Games

Task

In game theory, a game is a process in which two or more players participate, everybody wants to win and makes best possible turns. Each party has its own goal and uses a certain strategy, which can lead to either winning or losing.

Nim

Let's consider the following game. There are n piles, each containing some number of stones. In one move, a player can choose a pile and remove any non-zero number of stones from it. The player who takes the last stone wins.

This game is called Nim and it is one of the fundamental problems in game theory.

Simple version

Let there be only one pile with X stones. You can only take one or two stones. The player who takes the last stone wins.

How to solve such a task?

Simple version

dp[i] - the answer for a pile of i stones, true if the position is winning and false otherwise

dp[0] = false, because the player whose turn it is when there are 0 stones has lost

dp[i] = true if (dp[i - 1] = false or dp[i - 2] = false)

i = 0 .. X

dp[X] - answer

Example

dp = [0, 1, 1, 0, 1, 1, 0, 1, 1, 0]

let’s check 7 for example, we can remove 1 stone.

6 stones = Lose, because if you take 1 stone, opponent takes 2 and vice versa

3 stones = Lose, because if you take 1 stone, opponent takes 2 and vice versa

0 stones = Lose

Interesting solution

```
if amount of stones % 3 == 0:
        Lose
else
        Win
```

Bigger task

Very often, a game can be solved using dynamic programming (dp) or an interesting fact.

If we are allowed to take not just 1 or 2 stones but any number of stones in the range [1, y], then the solution using dp is almost the same, and with dynamic programming, we need to check not the remainder of division by 3, but by y + 1.

Example

If we can take [1, 6) stones.

So $n = 0, 6, 12, 18 \dots$ - losing

and everything else is win.

Nim

Let's now solve a simple Nim with n piles [X_0, …,X_(n - 1)].

We could use n-dimensional dynamic programming -

dp[am_0, am_1, .., am_(n - 1)], but there is a better solution.

Nim

To prove the simple solution for the last game, we used reasoning about the symmetry of moves (no matter how we move, or how many stones we take, our opponent will take 3 - x). In Nim with n piles, the logic is the same.

Solution

If (x_0 xor x_1 xor … xor $x_{(n - 1)}$) = 0, then the position is losing, otherwise, it is winning.

We will prove this by induction.

The position [0,0,…,0] is a losing one.

Xor

0 xor 0 = 0

1 xor 0 = 1

0 xor 1 = 1

1 xor 1 = 0

15 xor 31 = 16

01111

11111

1000

Example

[1, 2, 3] -> [1, 2, 0] -> [1, 1, 0] -> [0, 1, 0] -> [0, 0, 0]

-> [1, 0, 3] -> [1, 0, 1]

1 xor 2 xor 3 = 0, it’s lose

[1, 3, 3] -> [0, 3, 3] -> [0, 3, 1] -> [0, 1, 1] -> [0, 0, 1] - > [0, 0, 0]

1 xor 3 xor 3 = 1, it’s win

Solution

Let's show that if:

a) (x_0 xor … xor x_(n - 1)) = 0, then we only have transitions to positions [y_0, … , y_(n - 1)], such that (y_0 xor … xor y_(n - 1)) != 0

b) (x_0 xor … xor x_(n - 1)) != 0, we have a transition to a position [y_0, … , y_(n - 1)], such that (y_0 xor … xor y_(n - 1)) = 0

Examples

1) (3 xor 3) = 0, it's easy to demonstrate that any move leads to a non-zero sum, for example, (2 xor 3) = 1 or (3 xor 1) = 2
2) (3 xor 5 xor 4) = 2 != 0, take the number 3, replace it with 3^2 = 1, (1 xor 5 xor 4) = 0

Solution

Suppose before the move, the position was (x_0 xor … xor $x_{(n-1)}$) = 0.

After the move, we got the position (y_0 xor … xor $y_{(n-1)}$).

We need to prove that (y_0 xor … xor $y_{(n-1)}$) is always != 0.

Solution

But then consider such a possible move, during which exactly one element changed - let's call it i. Then we get (x_0 xor … xor x_(n - 1) xor x_i xor y_i), but (x_i xor y_i) != 0, as the element x_i decreased. Therefore, we should have obtained (x_0 xor … xor x_(n - 1)) xor ((x_i xor y_i)) = 0 xor !0, thus proving it.

Solution

"Then we get (x_0 xor … xor $x_{(n-1)}$ xor x_i xor y_i)"

you had ((x_0 xor … xor $x_{(n-1)}$))

you now that i-th position $x_i \to y_i$

now you have ((x_0 xor … xor $x_{(n-1)}$ xor x_i xor y_i)

Solution

Suppose before the move, the position was $(x_0 \text{ xor } \ldots \text{ xor } x_{(n-1)}) \neq 0$.

After the move, we got the position $(y_0 \text{ xor } \ldots \text{ xor } y_{(n-1)})$.

We need to find such a move that $(y_0 \text{ xor } \ldots \text{ xor } y_{(n-1)}) = 0$.

Solution

We need the xor-sum to change by s = (x_0 xor … xor x_(n - 1)).

Let the highest one bit of s be k, i.e., s = 2^k + s#, where s# < 2^k.

Find such an x that has the bit k, i.e., (x and 2^k != 0).

Replace x with x xor s. This number is definitely less than x, hence the move is correct.

Check of proof

a) (3 xor 3) = 0, it's easy to show that any move leads to a non-zero sum, for example, (2 xor 3) = 1 or (3 xor 1) = 2

b) (3 xor 5 xor 4) = 2 != 0, find any number that contains the first bit, that is 2. This number is 3, replace it with 3^2 = 1, (1 xor 5 xor 4) = 0

Let's play

[1, 3, 3, 2], 1 xor 3 xor 3 xor 2 = 3 - my turn

[1, 3, 0, 2], 1 xor 3 xor 2 = 0 - your turn

[1, 2, 0, 2], 1 xor 2 xor 2 = 1 - my turn

[0, 2, 0, 2] 2 xor 2 = 0 - your turn,

[0, 1, 0, 2] 1 xor 2 = 3 - my turn, 2 xor 3 = 1

[0, 1, 0, 1] 1 xor 1 = 0 - your turn,

[0, 1, 0, 0] -> [0, 0, 0, 0]

Bigger task

It turns out that many games can be reduced to Nim. Let's first try to solve the following problem - now we can not only remove stones but also add them.

It doesn’t matter what the rules for increasing are, it’s only important that the game remains acyclic.

Example

For example, it might be prohibited for one player to add stones two turns in a row.

Or each pile can be increased only once.

Example

[1, 3, 2] - 1 xor 3 xor 2 = 0, so it’s losable

[1, 3, 100] - 1 xor 3 xor 100 = 102

[1, 3, 2] - 1 xor 3 xor 2 = 0

Solution

Notice that if we are in a losing position and we even increase some pile, then our opponent can reduce it by the same amount (since there are no restrictions on reduction).

Graph games

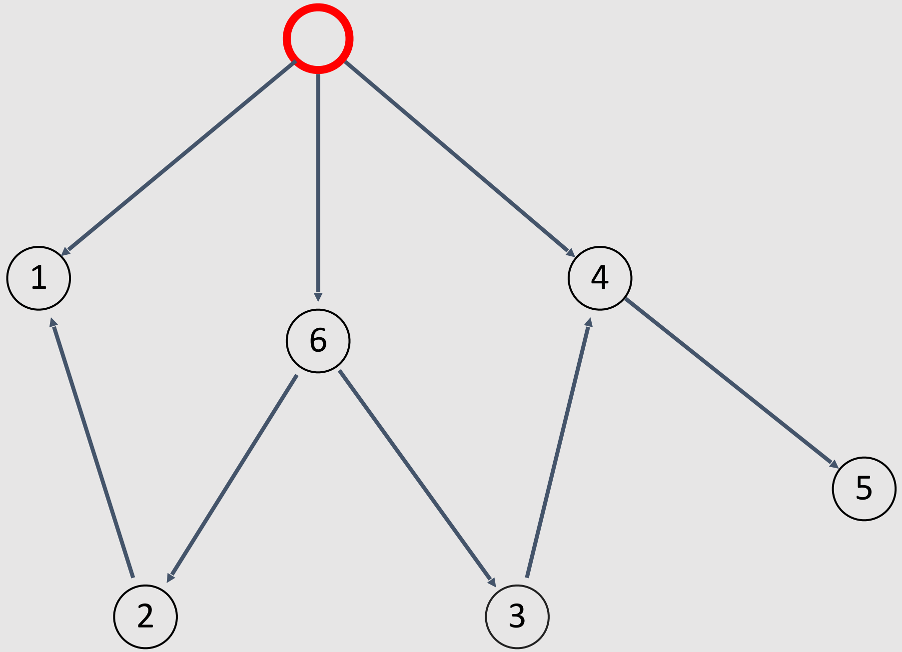
Let the game be played by two players on some graph G.

That is, the current state of the game is a certain vertex of the graph, and from each vertex, edges go to those vertices where one can move next.

