Briefly Explore the Combinatorial Game

Gong Cheng

Abstract:
The paper talks about a series of games called combinatorial games. Combinatorial games are two-person games with perfect information, no chance moves, and a win-or-lose outcome. To be precise, a combinatorial game is a game that satisfies the following conditions. The paper also talks about the rules of the combinatorial game, such as how many players the game has and the rules of the game specify for both players and each position that moves to other positions a legal move. Then, the meaning of the P-positions and N-positions were introduced. After that, the paper talks about the typical combinatorial game- the game of Nim, another example of the combinatorial game, to illustrate the paper's point of view. The paper talks about the whole rules of this example game, such as the number of players and the initial configuration of the piles/heaps. The author gives several strategies and talks about why these strategies can help the players win combinatorial games. The paper gives two cases to illustrate the game rules and give a better understanding after all. In the end, the paper gives a Python3 program for combinatorial games, making the ideas of combinatorial games more comprehensive.
Keywords: combinatorial game, Nim, position

1. Brief Introduction to Combinatorial Game

1.1 What is a Combinatorial Game

Combinatorial games are two-person games with perfect information, no chance moves, and a win-or-lose outcome. To be precise, a combinatorial game is a game that satisfies the following conditions. (Siegel, 2013)
(1) There are two players.
(2) There is a set, usually finite, of possible positions in the game.
(3) The rules of the game specify for both players, and each position that moves to other positions is a legal move. If the rules make no distinction between the players, that is, if both players have the same options of moving from each position, the game is called impartial; otherwise, the game is called partisan.
(4) The players alternate moving.
(5) The game ends when a position is reached from which no moves are possible for the player whose turn it is to move. Under the normal play rule, the last player to move wins. Under the misery play rule, the last player to move loses. If the game never ends, it is declared a draw. However, we shall nearly always add the following condition, called the Ending Condition. This eliminates the possibility of a draw.
(6) The game ends in a finite number of moves, no matter how played.
Besides, some important things have been omitted above. No random moves, such as the rolling of dice or the dealing of cards, are allowed. This rules out games like backgammon and poker. A combinatorial game is a game of perfect information: simultaneous and hidden moves are not allowed. This rules out battleship and scissors-paper-rock. No draws in a finite number of moves are possible. This rules out tic-tac-toe. (Erdös, & Selfridge, 1973)

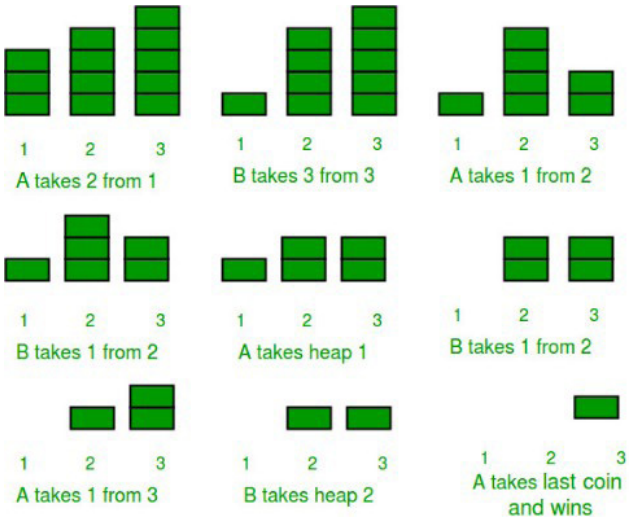
1.2 P-positions and N-positions

So now, let's talk about P-positions and N-positions. Those positions winning for the Previous player (the player who just moved) are called P-positions, and those positions winning for the Next player to move are called N-positions. In impartial combinatorial games, one can find in principle which positions are P- P-positions and N-positions by (possibly transfinite) induction using the following labeling procedure starting at the terminal positions. We say a position in a game is terminal if no moves from it are possible.
The following three statements define the characteristic property of P-positions and N-positions recursively.
(1) All terminal positions are P-positions.
(2) From every N-position, there is at least one move to a P-position.
(3) From every P-position, every move is to an N-position.

