THINK FAST

Competitive Programming

From Problem 2 Solution in O(1)

Combinatorial Game Theory

Sprague – Grundy - Examples

Mostafa Saad Ibrahim
PhD Student @ Simon Fraser University

Grundy and Sequence patterns

- If limits are big, although we can code it if input limits are small
  - Code it for small limits and see if there is **pattern**
  - Example: Given k piles (< 50), each pile up to $\mathbf{2^{90}}$ **stones**
    - In 1 move, user can take 2^m (for whatever m)
    - Loser: Can't make a move: Who is winner?
  - Notice, if # stones were little, we can grundy it
  - **Find the pattern**: 0 1 2 0 1 2 0 1 2 …
    - Grundy (n) = n%3
  - What if in 1 move, user can take 1 or prime number?
    - Grundy(n) = n%4

Grundy and Sequence patterns

- Given nim of N piles and moves can remove
  - any **even** number of coins but NOT the whole pile
  - The **whole** pile IFF it has an **odd** number of stones
- Normal grundy, with 2 losing positions
  - N = 0 [Classical case]
  - N = 2 [Case won't fit with the 2 rules]
  - Be careful with rules :)
- Code and compare your output
  - SG(n) = 0 1 0 2 1 3 2 4 3 5 4
  - Can you identify a pattern of this sequence? (2 equations)
    - **Your turn**. Solution in next side

Grundy and Sequence patterns

- Solution
  - Pattern: $SG(2k) = k - 1$ and $SG(2k-1) = k$ for $k \geq 1$

Your turn: Sequence patterns

- Example:
  - Given k piles (< 50), each pile up to 10^5 stones
  - In 1 move, user can take any v <= pile_size / 2
  - Loser: Can’t make a move: Who is winner?
  - Try to find the pattern
  - Hint: One simple rule to even position and one for odd
  - My pattern videos identification should help
  - Solution in next side

Your turn: Sequence patterns

- **Solution**
  - Pattern: 0 1 0 2 1 3 0 4 2 5 1 6 3 7
  - if n is even: grundy(n) = n/2
  - if n is even: grundy(n) = grundy(n/2)

Example: Sequence patterns

- Doubloon Game problem:
  - Given 1 pile of size S coins (10^9) [very big] and K
  - Move: pick number that is k^m (1 <= K <= 100)
  - Loser: Nothing to do
  - Request: If will win, **the smallest number** of coins
  - This time, it doesn't need only win or lose! But also the min number of coins to select!
  - Adjust grundy code to find the smallest coins
    - For every sub-state you can win, minimize over it
  - Print table k * n and identify pattern
  - Observe: For a given k: most of row is 0s and 1s, some times value k. Trivial pattern to find a rule.

Example: Sequence patterns

```
int k;
int steps[100];

int calcGrundy3(int n) {
  if (n == 0)
    return 0;

  int &ret = grundy[n];
  if (ret != -1)
    return ret;

  unordered_set<int> sub_nimbers;
  steps[n] = 1000000;

  for (int v = 1; v <= n; v *= k)
  {
    int sol = calcGrundy3(n-v);
    sub_nimbers.insert(sol);

    if(sol == 0)  // then I win, let's minimize
      steps[n] = min(steps[n], steps[n-v] + v);
  }
  if(steps[n] == 1000000)
    steps[n] = 0; // we will lose

  return ret = calcMex(sub_nimbers);
}
```

Example: Sequence patterns

k= 6: 0 1 0 1 0 1 **6** 0 1 0 1 0 1 **6** 0 1 0 1 0 1 **6** 0 1 0 1 0 # coins table k x n
k= 7: 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0
k= 8: 0 1 0 1 0 1 0 1 **8** 0 1 0 1 0 1 0 1 **8** 0 1 0 1 0 1 0 1 **8** 0
k= 9: 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0
k= 10: 0 1 0 1 0 1 0 1 0 1 10 0 1 0 1 0 1 0 1 0 1 10 0 1 0 1 0 1 0 1
k= 11: 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0
k= 12: 0 1 0 1 0 1 0 1 0 1 0 1 12 0 1 0 1 0 1 0 1 0 1 0 1 12 0 1 0 1 0 1
k= 13: 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0
k= 14: 0 1 0 1 0 1 0 1 0 1 0 1 0 1 14 0 1 0 1 0 1 0 1 0 1 0 1 0 1 14 0 1 0 1
k= 15: 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0
k= 16: 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 16 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 16 0 1
k= 17: 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0
k= 18: 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 **18** 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 **18**

```
int solve(int s, int k) {
  int x = s % (k + 1);
  if(k % 2 == 0 && x == k)
    return k;     // value k repeats in regular way
  return x % 2;   // all other values are 0 and 1
}
```