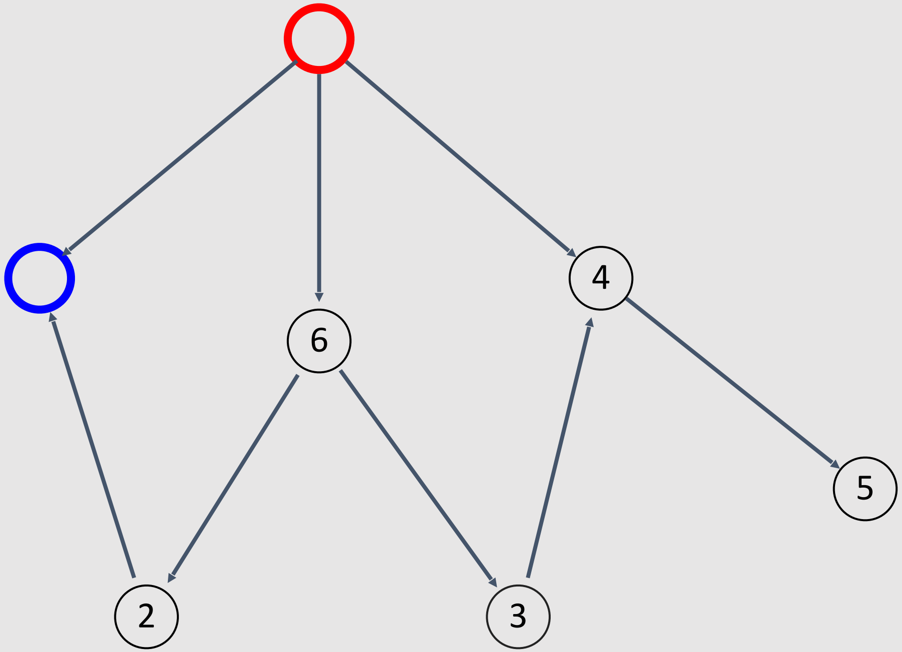
Graph games

We are considering the most general case - the case of an arbitrary directed graph with cycles. The task is to determine, for a given starting position, who will win with optimal play from both players (or determine that the result will be a draw).
It doesn't matter who wins - the one who makes the last move or the one who can't make a move, this only changes the code, but not the idea, so let's solve for the first option.

red = start



0 is winning, 1 is losing



Graph games

In fact, the logic here is somewhat similar to Nim. Let's divide the states into three types:

1) Win - there is some edge to a losing vertex.
2) Loss - there are edges only to winning vertices.
3) Draw - there is no edge to a losing vertex, but we can play for unlimited time.

Graph games

Let's first assume that there are only winning and losing vertices.

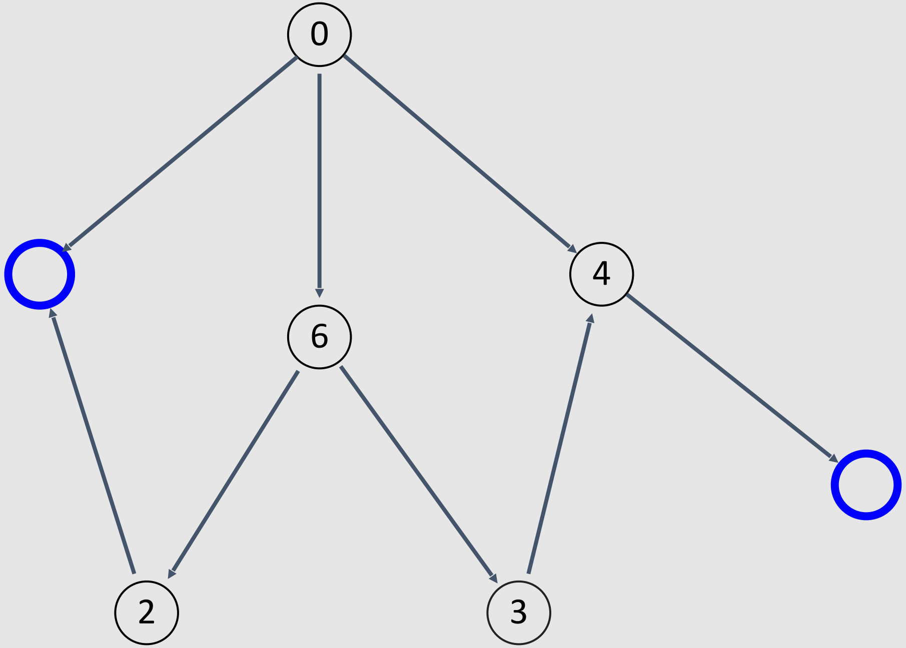
Then, we'll create the following algorithm, using a DFS, which:

1) For vertices adjacent to a losing vertex, will immediately mark them as winning.
2) For each vertex, will maintain the count of unmarked vertices, and as soon as it becomes 0, will mark the vertex as losing.

Graph games

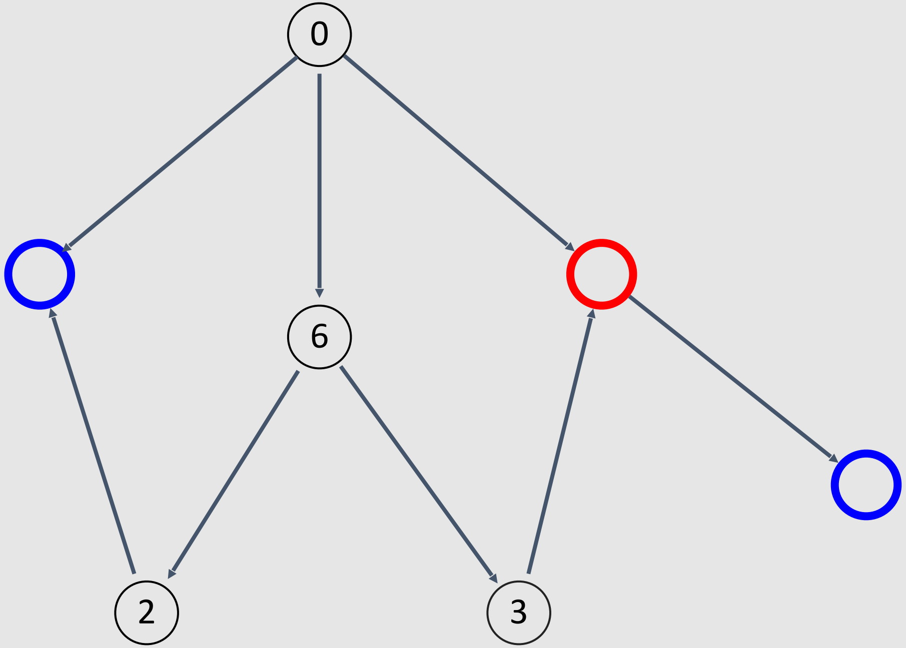
Initially, we will start from vertices with a degree of 0, as they are certainly losing.