2. A Typical Combinatorial Game- the Game of Nim

The following rules describe the Game of Nim- "Given several piles in which each pile contains some numbers of stones/coins. In each turn, a player can choose only one pile and remove any number of stones (at least one)

from that pile. The player who cannot move is considered to lose the game (i.e., one who takes the last stone is the winner).” For example, consider that there are two players- A and B, and initially there are three piles of coins having 3, 4, Five coins in each of them, as shown below. We assume that the first move is made by A. See the figure below for a clear understanding of the whole gameplay.



Game model

In conclusion, this game depends on two factors:
(1) The player who starts first.
(2) The initial configuration of the piles/heaps.
We can predict the winner of the game before even playing the game! Nim-Sum: The cumulative XOR value of the number of coins/stones in each pile/heap at any point of the game is called Nim-Sum at that point. “If both A and B play optimally (i.e., they don’t make any mistakes), then the player starting first is guaranteed to win if the Nim-Sum at the beginning of the game is non-zero. Otherwise, if the Nim-Sum evaluates to zero, then player A will lose.” Here are the optimal strategies.
(1) If the XOR sum of ‘n’ numbers is already zero, then there is no possibility to make the XOR sum zero by a single reduction of a number.
(2) If the XOR sum of ‘n’ numbers is non-zero, then there is at least a single approach by which the XOR sum is zero if you reduce a number.
Initially, two cases could exist.
Case 1: Initial Nim Sum is zero. In this case, if played optimally, B wins, which means B would always prefer to have a Nim sum of zero for A’s turn. So, as the Nim Sum is initially zero, whatever number of items A removes, the new Nim Sum would be non-zero (as mentioned above). Also, as B would prefer the Nim sum of zero for A’s turn, he would then play a move to make the Nim Sum zero again (which is always possible, as mentioned above). The game will run as long as there are items in any of the piles, and in each of their respective turns, A would make Nim Sum non-zero, and B would make it zero again, and eventually, there will be no elements left, and B being the one to pick the last wins the game. It is evident by the above explanation that the optimal strategy for each player is to make the Nim Sum for his opponent zero in each of their turn, which will not be possible if it’s already zero.
Case 2: Initial Nim Sum is non-zero. Going by the optimal approach, A would make the Nim Sum zero(which is possible as the initial Nim Sum is non-zero, as mentioned above). Now, in B’s turn, as the Nim Sum is already zero, whatever number B picks, the Nim Sum would be non-zero, and A can pick a number to make the Nim Sum zero again. This will go as long as there are items available in any pile. And A will be the one to pick the last item. So, as discussed in the above cases, it should be obvious that the Optimal strategy for any player is to make the Nim Sum zero if it’s non-zero, and if it is already zero, then whatever moves the player makes now can be countered.
Now, let’s see the coding of a Python3 program that implements the game of Nim.

```
# A Python3 program to implement Game of Nim. The program
# assumes that both players are playing optimally
, import random

COMPUTER = 1
HUMAN = 2
# A Structure to hold the two parameters of a move#
move has two parameters-
# 1) pile_index = The index of pile from which stone is
```

```
#                    going to be removed
# 2) stones_removed = Number of stones removed from the
#                        pile indexed = pile_index */


Class move:
    def___init_(self): self.
        pile_index = 0
        self.stones_removed = 0
# piles[] -> Array having the initial count of stones/coins
#            in each piles before the game has started.
# n     -> Number of piles

# The piles[] are having 0-based indexing

# A function to output the current game state.
def showPiles(piles, n): print("Current
    Game Status -> ")print(*piles)

# A function that returns True if game has ended and#
False if game is not yet over
def gameOver(piles, n):
    for i in range(n):
        if (piles[i] != 0):
            return False

    return True

# A function to declare the winner of the game
def declareWinner(whoseTurn):
