CMSC 474, Introduction to Game Theory

Combinatorial Games and The Games of NIM

Mohammad T. Hajiaghayi

University of Maryland

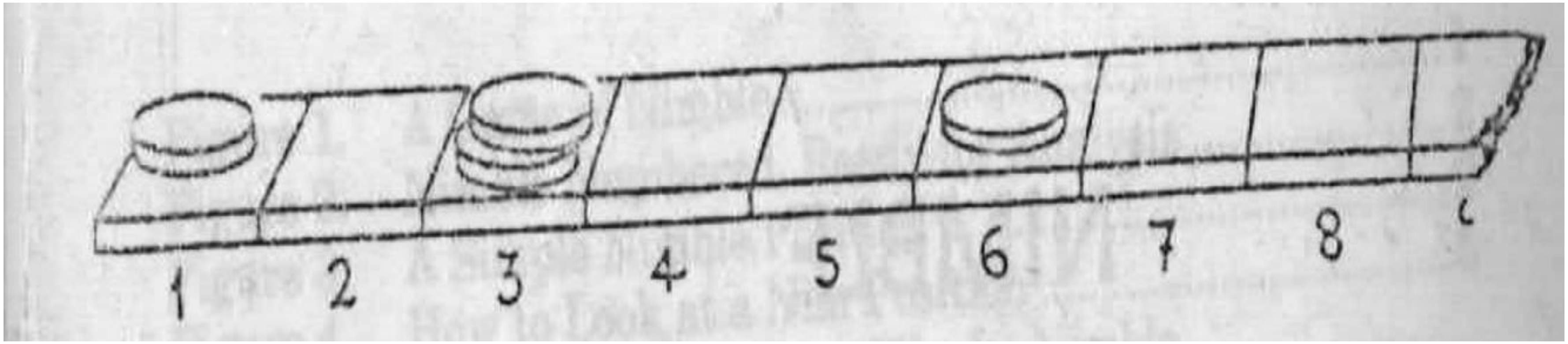
Combinatorial Games

- A two-player **combinatorial game** is a perfect-information extensive-form game requiring:
  - Two player: P_1 and P_2
  - Finitely many positions and a fixed starting position
  - A player strategy is a set of moves from his/her current position to another position
  - The player who cannot move loses the game
  - Play always ends
  - Players have complete information

    (since the game is perfect-information)
  - No chance (probabilistic) moves
- Famous examples:
  - Go, Chess, Checkers, Tic-Tac-Toe, Hex, NIM games (this class), etc.
  - NIM is indeed a family of games and we show only a few examples in this session

Nimble

- **Nimble** is a (two-player) combinatorial game



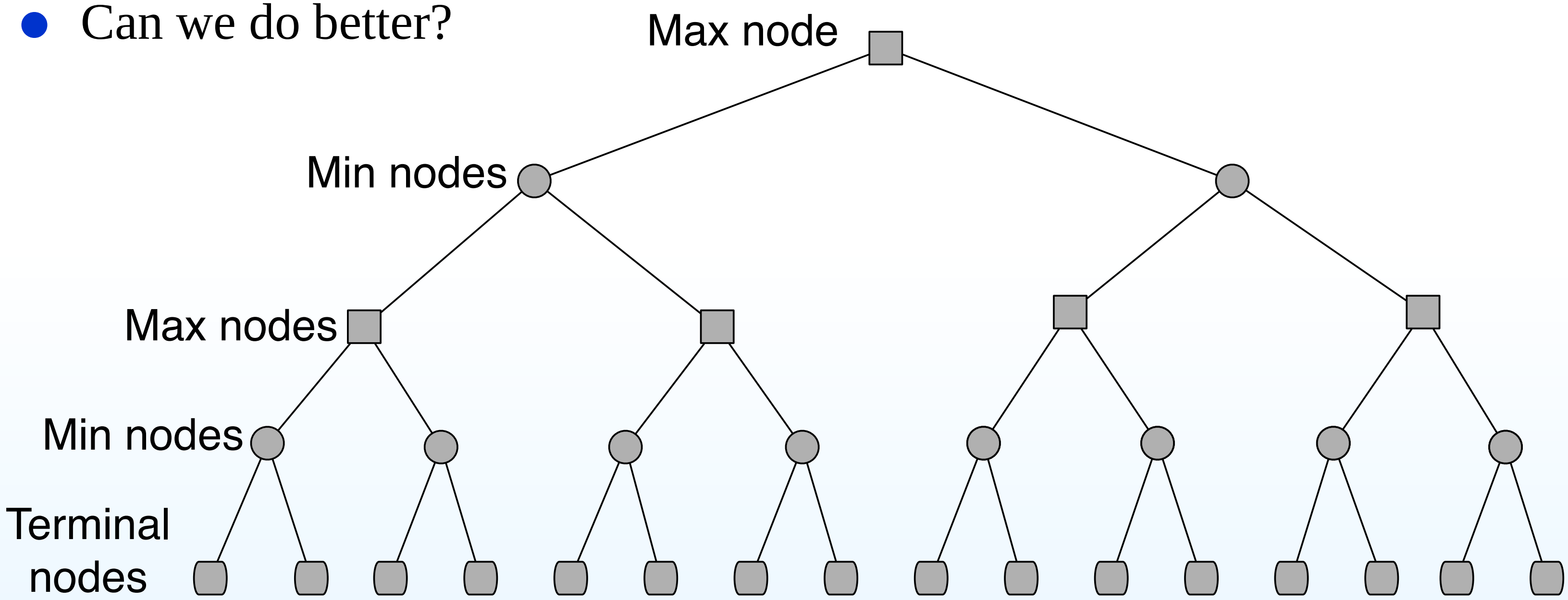
- Put some coins on a strip of squares
- Take turns, moving just one coin to the left.
- No other restrictions:
  - You can jump onto or over other coins, even clear off the strip.
  - You can have any number of coins on a square
- A player who cannot move loses (i.e., when the strip is clear)
- Have any of you seen this game before?