Your turn: Sequence patterns

- Shuriken Game problem:
  - Given 1 pile of size S coins (10^5) [very big]
  - Move: pick number from 1 to k is (2 <= K <= 100)
  - Restriction: user can’t do the same action as last player (e.g. **No duplication strategy)**
  - Loser: Nothing to do
  - Input: S, K, T
    - Game is actually running, last player removed T)
  - Request: If will win, **the smallest number** of coins
  - E.g. For (6, 6, 6). Last player took 6, so I can take [1-5]
  - Let me take 3. So now S = 3. Other can take [1, 2, 4, 5, 6]
  - Actually, opponent can take only 1, 2 as [4, 5, 6] are > 3

Your turn: Sequence patterns

Almost same code style

Your turn: Find the pattern or whatever to get it AC!

```
int grundy2[120][120];
int steps2[120][120];

int calcGrundy4(int n, int last) {
  if (n == 0)
    return 0;

  int &ret = grundy2[n][last];
  if (ret != -1)
    return ret;

  unordered_set<int> sub_nimbers;
  steps2[n][last] = 1000000;

  for (int v = 1; v <= k && n - v >= 0; v++)
    if (v != last) {
      int sol = calcGrundy4(n - v, v);
      sub_nimbers.insert(sol);

      if (sol == 0)  // then I win, let's minimize
        steps2[n][last] = min(steps2[n][last], steps2[n - v][v] + v);
    }
  if (steps2[n][last] == 1000000)
    steps2[n][last] = 0;  // we will lose

  return ret = calcMex(sub_nimbers);
}
```

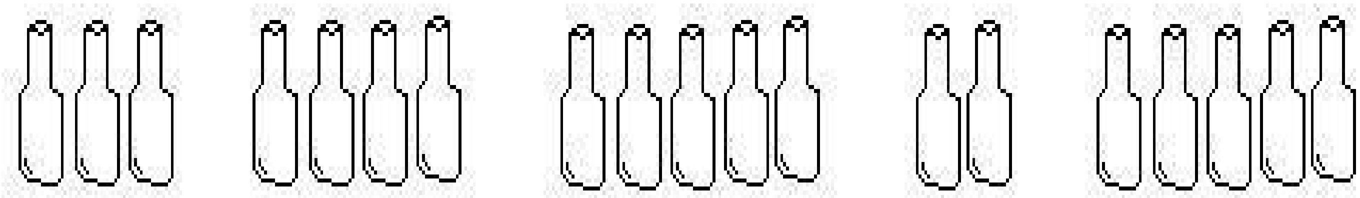
Lasker's Nim

- Generalization of Nim to ***Take-and-Break Game*** where moves are:
  - Remove any number of coins from one pile as in nim
  - Split one pile to 2 non-empty piles (This rule is a single move of **multiple independent sub-games**)
  - For N = 3, we can do:
  - Rule 1: moves to states 0, 1, 2
  - Rule 2: split pile(3) to pile(1), pile(2) => don't try 2, 1
  - So SG(3) = mex(SG(0), SG(1), SG(2), **SG(1)^SG(2)**)
  - SG(n) = 0 1 2 4 3 5 6 8 7 9 10 12 …
  - Find the pattern (4 equations). Solution in next slide

