Rare Topics in Math

Agenda

- Game theory
  - Game trees
  - Nim
- Cycle finding

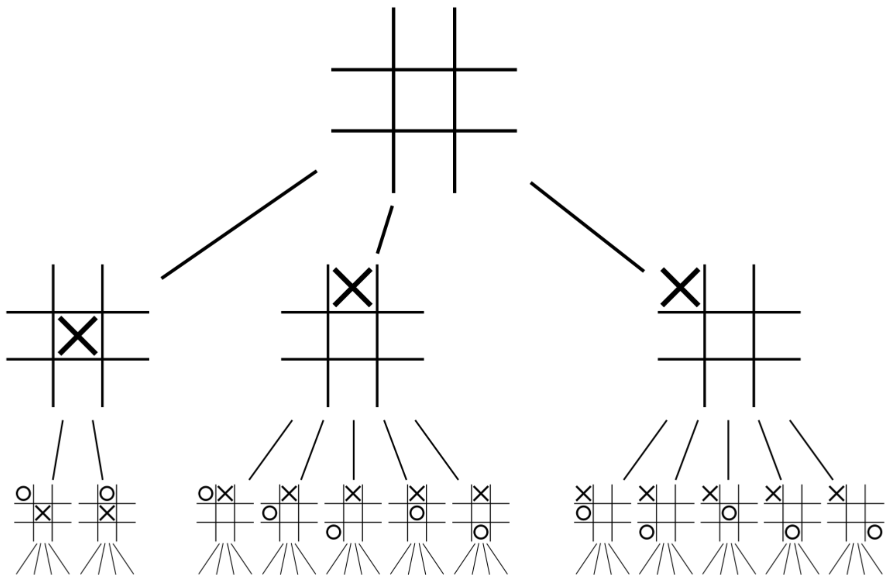
Game Theory

Game Theory

- Game theory is a branch of math which studies strategic decision making
- Game theory doesn’t always deals with “games” in the classic meaning
- Many games appear in competitive programming
- Usually, the games will be:
  - Zero sum – If one player wins, it means the other lost the same amount
  - Perfect Information – No hidden details (as in Poker) or randomness (as In Backgammon)
  - Usually two players games
- Examples: Chess, Tic-Tac-Toe, Nim, Chomp, invented games
- The most common question is – given a game rules and state – which player should win?
  - Assuming perfect play by both sides

Game trees

- A game tree is a tree where each vertex represent a state of the game, and the sons of each vertex are the states that are obtainable from the current state
- Tic tac toe game tree (up to symmetries):