red is winning, blue is losing



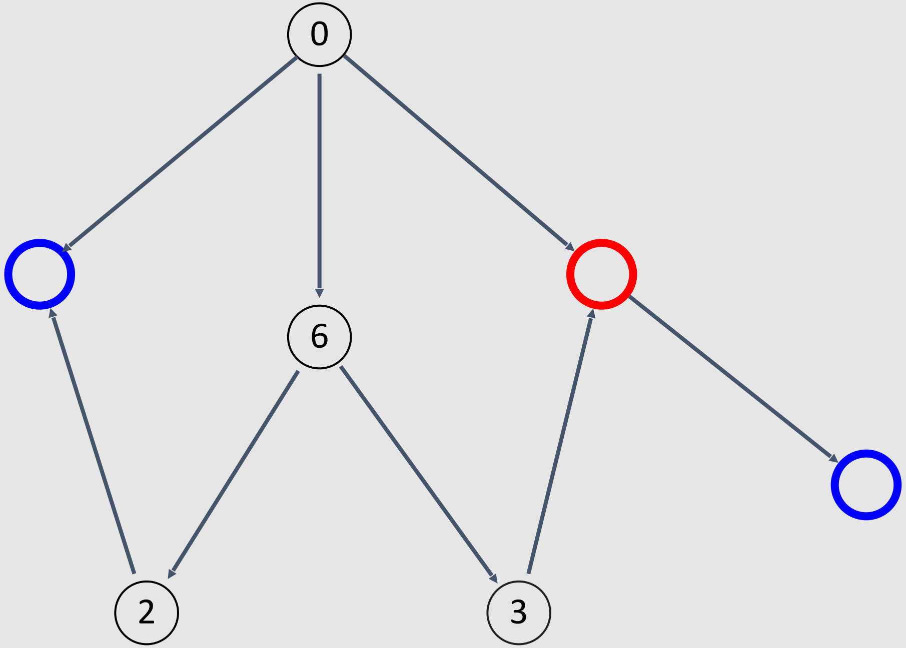
red is winning, blue is losing

start dfs from 5



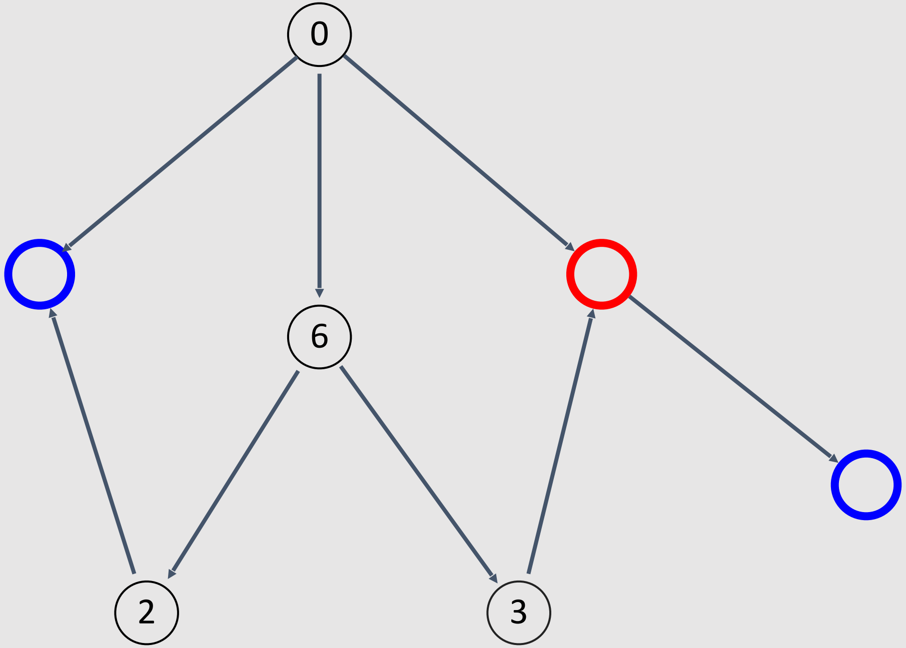
red is winning, blue is losing

start dfs from 4



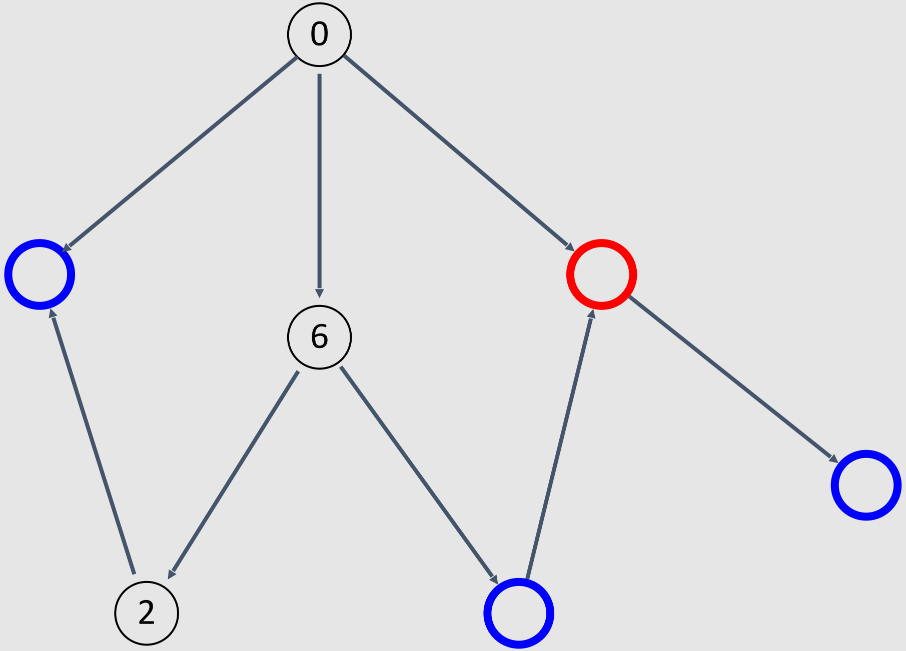
red is winning, blue is losing

check 0, still not finished



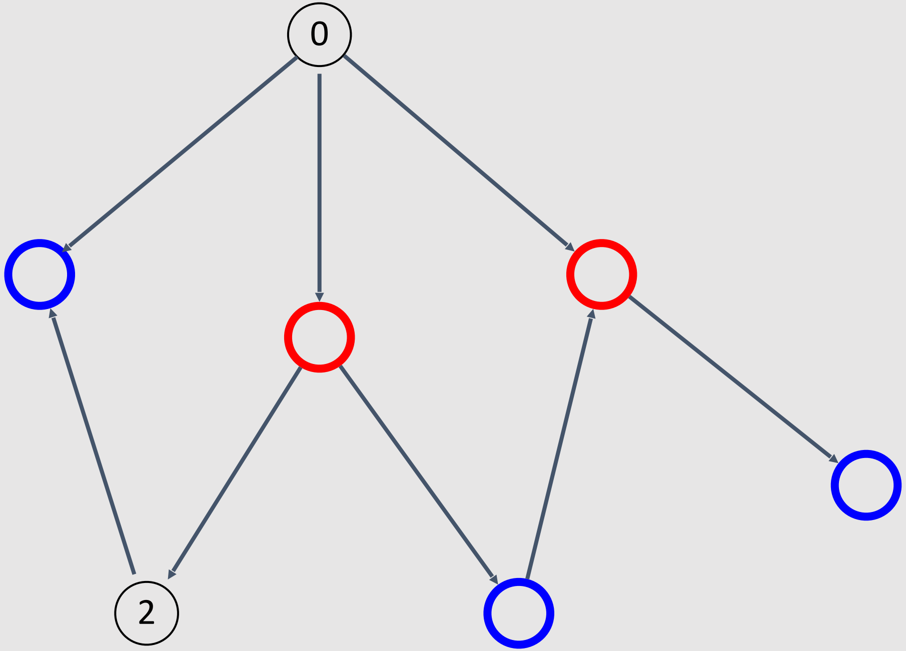
red is winning, blue is losing

check 3



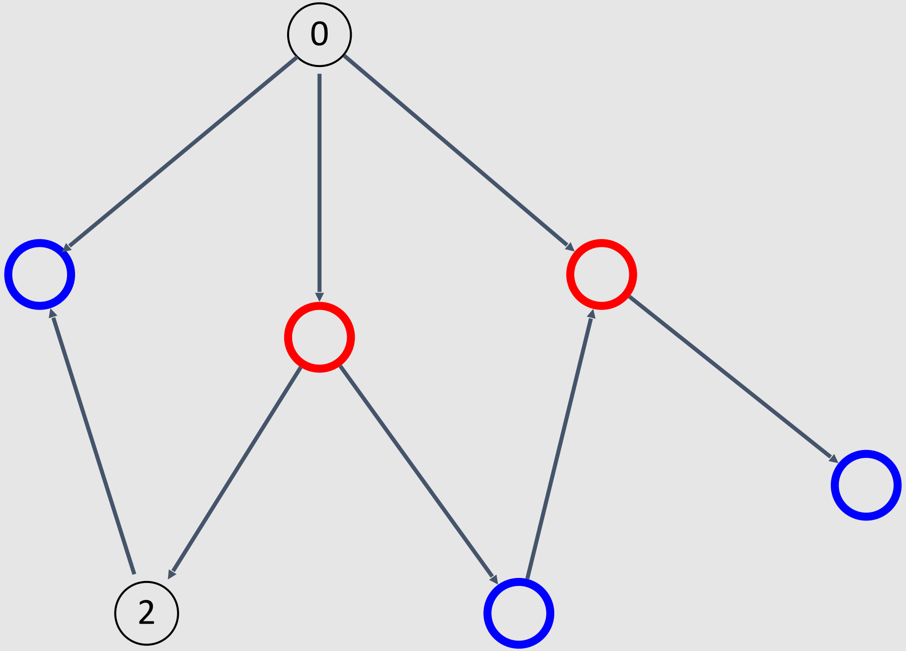
red is winning, blue is losing

check 6



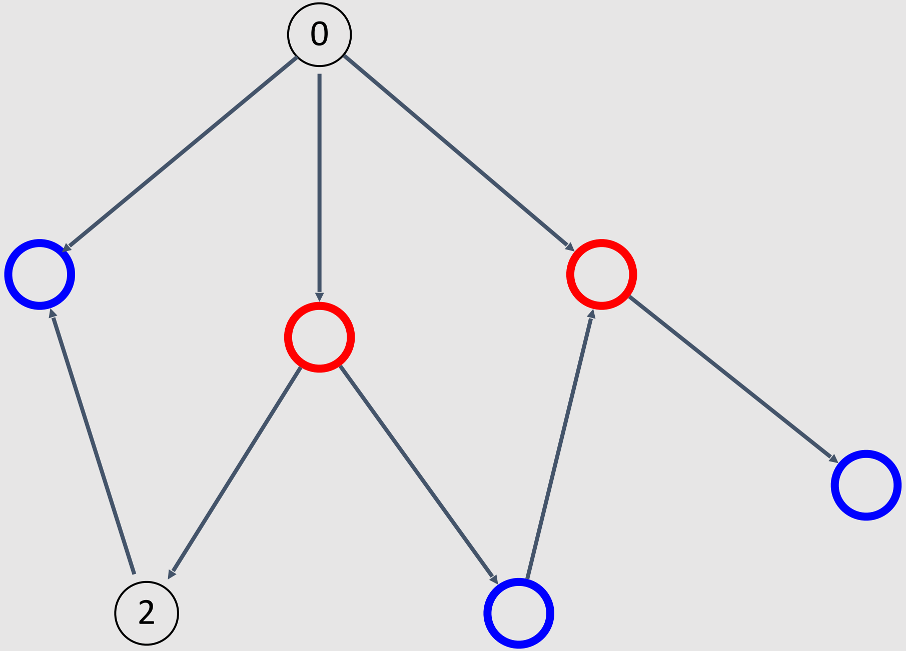
red is winning, blue is losing

check 0, still not finished



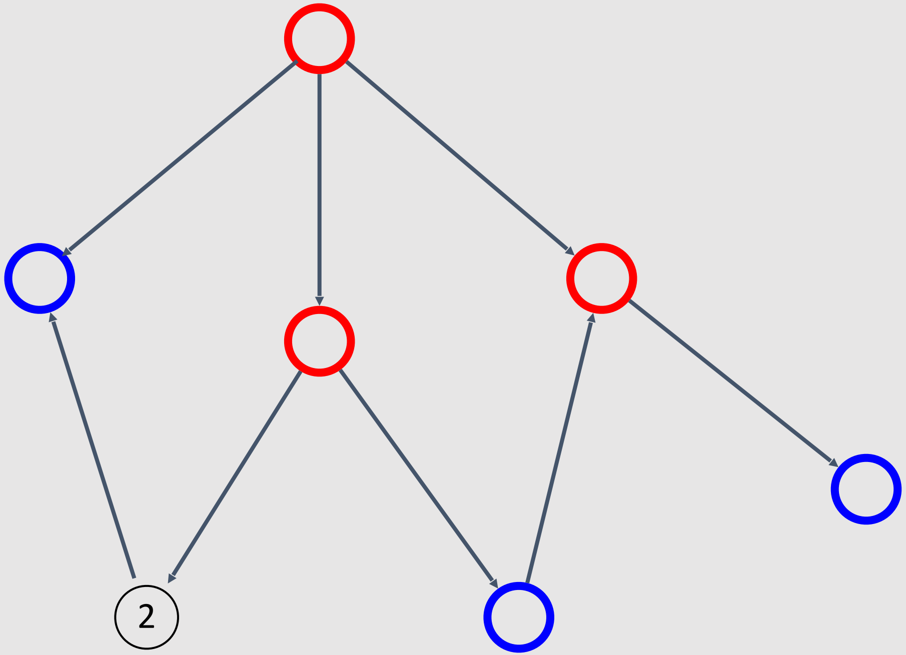
red is winning, blue is losing

start dfs from 1.



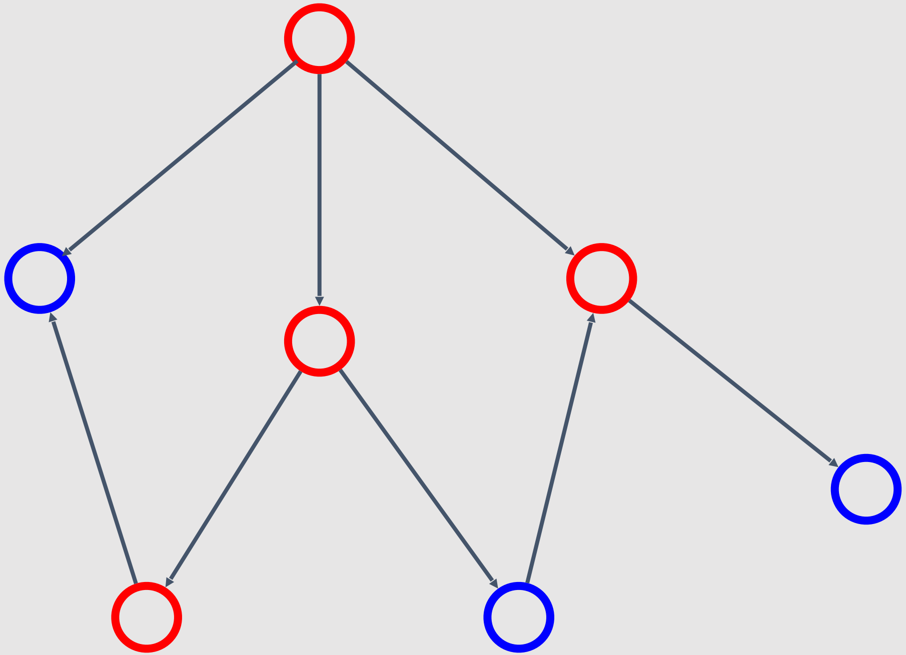
red is winning, blue is losing

check 0



red is winning, blue is losing

check 2

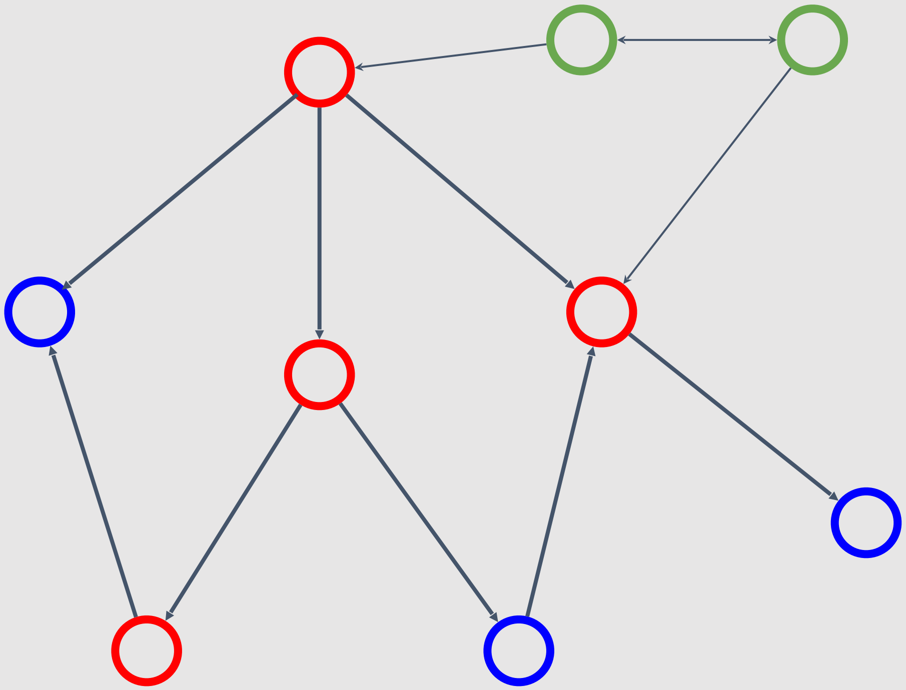


Graph games

But what should we do with the draw vertices?

If some vertex remains unmarked at the end of the algorithm, then it is a draw vertex, as this means that we can infinitely move around the cycle and never find an exit.

red is winning, blue is losing, green is draw



```cpp
enum VertexResult { win, lose, draw };

void solve(int v, vector<vector<int> > &g, vector<VertexResult> &state, vector<int> &degree, vector<bool> &visited) {
    visited[v] = true;