Lasker’s Nim

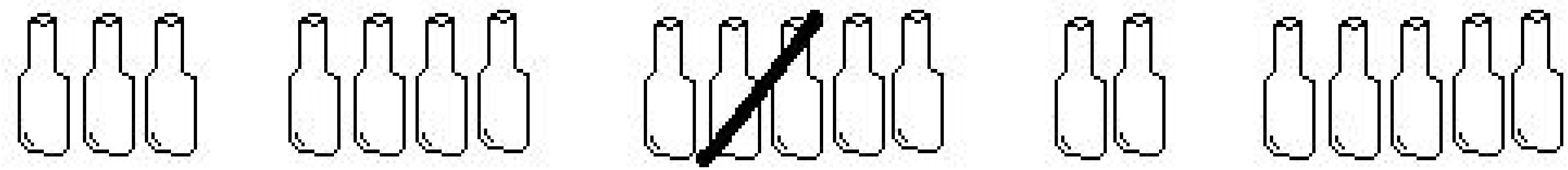
- Solution
  - Pattern: for all $k \geq 0$
  - $g(4k+1)=4k+1$,
  - $g(4k+2)=4k+2$
  - $g(4k+3)=4k+4$
  - $g(4k+4)=4k+3$

Game of Kayles

- Given N set each of contiguous bottles
- Player move: knocks down one or two adjacent bottles
  - When knocking bottles, they may remain 1 group (if from boundaries) or split to 2 sub-groups
  - E.g. **Take-and-Break Game**
- Loser: nothing to knock
- 

Src: https://simomaths.wordpress.com/2012/08/06/combinatorial-game-theory-iv/

Game of Kayles



- Above, 5 groups
- An example of valid move for 3rd group
- Now, we have 6 groups
- Compute Grundy(n)
  - Try all 1 knocking
  - Try all 2 knockings
  - Some will generate 1 group and others 2 groups
  - More than a group: internal xor of the sub-games grudnies

Game of Kayles

- Code and compare with these Grundy(N)

0	1	2	**3**	1	4	**3**	2	1	**4**	2	**6**
4	1	2	**7**	1	4	**3**	2	1	**4**	**6**	7
4	1	2	8	**5**	4	**7**	2	1	8	6	7
4	1	2	**3**	1	4	7	2	1	8	2	7
4	1	2	8	1	4	7	2	1	**4**	2	7
4	1	2	8	1	4	7	2	1	8	**6**	7
4	1	2	8	1	4	7	2	1	8	2	7

- The pattern will be in **precycle-cycle format**
  - First 71 values are pre cycle
  - From n = 72, the SG-values are periodic with period 12

Dawson’s Kayles

- Same rules as Kayles, except that every player must remove exactly two contiguous bottles
- Your turn: Find the pattern
  - Solution in next slide

Dawson’s Kayles

- Solution
  - Pattern:
  - From n = 52, the SG-values are periodic with period 34

Your turn: Coins in the DAG

- Given DAG, with some nodes has a coin
  - Move: move a coin through directed edge
  - Loser: Can’t move
  - Actually DAG define grundy recursive calls (edge=move)
- Version 2
  - Assume if 2 coins are now in same nodes they **disappear**
  - This information is **useless** (e.g. equal to above game)
  - Because when we have 2 equal nim sizes, xor cancel them
    - E.g. if result is 15, then 15 xor 15 = 0
  - Equal values in same node **by game nature** are cancelled
  - Don’t be tricked by fake dependency rules/moves

Recall: Turning Turtles Game

- Given a horizontal line of N coins: Head/Tail
  - Move: Pick any head, and flip it to tail
  - Optionally, flip any coin on left of your chosen coin
- Solution
  - For every head at position k => pile of size k
  - Depdendent sub-games:
    - When the optional flipped coin goes from T to H, kind of dependency (e.h. TTHTTH**H** => **H**THTTHT)
    - We proved they are actually independent
    - So HTTHH = HTTTT + TTTHT + TTTTH
    - That is, every H is independent sub-game

Your turn: Mock Turtles

- Variation of Turning Turtles Game
  - Now optionally turn up to 2 coins on your left
  - Assume N = 10^9 (but 10^5 heads maximum)
  - It is complex game now to prove nim equivalence
  - Better way, try to compute grundy value for the game
  - Very big N? Try small N and find a **pattern**

تم بحمد الله

علمكم الله ما ينفعكم

ونفعكم بما تعلمتم

وزادكم علماً