    if (whoseTurn == COMPUTER):
        print("\nHUMAN won")
    else:
        print("\nCOMPUTER won")
```

```
    return


# A function to calculate the Nim-Sum at any point#
of the game.
def calculateNimSum(piles, n):
    nimsum = piles[0]
    for i in range(1, n):
        nimsum = nimsum ^ piles[i]
    return nimsum

# A function to make moves of the Nim Game
def makeMove(piles, n, moves):
    nim_sum = calculateNimSum(piles, n)

    # The player having the current turn is on a winning
    # position. So he/she/it play optimally and tries to make
    # Nim-Sum as 0
    if (nim_sum != 0):
        for i in range(n):

            # If this is not an illegal move#
            then make this move.
            if ((piles[i] ^ nim_sum) < piles[i]):

                moves.pile_index = i
                moves.stones_removed = piles[i]-(piles[i] ^ nim_sum)
                piles[i] = (piles[i] ^ nim_sum)
                break

    # The player having the current turn is on losing
```

```
    # position, so he/she/it can only wait for the opponent
    # to make a mistake(which doesn't happen in this program
    # as both players are playing optimally). He randomly
    # choose a non-empty pile and randomly removes few stones
    # from it. If the opponent doesn't make a mistake,then it
    # doesn't matter which pile this player chooses, as he is
    # destined to lose this game.

    # If you want to input yourself then remove the rand()
    # functions and modify the code to take inputs.
    # But remember, you still won't be able to change your
    # fate/prediction.
    else:
        # Create an array to hold indices of non-empty pilesnon_
        zero_indices = [None for _ in range(n)]
        count = 0
        for i in range(n):
            if (piles[i] > 0): non_zero_
                indices[count] = icount += 1

        moves.pile_index = int(random.random() * (count))moves.
        stones_removed = 1 + \
            int(random.random() * (piles[moves.pile_index]))
        piles[moves.pile_index] -= moves.stones_removed

        if (piles[moves.pile_index] < 0):piles[moves.pile_
            index] = 0

    return

# A C function to play the Game of Nim
def playGame(piles, n, whoseTurn):
    print("\nGAME STARTS")
    moves = move()
```

```
    while (gameOver(piles, n) == False):
        showPiles(piles, n) makeMove(piles,
        n, moves)

        if (whoseTurn == COMPUTER):

            print("COMPUTER removes", moves.stones_removed, "stones
                  from pile at index ", moves.pile_index)
            whoseTurn = HUMAN

        else:
            print("HUMAN removes", moves.stones_removed,
                  "stones from pile at index", moves.pile_index)
            whoseTurn = COMPUTER

    showPiles(piles, n)
    declareWinner(whoseTurn)return

def knowWinnerBeforePlaying(piles, n, whoseTurn):
    print("Prediction before playing the game -> ", end="")if
    (calculateNimSum(piles, n) != 0):

        if (whoseTurn == COMPUTER):
            print("COMPUTER will win")
        else:
            print("HUMAN will win")

    else:

        if (whoseTurn == COMPUTER):
            print("HUMAN will win")
        else:
            print("COMPUTER will win")

    return
```

```
    # Driver program to test above functions#
    Test Case 1
    piles = [3, 4, 5]n =
    len(piles)

    # We will predict the results before
    playing# The COMPUTER starts first
    knowWinnerBeforePlaying(piles, n, COMPUTER)

    # Let us play the game with COMPUTER starting first
    # and check whether our prediction was right or not
    playGame(piles, n, COMPUTER)
```

```
# This code is contributed by phas-
ing17
```

Conclusion

In recent years, research on some games combined with computational complexity or algorithm analysis in computer science has also been quite active in the research circle of combinatorial game theory. The paper not only gives an understanding of the combinatorial game but also hopes that future study could focus on the computational complexity and algorithm analysis to further understand this field.

References

Siegel, A. N. (2013). *Combinatorial game theory* (Vol. 146). American Mathematical Soc..

Erdös, P., & Selfridge, J. L. (1973). On a combinatorial game. *Journal of Combinatorial Theory, Series A*, *14*(3), 298-301.

The Game of Nim. (n.d.). Code Review Stack Exchange. https://codereview.stackexchange.com/questions/284216/the-game-of-nim