    for (auto i: g[v]) {
        if (!visited[i]) {
            degree[i]--;
            if (state[v] == VertexResult::lose) {
                state[i] = VertexResult::win;
            }
            else if (!degree[i]) {
                state[i] = VertexResult::lose;
            }
            else {
                continue;
            }
            solve(i, g, state, degree, visited);
        }
    }
}
```

```
int main() {
    int n, m;
    cin >> n >> m;
    vector<vector<int> > g(n);
    vector<VertexResult> state(n);
    vector<int> degree(n);
    vector<bool> visited(n);
    for (int i = 0; i < m; i++) {
        int a, b;
        cin >> a >> b;
        g[b].push_back(a);
        degree[a]++;
    }
    for (int i = 0; i < n; i++) {
        if (!degree[i] && !visited[i]) {
            state[i] = VertexResult::lose;
            solve(i, g, state, degree, visited);
        }
    }
    for (int i = 0; i < n; i++) {
        if (!visited[i]) {
            state[i] = VertexResult::draw;
        }
        switch(state[i])
        {
            case VertexResult::win  : std::cout << "win "; break;
            case VertexResult::lose : std::cout << "lose "; break;
            case VertexResult::draw : std::cout << "draw "; break;
        }
        cout << "\n";
    }
    return 0;
}
```