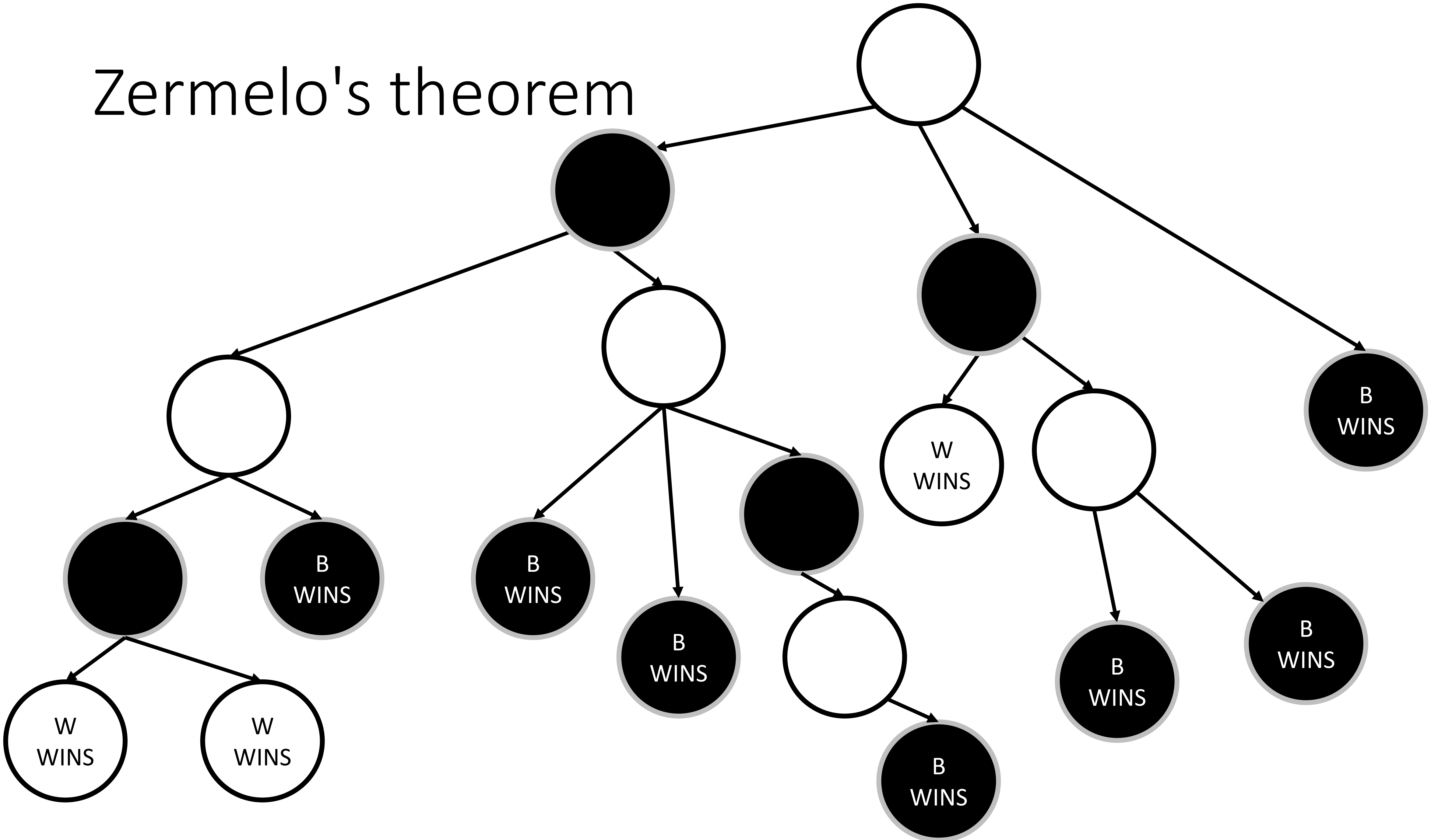
Zermelo's theorem

- Zermelo's theorem: In an turn-based game, where both players have perfect knowledge either:
- The first player can force a win (Connect 4)
- The second player can force a win (Chomp)
- Both players can force a draw (Tic-Tac-Toe, Checkers)

Zermelo's theorem

- Zermelo's theorem proof (disregarding draws):
- The proof is to show an algorithm that decides who wins, we will traverse the tree post-order:
  - Mark each leaf as either “White wins” or “Black wins”
  - While not all vertices are marked:
    - For each node with all its descendants marked:
      - If it is white turn and there is “White wins” son mark it as “White wins”
      - Otherwise (all sons are black wins), white has no good move and it is “Black wins”
      - And vice versa for black
- At the end the root will be “White wins” or “Black wins”, the winning player should choose winning state at each turn and he will win

