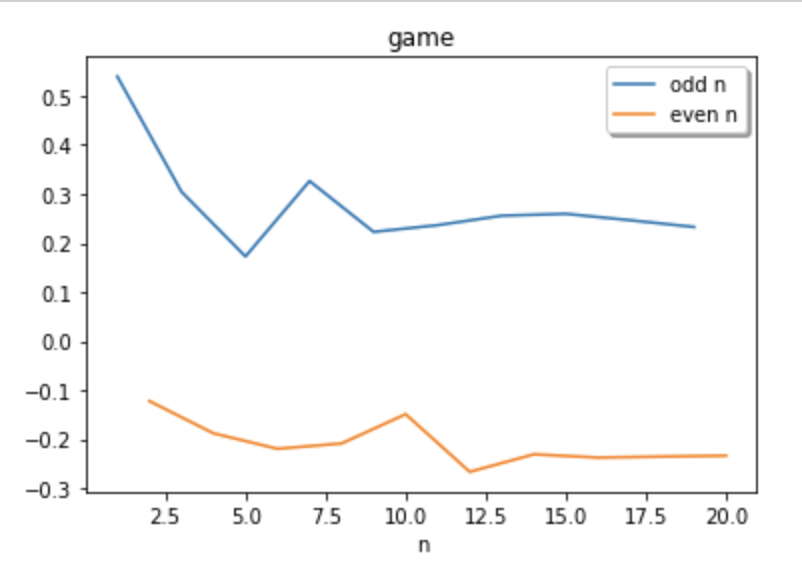
From our data, it is beneficial for the player to play last. For Paul, he can get more

money and Carole can pay less money. Additionally, whether Paul plays first or Carole

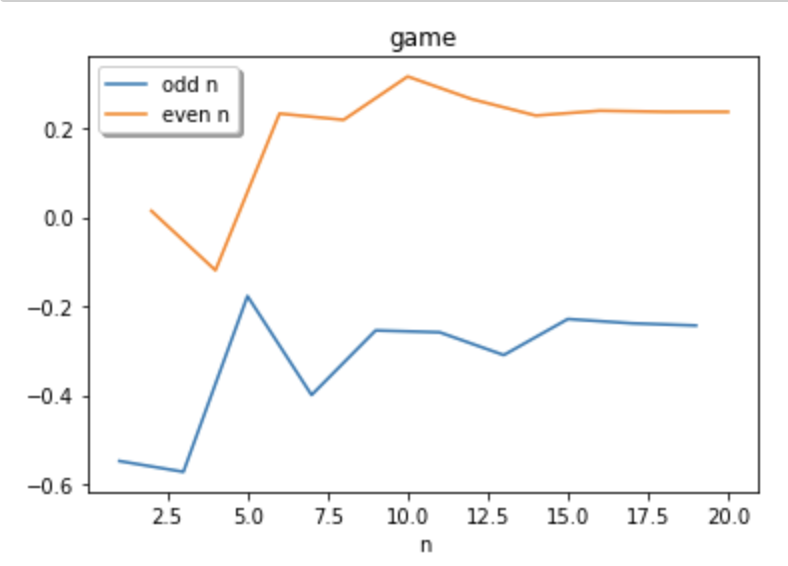
plays first, there is no influence on the result.

Our analysis is that the last move decides the final value,

Plot for Paul going first.



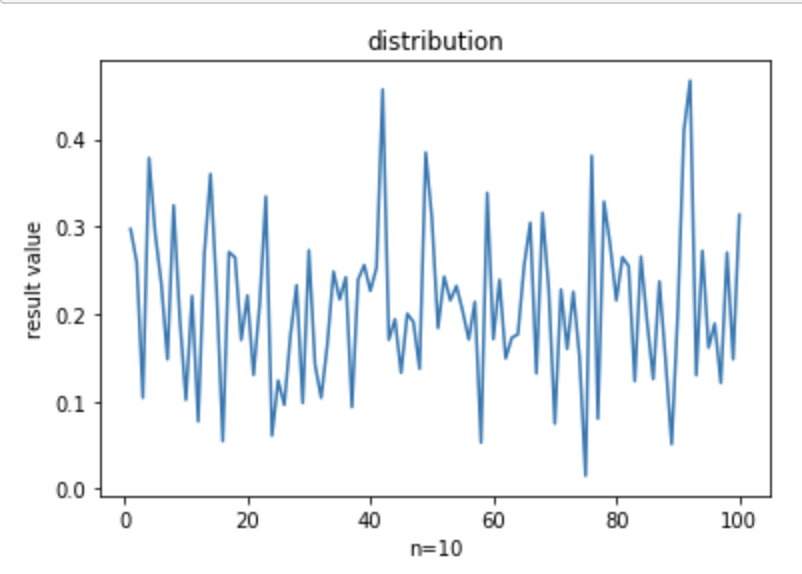
Plot for Carole going first.



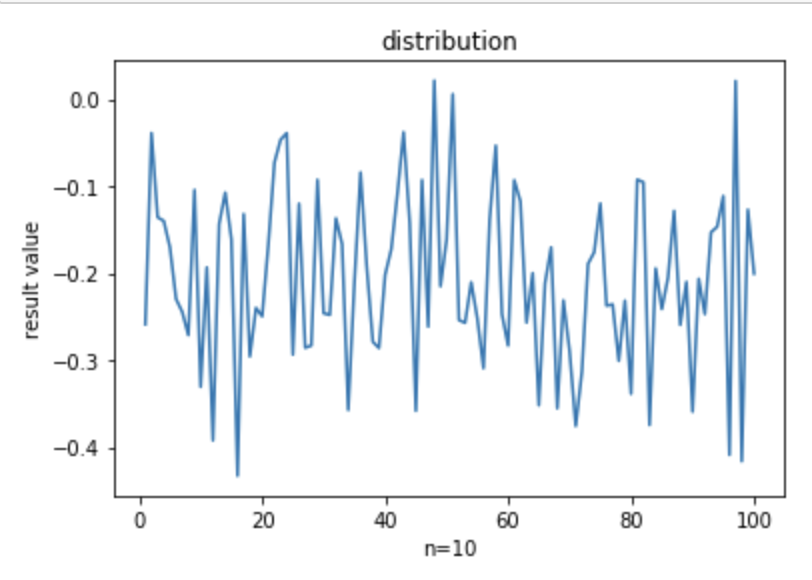
We took n=10 with Carole going first and did 100 runs. The variance of the results is approximately 0.008, which seems relatively low.

With n=10 and Paul going first, the variance of the results is approximately 0.010, which is also relatively low.

Below is the plot for the results in Q3. With Carole going first and even-number of moves, the result values are always positive.



Below is another plot for the results in Q3. With Paul going first and even-number of moves, the result values are always negative.



The limiting behavior as n→∞.

At first, the results seemed to be random when n is small. However, as we make n

bigger and bigger, there appeared a certain pattern of the results. That is, a regular

fluctuation of the outcome of the same absolute value approximately 0.23, one is positive and the next is negative, etc. This is a very interesting behavior as n tends to be big. Our conjecture is that when n is large (bigger than 20 or so) if it’s an even number, the result will be approximately 0.23, while -0.23 when it’s odd.

From the plot, we learn that the result value oscillates around 0.2 and -0.2 as n→∞.