stdin

```
9 12
0 1
2 1
6 2
6 3
3 4
4 5
0 4
0 6
7 0
8 4
7 8
8 7
```

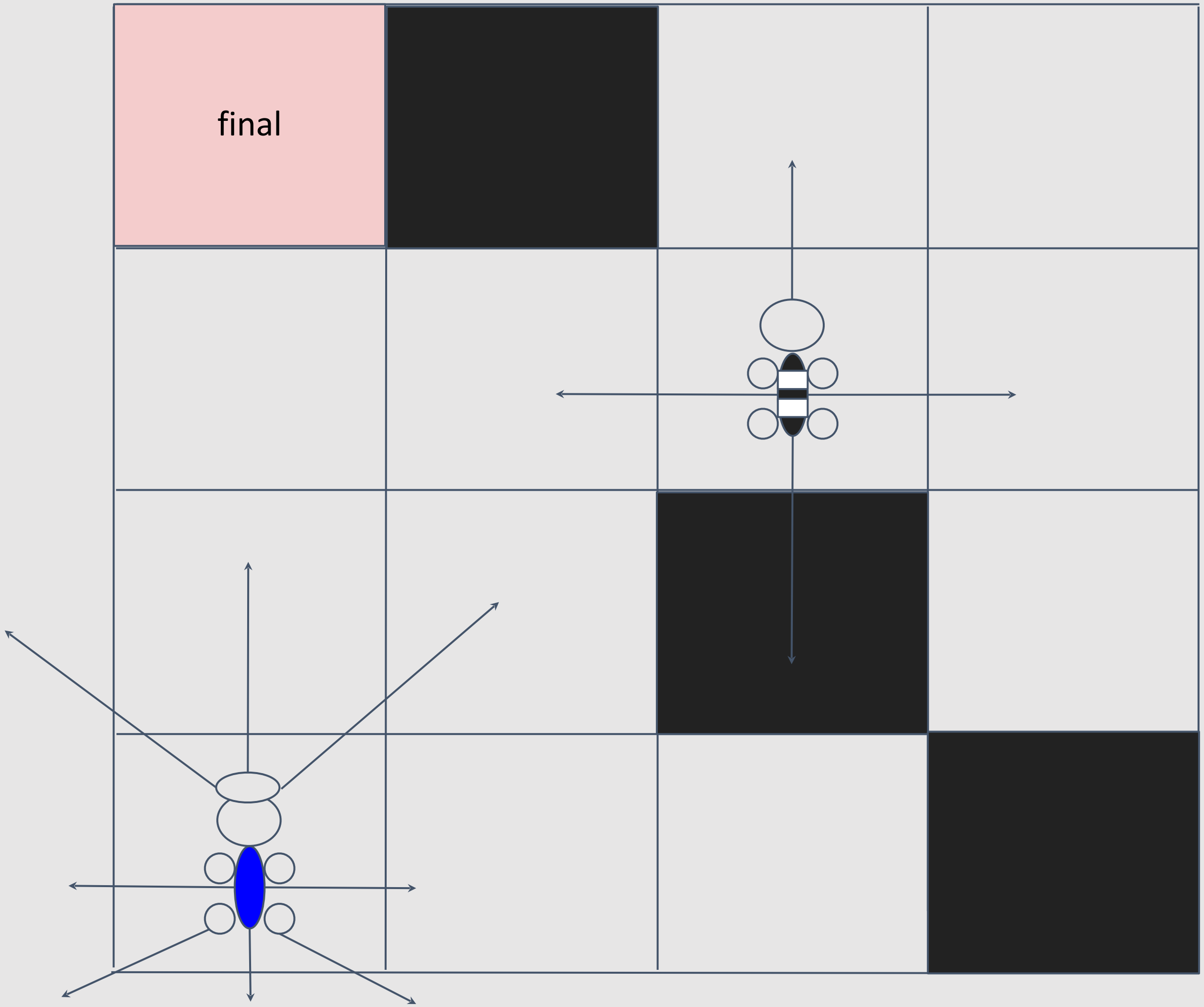
stdout

```
win
lose
win
lose
win
lose
win
draw
draw
```

Game example 1

There is a field of size MxN cells, and some cells are inaccessible. The initial coordinates of the policeman and the thief are known. There is also an exit on the map. If the policeman ends up in the same cell as the thief, then the policeman wins.

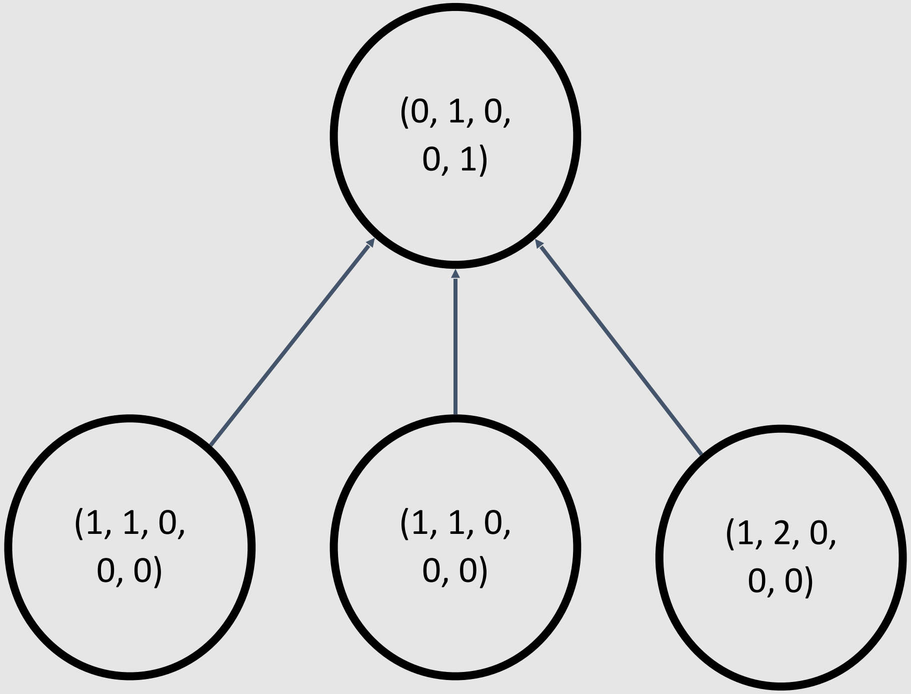
If the thief reaches the cell with the exit (and the policeman is not in this cell), then the thief wins. The policeman can move in 8 directions, the thief only in 4 (along the coordinate axes). The policeman moves first.


final

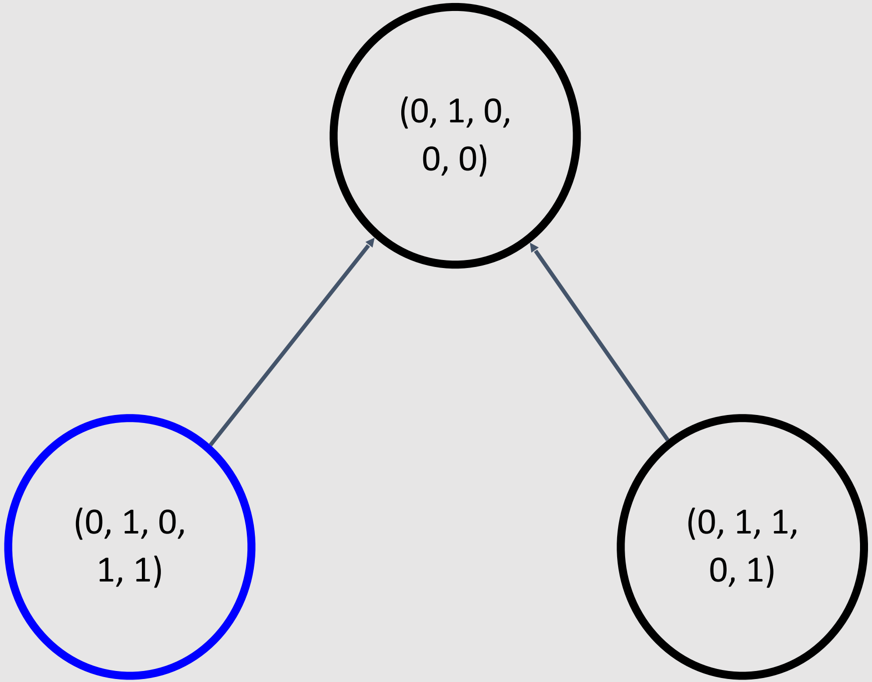
Game example 1

To solve this problem, let's create a graph where a vertex is (row_thief, col_thief, row_police, col_police, player), meaning the cell of the field for the thief, for the policeman, and whose turn it is. An edge (u, v) is drawn between cells if after the state corresponding to vertex u, one can reach the state corresponding to vertex v.

(0, 1, 0, 0, 1) - now is the bandit turn, bandit in (0, 0), police in (0, 1)



(0, 1, 0, 0, 0) - now is the police turn, bandit in (0, 0), police in (0, 1)



Game example 1

A winning position is (ANY_X, ANY_Y, finish_x, finish_y, 1), with (ANY_X, ANY_Y) != (finish_x, finish_y), because this means the thief has left the field while the policeman is not in that cell.

Losing positions are (ANY_X, ANY_Y, ANY_X, ANY_Y, ANY), because this means the thief and the policeman are on the same cell.

Sprague–Grundy theorem

Suppose we now have a more complex game where piles can be split into several parts, rather than just removing stones from them.

Sprague–Grundy theorem

[1, 4, 4] -> [1, 2, 2, 4]

Sprague–Grundy theorem

Here is a simple example of such a game:

There are n piles of stones, their sizes are [a_i]. In one move, a player can take any pile of at least size 3 and split it into two non-empty piles of unequal sizes. The player who cannot make a move (i.e., when the sizes of all remaining piles are less than or equal to two) loses.

Example

[2] - lose, no turns

[3] - win -> [1, 1]

[4] - win -> [2, 2]

Sprague–Grundy theorem

Consider any state v of a certain two-player game. Suppose there are transitions from it to some states v_i. It is asserted that the state v of this game can be corresponded to a pile of Nim of some size x (which will fully describe the state v of our game - i.e., these two states of two different games will be equivalent). This number x is called the Sprague-Grundy value of the state v.