NIM

- **NIM** is another (two-player) combinatorial game
- Start with n piles (heaps) of stones each has at most m stones
- Players take turns to move. In each turn:
  - A player selects one of the piles, and
  - Takes as many stones from it as he/she likes: perhaps the whole pile, but at least one stone
- A player who cannot move loses
- Since it is a perfect-information extensive-form game, we can draw its game tree

NIM from Computational Point of View

- Let b = maximum branching factor
- Let h = height of tree (maximum depth of any terminal node)
- As we have seen, the number of nodes in the game tree is $O(b^h)$
- Now what are b and h for NIM?
- In the worst case $b=nm$ and $h= nm$. WHY?
- Worst-case time complexity = size of the game tree = $O((nm)^{nm})$
- Just think about $n= 5$ and $m= 100$?
- Can we do better?



Improved Running Time via Memoization

- ***Memoization*** is a technique for improving the performance of recursive (e.g. backtracking) algorithms
- It involves rewriting the recursive algorithm so that as answers to problems are found, they are *stored in an array*.
- Recursive calls can look up results in the array rather than having to recalculate them
- Memoization improves performance because partial results are never calculated twice
- What would be the size of the array then?
  - We need to keep track of the winner for each possibility of piles of stones
  - Since each pile can have at most m stones (and thus $m+1$ possibilities), and we have n piles, the size is of $O((m+1)^n)$
- It is much better than the previous bound $O((nm)^{nm})$ but still too high.
- Can we do better?

First Deeper Understanding of Two Piles

- Say you have two heaps of size k and h each
- ***Theorem***: If $k=h$ then the *second player* always can *win*; otherwise the first player can always win.
- ***Proof***: By induction on $k+h$:
- Basis of the induction: $k+h= 0$ (second player wins) or even $k+h=1$ (first player wins)
- Induction Hypothesis: For $k+h< p$, the statement of Thm is correct. What about $k+h=p$?
  - If $k= h$: after any move of the first player (say taking r stones from one pile), two piles of unequal size ($k-r\neq h$) remains; by Induction Hypothesis (since $k-r+h<p$) the first player now (i.e., the original second player) wins
  - If $k \neq h$ (say $k< h$): the first player takes $h-k$ from the pile of size h to make both piles equal; by Induction Hypothesis the second player now (i.e., the original first player) wins.
- Note that for the case of $k= h$ you can also think of the second player always ***mirroring*** moves of the first player (i.e., by taking the same number of stones from the other pile)].

Any Number of Piles

- We need a generalization of *being equal*
- We define **NIM sum** (a.k.a XOR) of numbers $a_1, a_2, \dots, a_n$ (pile sizes):
  - Write each number as a binary number
  - Add the piles modulo 2 in each column (i.e., if the number of ones in the column is odd the result is 1; otherwise 0)
  - The final non-negative number is the NIM sum of piles.

An example:

with three piles, one of size 6, one of size 4, and one of size 3.

- Write the number of stones in each pile as a binary number.
- Add the piles, modulo 2 in each column.
- This nonnegative number is the Nim sum of the piles.