Zermelo's theorem

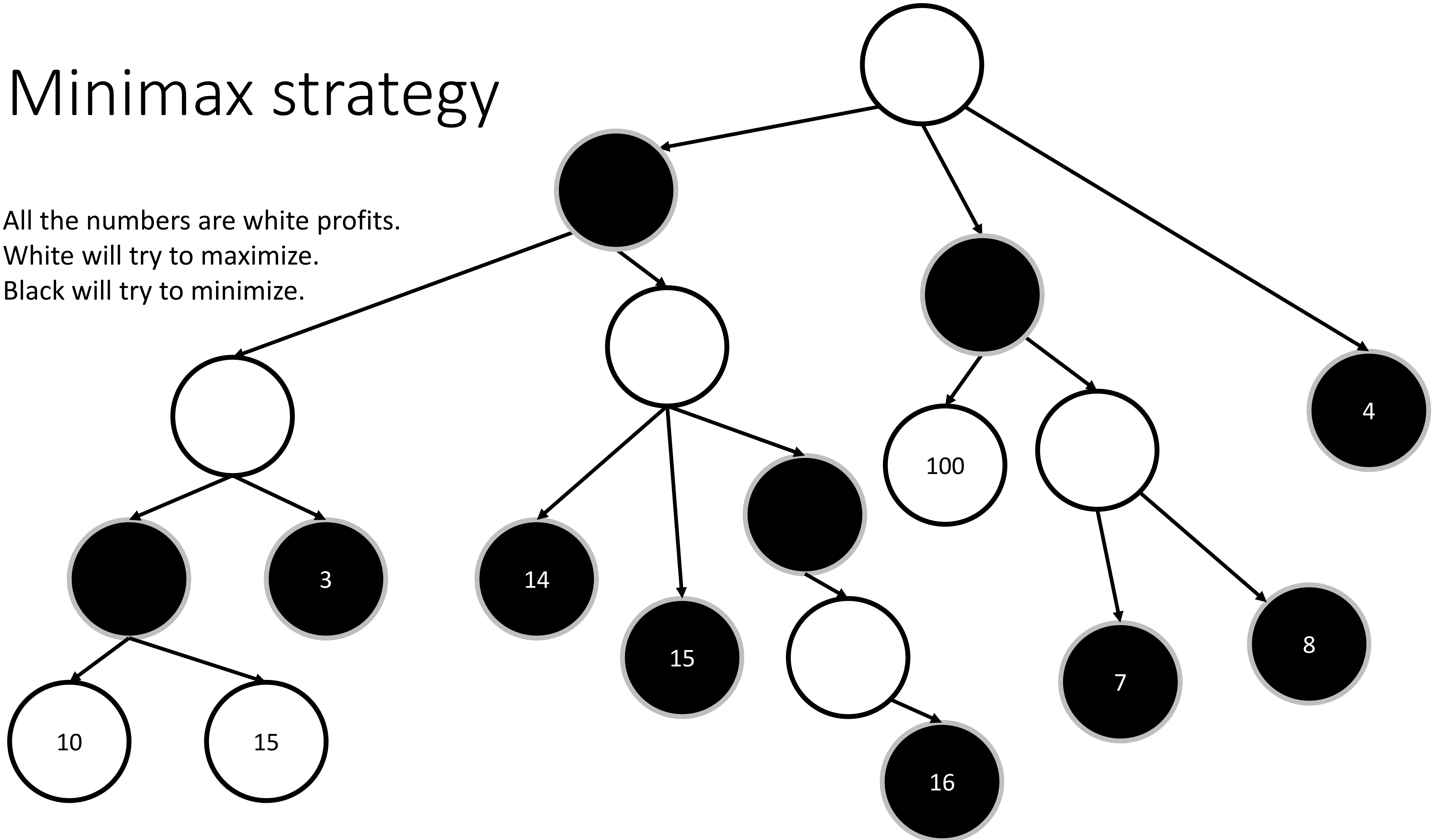


Minimax strategy

- In some cases, the player doesn’t simply wins, there is a certain profit he need to maximize
- We discuss zero-sum games, so the profit for white is the lose for black
- What is the best profit that white can get?
- Again we will, traverse the tree postorder:
- The profit of a leaf is given
- In White turn, he will try to maximize his profit.
- In Black turn, he will try to minimize black profit.
- This strategy is called the minimax strategy

Minimax strategy

All the numbers are white profits.
White will try to maximize.
Black will try to minimize.



Nim

- One popular game in competitive programming is Nim:
- The game rules are:
  - There are k heaps of objects
  - In each turn, a player takes as many objects as he wants from a single heap
  - The goal is to be the last player which takes an object

Nim - Strategy

- Optimal strategy:
- It can be shown that losing positions are positions where to xor of all the piles is zero
  - The sum will be marked by S
- For example: in the state (3,4,5) the xor is $S = 011 \oplus 100 \oplus 101 = 010$
- We can remove 2 objects from the first pile and we will be in: (1,4,5), its xor is $S = 001 \oplus 100 \oplus 101 = 000$, a losing position
- How to find from which pile to remove?
- For each pile, p_i compute if $p_i \oplus S < p_i$ then remove $p_i - p_i \oplus S$ from this pile

Nim

Do you want to play first, or should the other player play first?

$S = \; 5 \oplus 4 \oplus 3 = 2 \neq 0$

You should play first!

$S \oplus 5 = 7 > 5$ ✗

$S \oplus 4 = 6 > 4$ ✗

$S \oplus 3 = 1 \leq 3$ ✓

Change this heap to 1

Nim

Second player will play some random move.
Any move will lose.

$$S = 5 \oplus 4 \oplus 1 = 0$$

Nim

$$S = 2 \oplus 4 \oplus 1 = 7$$

$S \oplus 7 = 5 > 2$ ✘

$S \oplus 4 = 3 \leq 4$ ✓

Change this heap to 3

$S \oplus 1 = 6 > 1$ ✘

Nim

Random move…

Nim

$S = 3 \oplus 1 = 2$

$S \oplus 3 = 1 \leq 3$ ✓

Change this heap to 1

$S \oplus 1 = 3 > 1$ ✗

Nim

Random move…

You take the last coin, and win!