Example

[1, 2, 1] - nim size x

[1, 3, 1] - nim size y

Sprague–Grundy theorem

Moreover, this number x can be found in the following recursive way:

Calculate the Sprague-Grundy value x_i for each transition (v, v_i), and then we can calculate

x as $\mathrm{mex}(x_1, \ldots, x_k)$, where the mex function for a set of numbers returns the smallest non-negative number not present in that set.

Informally

we will take state v, we will calculate answers for all (v, vi) and

take mex(x_1, … , x_k), where x_i answer for v_i

Sprague–Grundy theorem

mex() - least natural number that is not in set

mex(1, 2, 3) = 0

mex(0, 1) = 2

Sprague–Grundy theorem

Thus, starting from states without outgoing edges, we can calculate the Sprague-Grundy values for all states of our game. If the Sprague-Grundy value of any state equals zero, then that state is losing; otherwise, it is winning.

Sprague–Grundy theorem

Let's prove this by induction.

For states from which there are no edges, the value of x, according to the theorem, will be obtained as mex from an empty set, i.e., $x = 0$.

Indeed, a state without edges is a losing state, and it really should correspond to a Nim pile of size 0.

Sprague–Grundy theorem

Now let's consider any state v from which there are edges. By induction, we can assume that for all states v_i, into which we can move from the current state, the values x_i are already calculated.
Calculate the value $p = \text{mex}(x_1, \ldots , x_n)$. Then, according to the definition of the mex function, we know that for any number i in the range $[0; p)$, there will be at least one corresponding edge to some of the v_i states. Moreover, there can also be additional edges - to states with Sprague-Grundy values greater than p.

What is nim?

Nim with one pile - it’s a game in which you can take any amount of stones

Nim [x] -> [x- 1] or [x - 2] or … or 2 or 1 or 0

state v -> state v_0 .. state v_i .. state v_n - 1

[x - 1] -> state_i_(x-1)

[x - 2] -> state_i_(x - 2)

Example

state v

state vi can give you SGs [1, 0, 1, 1, 5]

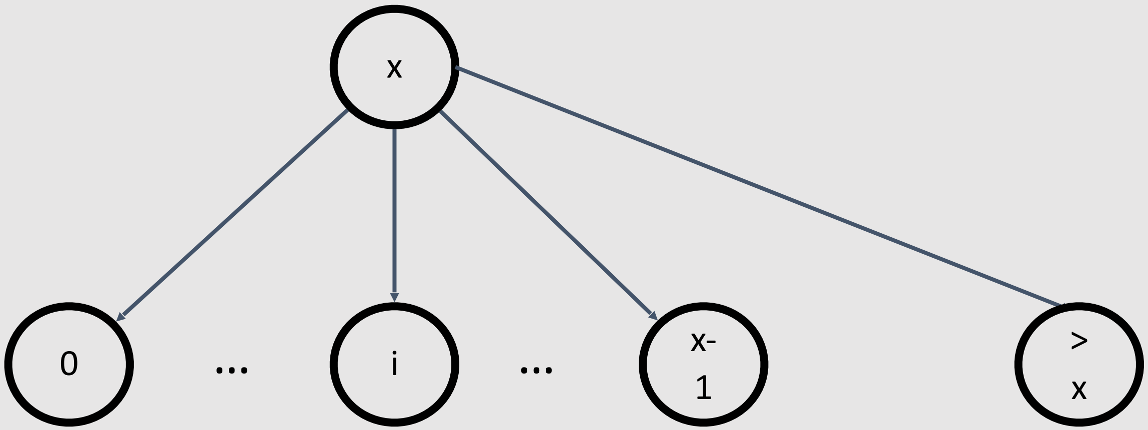
SG for state v = mex([1, 0, 1, 1, 5]) = 2

It's the same as nim with increasing for size 2

2->1 (It's like turn from v to v_0 or v_2 or v_3)

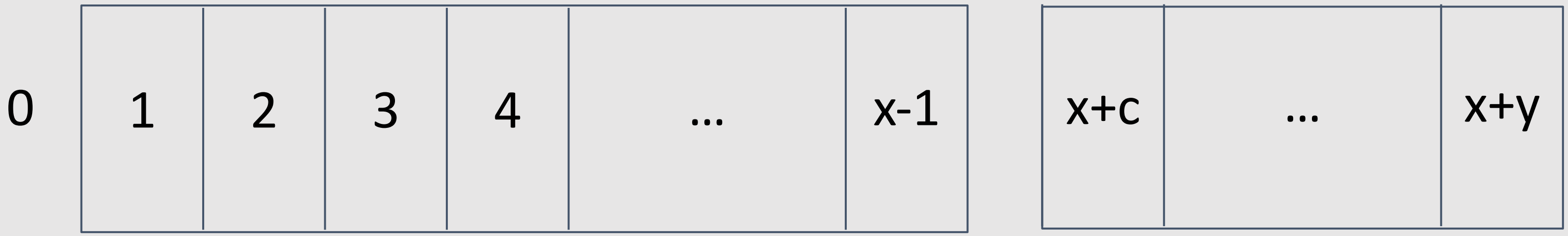
2->0 (It's like turn from v to v_1)

2->5(It's like turn from v to v_4)



x
0
…
i
…
x-
1
>
x

pile with x stones



Sprague–Grundy theorem

This means that the current state is equivalent to a state of Nim with increases for a pile of size p: we have transitions from the current state to states with piles of all smaller sizes, and there may also be transitions to states of larger sizes.

Therefore, the value of mex(x_1, …, x_k) really is the Sprague-Grundy value for the current state, which is what we needed to prove.

Game 1

Consider a 1 * n cell stripe. In one move, a player must place one cross, but it is forbidden to place two crosses next to each other (in adjacent cells). The player who cannot make a move loses. Determine who will win with optimal play.

			X			

	X		X			

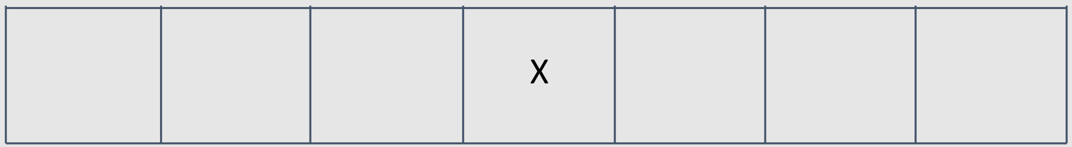
	X		X		X	

Game 1

Notice that if we place a cross on a stripe of n cells in position i, the game splits into two stripes of $i - 2$ elements and $n - i - 1$ elements.

These two stripes can be represented as Nim $(i - 2, n - i - 1)$, as the games are independent.