6	1	1	0
4	1	0	0
3		1	1

6	1	1	0
4	1	0	0
3		1	1
	0	0	1

The General Case

- A very similar theorem to that of the two pile case
- ***Theorem***: If the NIM sum is *zero* then the *second player* always can win; otherwise the first player can always win.
- Note that for two piles $k=h$ if and only if NIM sum is equal to zero

  (thus the above Thm generalizes the previous one)
- ***Proof***: By induction on $a_1+a_2+\ldots+a_n$ (sum of pile sizes):
- Basis of the induction: $a_1+a_2+\ldots+a_n= 0$ (second player wins) or even $a_1+a_2+\ldots+a_n = 1$ (first player wins)
- Induction Hypothesis: For $a_1+a_2+\ldots+a_n < p$, the statement of Thm is correct. What about $a_1+a_2+\ldots+a_n = p$.
  - If NIM sum is zero: after any move of the first player ***the NIM sum becomes non-zero*** and thus by Induction Hypothesis (since sum of sizes becomes strictly less) the first player now (i.e., the original second player) wins
  - If NIM sum is non-zero: the first player can always make ***the NIM sum zero by taking from one pile*** and thus by Induction Hypothesis (since the sum of sizes becomes strictly less) the second player now (i.e., the original first player) wins.

Zero NIM Sum Becomes Non-zero After Each Move

- Say the first player chooses a pile *i* and makes its number of stones $a_i' < a_i$
- Consider the first bit from the ***left*** that a_i' and a_i are different.
- It means in the corresponding bit column, the number of ones was *even* before and becomes *odd* now.
- Thus the corresponding bit in the new NIM sum becomes one
- It means the new NIM sum is non-zero.

$$
\begin{array}{rr}
 & 110101000 \\
 & 101000011 \\
a_i': & 000001010 \\
a_i: & 000011110 \\
 & 011110101 \\
\hline
 & 000000000 \\
 & 00001\mathrm{xxxx}
\end{array}
$$

Non-zero NIM Sum Can Become Zero After A Move

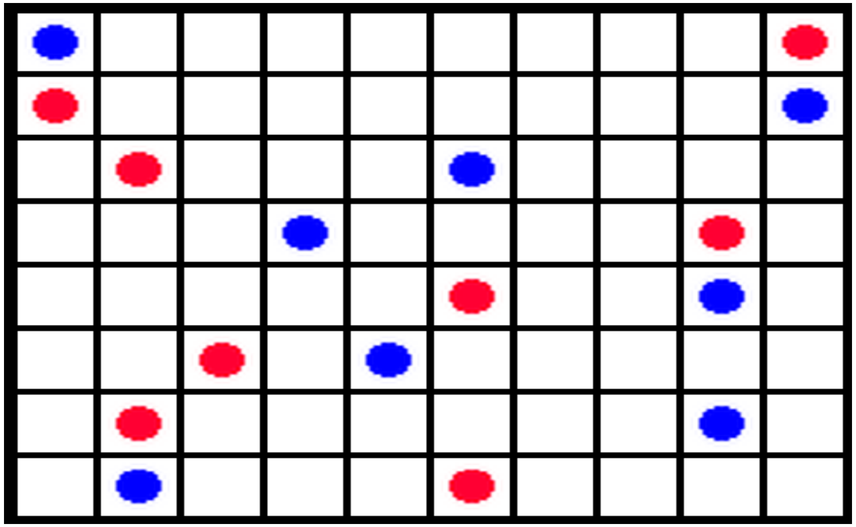
- Consider the first bit from the ***left*** in which the NIM sum is 1.
- There should be an a_i which has 1 in the column corresponding to the bit (since the number of ones is *odd* in the column)
- Staring from that bit to the right, reverse each bit of a_i if the corresponding NIM sum bit is 1 to obtain a_i'
- This makes the new NIM sum zero
- Note that the new number $a_i' < a_i$, since the first different bit from the left (the most significant bit of difference) is 1 in a_i and zero in a_i'
- Thus decreasing the number of stones in pile *i* from a_i to a_i' (i.e., by taking $a_i - a_i'$ stones) makes the NIM sum zero.

```
       110101000
a_i':  100101111
a_i:   101000011

       000011110

       010011001
       ---------
       001101100
       000000000
```

Improved Time Complexity

- To find out who can always win, we only need to compute the NIM sum
- Takes $O(n \log m)$ to obtain binary representation of ALL pile sizes ($O(\log m)$ for each)
- Takes $O(n \log m)$ to obtain the NIM sum and thus the winner.
- Overall only $O(n \log m)$ instead of $O((m+1)^n)$ *or even* $O((nm)^{nm})$
- *HUGE improvement!!!*

Games of Soldiers: Northcott's Game



- **Northcott's** is another (two-player) combinatorial game
- There is just one checker of each color on each row of a checker-board
- Players take turns
- Each player in each turn to move, slides one of his/her checkers any number of squares in its own row without
  - Jumping over the opponent's checker, or
  - Going off the board
- A player who cannot move loses
- It is NIM with a caveat!!!