Cycle Finding

Cycle finding

- Given a finite set S, and a function $f: S \rightarrow S$, and a value x_0
- The iterated sequence of these values is:
$$x_0, x_1 = f(x_0), x_2 = f(x_1), \ldots$$
- Since S is finite at some point the sequence will start to repeat itself
- Example 1: $f(x) = (7x + 5)\%12, x_0 = 4$
- The sequence is: $4,9,8,1,0,5,4,9,8, \ldots$
    - The period is 6
- Example 2: $f(x) = (3x + 1)\%4, x_0 = 7$
- The sequence is: $7,2,3,2,3,2,3 \ldots$
    - The period is 2, and there is a non-periodic prefix of size 1

Cycle finding

- We will denote the size of the prefix by μ and the period by λ
- Problem, given f and x_0, find μ and λ
- Trivial algorithm:
- Apply f iteratively, and store the values and their positions in a map
- Do this until a value repeats
- Complexity: $O\big((\mu + \lambda)\log(\mu + \lambda)\big)$ (using map) or $O\big((\mu + \lambda)\big)$ on average (using unordered map)
- Space complexity: $O\big((\mu + \lambda)|s|\big)$ ($|s|$ is the size of an element of S)

Floyd’s Cycle finding algorithm

- Sometimes the space complexity of the trivial algorithm is too big
- Observation: $\forall i \geq \mu, k \in \mathbb{N}$: $x_i = x_{i+k\lambda}$
- So for $i = k\lambda$, we get $x_i = x_{2i}$. Let’s look for such i
- We will have two indices, the tortoise  and the hare 
- For each step the tortoise will do, the hare will do two
- Example:

$k\lambda = 6$

5 6 2 4 3 9 7 3 9 7 3 9 7 3 9 7 3 9 7

Floyd’s Cycle finding algorithm

- Now we have a difference of $k\lambda$ between the rabbit and the hare
- Second step, find μ
- Take the hare back to the start, this will maintain the difference of $k\lambda$
- After μ steps they will have the same value

$$\mu = 4$$

5 6 2 4 3 9 7 3 9 7 3 9 7 3 9 7 3 9 7

Floyd’s Cycle finding algorithm

- Now we only need to find λ
- Take the tortoise back to where the period starts (in practice, copy x_μ from the hare)
- And move the hare until they have the same value

$$\lambda = 3$$

5 6 2 4 3 9 7 3 9 7 3 9 7 3 9 7 3 9 7

Floyd's Cycle finding algorithm

- Summary:
- Find $k\lambda$: both start from x_0, for each step of the tortoise, the hare do two. When the values are equal we found $k\lambda$
- Find μ: move the hare to the start, the difference is still $k\lambda$, move hare and tortoise together until they meet. This happens after μ steps.
- Find λ: move the tortoise to x_μ, then start moving the hare. After λ steps the values will repeat
- We can use this algorithm to find loops in a linked list
  - Tip: Finding a loop in a list with $O(1)$ memory is a classic job interview question

Factorization

Factorization – Pollard's Rho Algorithm

- The problem: given a number, find its prime factors.
- A simpler problem: given a number n, find p s.t. $n \,\%\, p = 0$.
- Trivial solution: try numbers one by one.
- Complexity: $O(p)$ where p is the smallest prime factor.
- Can we do better?

Factorization – Pollard's Rho Algorithm

- The basic idea: if we can find a pair x, y, s.t. $x \equiv_p y$ we have found a divisor of n.
- Why? $x \equiv_p y \Rightarrow x - y \equiv_p 0 \Rightarrow \gcd(|x-y|, n) = r \cdot p$
- Guessing such a pair is hard.
- But given a set $X = \{x_0, x_1, \dots, x_k\}$, what is the probability that such a pair exists in X?
  - What are the chances that two of you shares the same birthday?
- If $k = \Theta(\sqrt{p})$ the chances are good (details omitted).

Factorization – Pollard's Rho Algorithm

- Problem: checking all pairs in $X = \{x_0, x_1, \dots, x_k\}$ will take $O(k^2)$ time.
- What can we do?
- Consider a sequence which is generated by some function f, i.e.
$$x_0, x_1 = f(x_0), x_2 = f(x_1)$$
- If we will look at this sequence modulo p, after a finite amount of steps it will start to repeat itself, when we find the period we will have pairs of number which are congruent modulo p.
- How do we find it? Using the tortoise and the hare!

Factorization

- <u>Pollard's Rho Algorithm</u>
- Generate a random sequence of numbers.
  - Start with a random x_0, and $f(x)$ = $x^2 + c \ mod \ n$ where c is a random constant.
- Run over this sequence with the tortoise and the hare.
  - Let the values be t and h.
- We can't check if $t \equiv_p h$, since we do not know p, but when it will happen, we will have that $\gcd(|t - h|, n) > 1$.
  - If $\gcd(|t - h|, n) = n$ try again.