i = 4



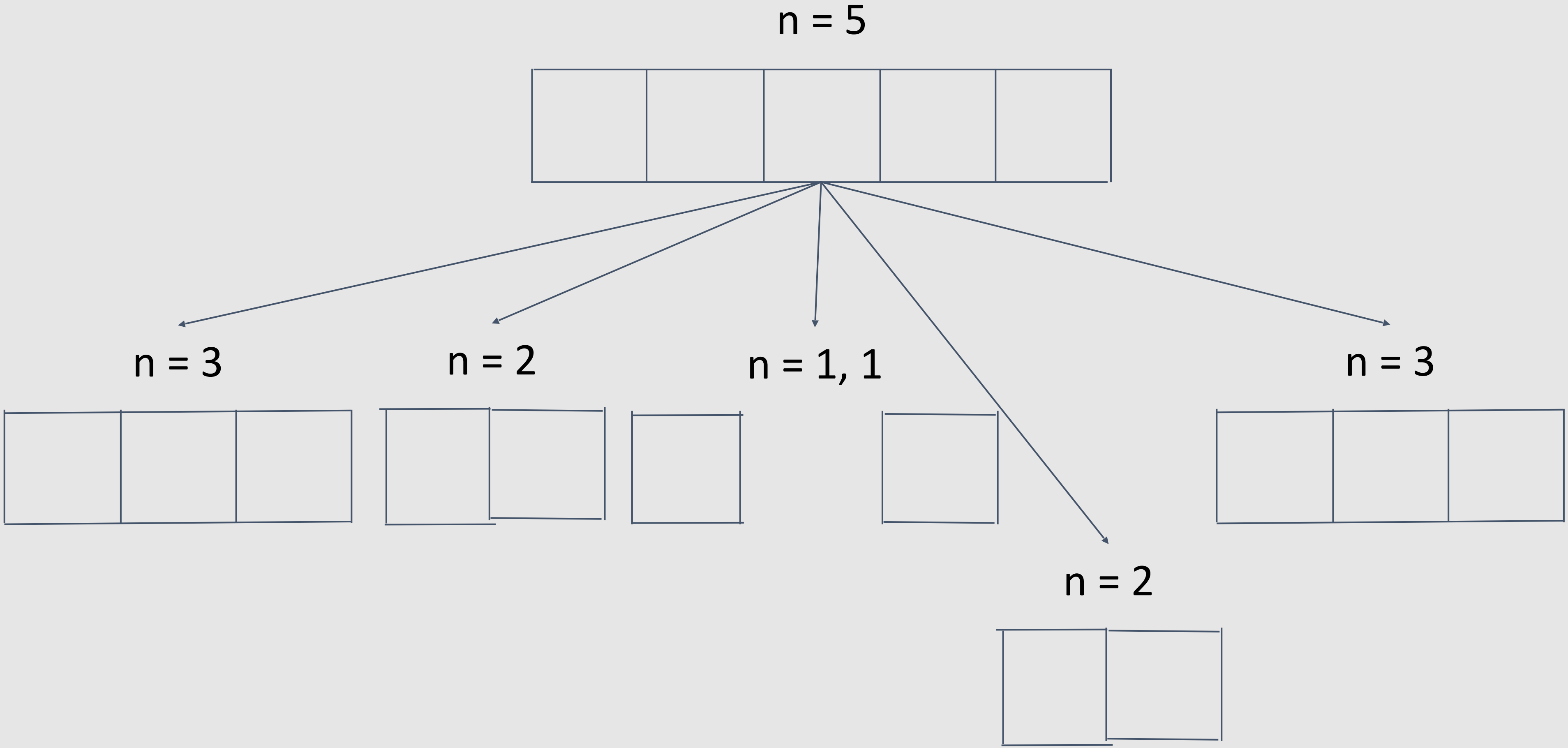
$i - 2 = 2$

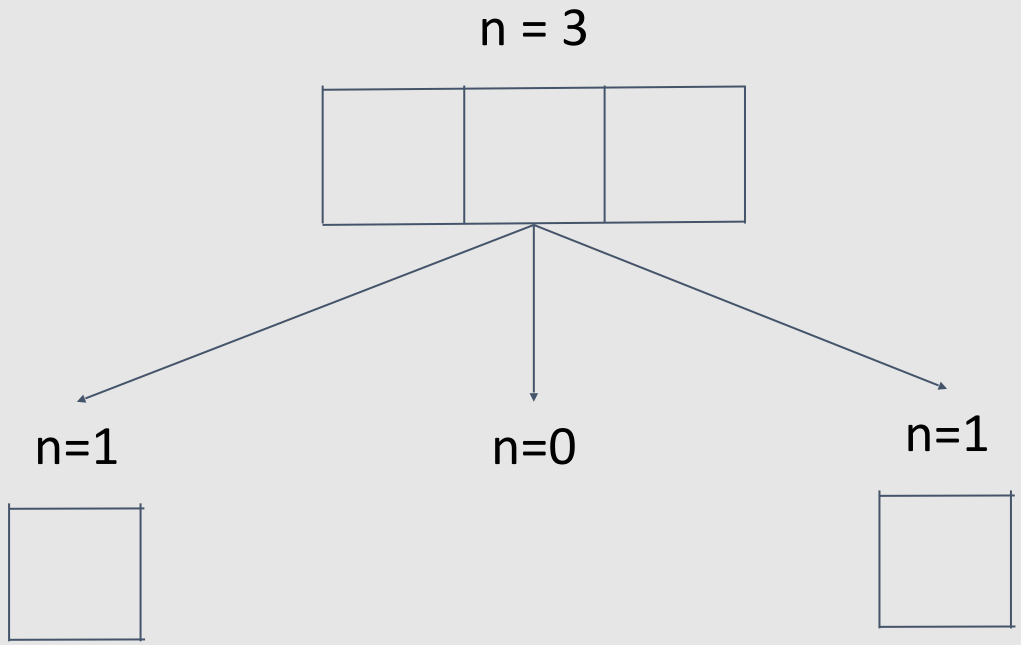
$n - i - 1 = 2$

Game 1

The formula turns out to be as follows:

g(n) = mex(g[0] xor g[n - 2], …, g[i - 2] xor g[n - i - 1], …, g[n - 2] xor g[0])


n = 5
n = 3
n = 2
n = 1, 1
n = 3
n = 2

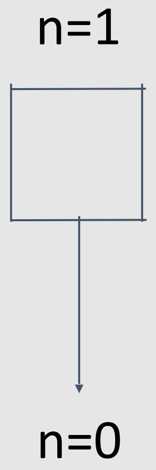


n = 3
n=1
n=0
n=1

answer = 0

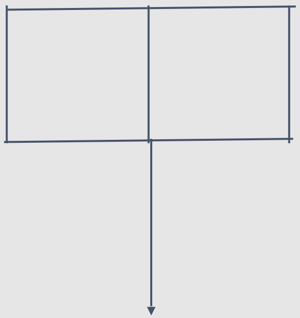
n=0

answer = mex(0) = 1



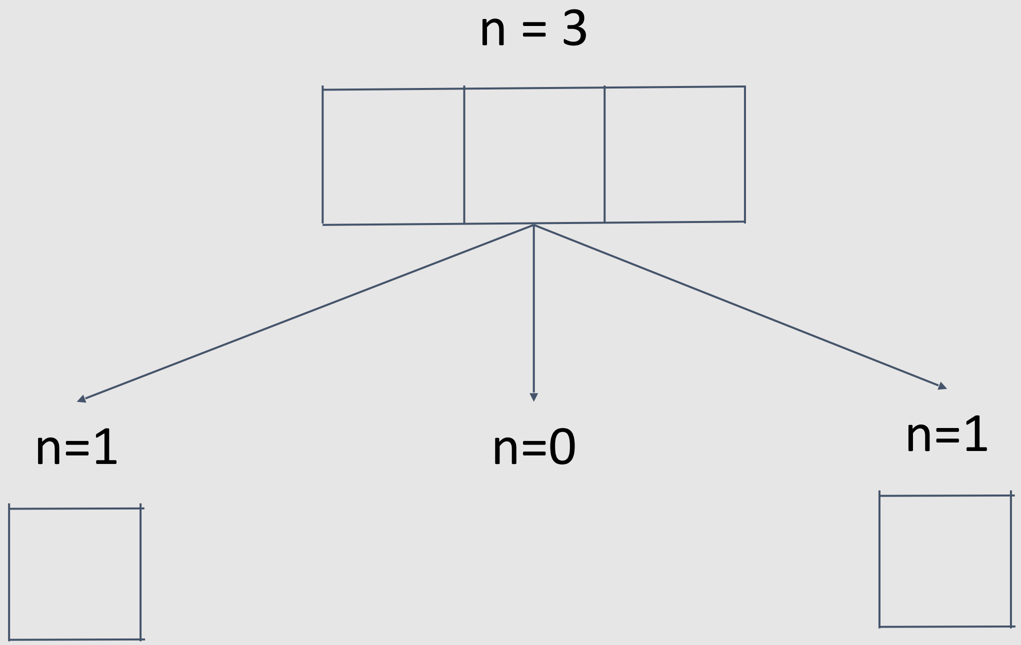
answer = mex(0) = 1

n = 2

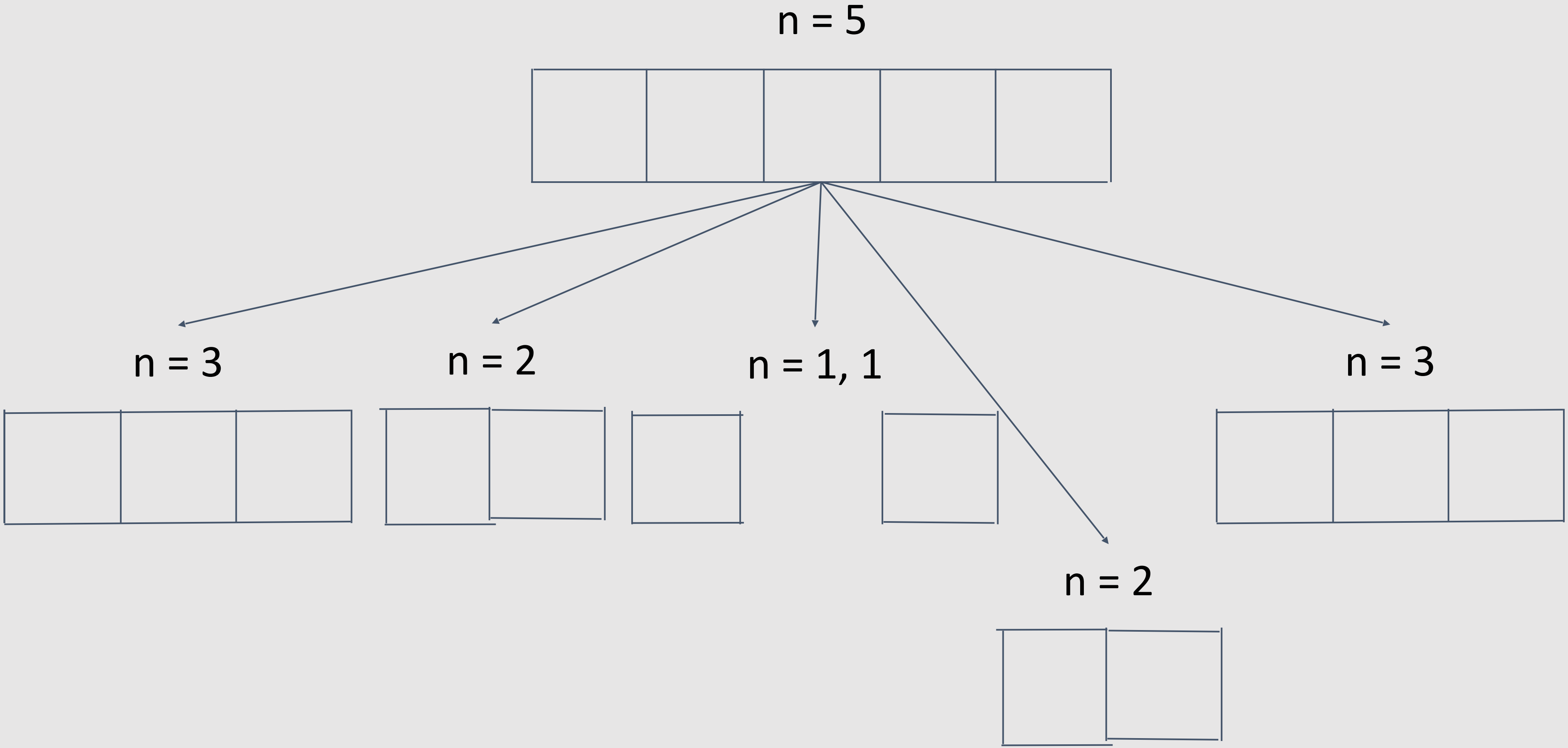


n=0

answer = mex(1, 0, 1) = 2



answer = mex(2, 1, 1 xor 1, 1, 2) = mex(2, 1, 0, 1, 2) = 3



Game 1

In fact, this way we also obtain the strategy - choose a transition to 0, then the opponent's move, again a transition to 0, and so on indefinitely.

For example, here we place a cross in the middle and get (1, 1), whichever pile our opponent removes, we remove the other one.

Example

state = {4}

state_sons = {2, 1, 1, 2}

sol(2) = 1

sol(1) = 1

sg(4) = mex(1, 1, 1, 1) = 0 -> 4 is lose

Example

state = {7}

state_sons = {5, 4, {1, 3}, {2, 2}, {3, 1}, 4, 5}

sol(3) = 2

sol(2) = 1

sol(1) = 1

sol(5) = 3

sol(4) = 0

Example

state = {7}

state_sons = {5, 4, {1, 3}, {2, 2}, {3, 1}, 4, 5}

sg(7) = mex(3, 0, (1 xor 2), 0, (1 xor 2), 0, 3) = 1

Game 2

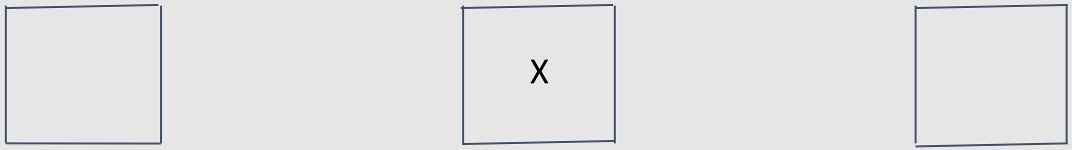
Again, the game is played on a 1 * n cell stripe, and players take turns placing one cross each. The player who places three crosses in a row wins.

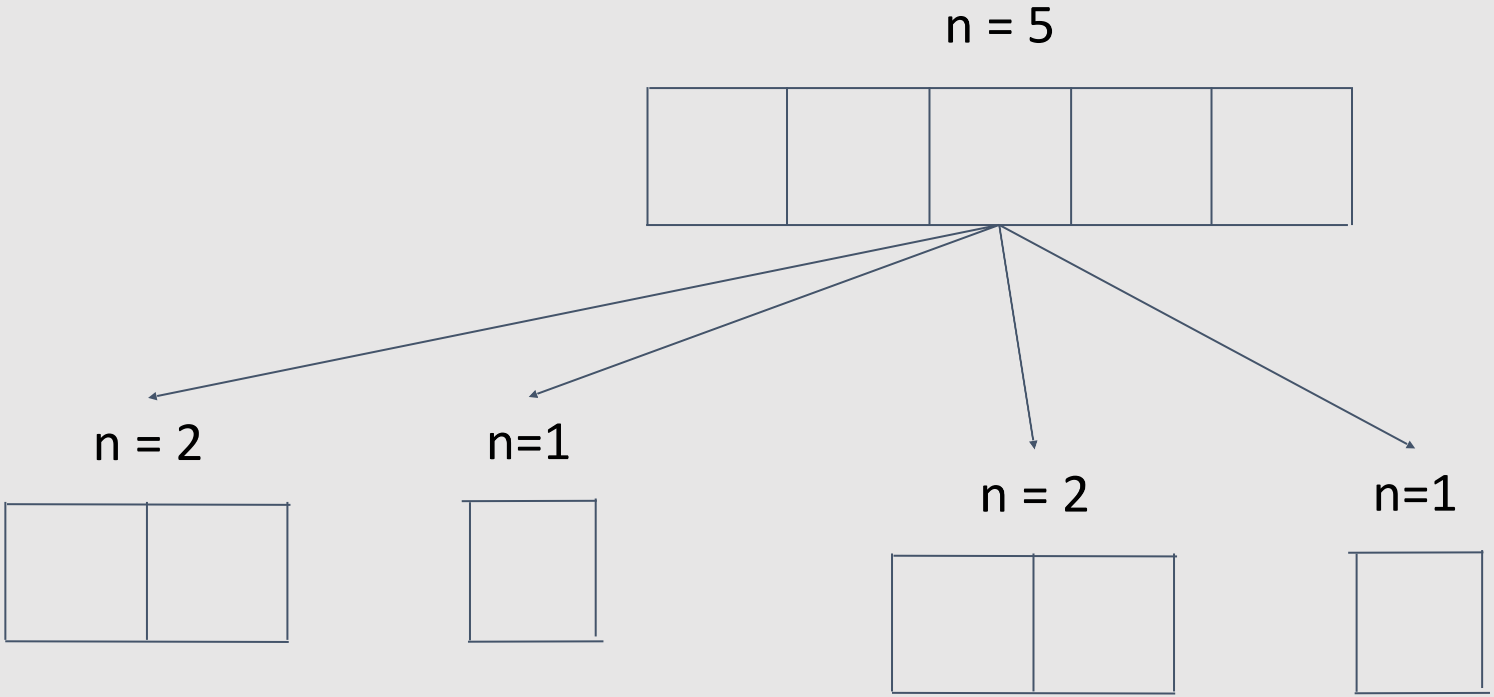
		X	X	X		

Idea

If there is already the cross in the i-th cell, we cant place a cross into the (i-1), (i + 1), (i - 2), (i + 2).

			X			


X



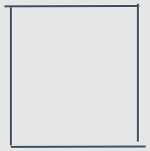
n = 5
n = 2
n=1
n = 2
n=1

lose

n = 2

lose

n=1



Game 1

The formula turns out to be as follows:

g(n) = mex(g[n - 3], g[n - 4], g[n - 5], g[i] xor g[n - 5 - i])

Game 3

There is a pile of n stones. In one move, a player can take any non-zero number of stones from any pile, or split any pile into two non-empty piles. The player who cannot make a move loses.

Removing stones

n stones -> n - 1 stones or n - 2 or n - 3 or … or 1

g(n) = mex(g(n - 1), g(n - 2), g(n - 3) .. g(1))

Example

4 stones -> 3 stones or 2 or 1 or 0

$g(4) = \mathrm{mex}(g(3), g(2), g(1), g(0))$

Splitting stones

n stones -> i stones and n - i stones

$g(n) = \text{mex}(g(i) \text{ xor } g(n - i))$ for all i from 1 to n - 1

Splitting stones

4 stones -> 1 stone and 3 stones or 2 stones and 2 stones or

3 stones and 1 stone

g(n) = mex(g(1) xor g(3), g(2) xor g(2), g(3) xor g(1))

Splitting stones

because we have g(0) -> it's obvious that every state is win.

Game 3’

There are n piles of stones. In one move, a player can take any non-zero number of stones from any pile, or split any pile into two non-empty piles. The player who cannot make a move loses.

Game 3’

As we said SG makes a transition between some hard game and nim.

Game 3'

What we have done if we have nim(x_1, x_2, …, x_n)

$x1$ xor .. x_n

Game 3’

SG(x_1) xor SG(x_2) xor .. xor SG(x_n)

Game 4

There are n bowling pins arranged in a row. In one strike, a player can knock down either one pin or two adjacent pins. The player who knocks down the last pin wins.

Remove one pin

The same as crosses

g(n) -> mex(g(n - 1), g(n - 2) xor g(1), g(n - 3) xor g(2))

Remove two pins

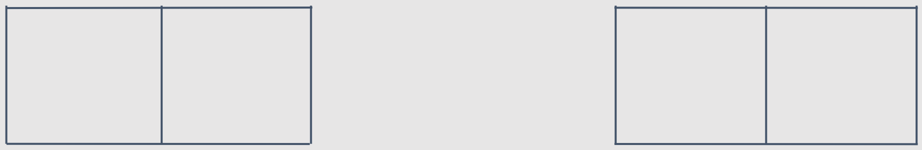
The same as crosses

g(n) -> mex(g(n - 2), g(n - 3) xor g(1), g(n - 4) xor g(2))

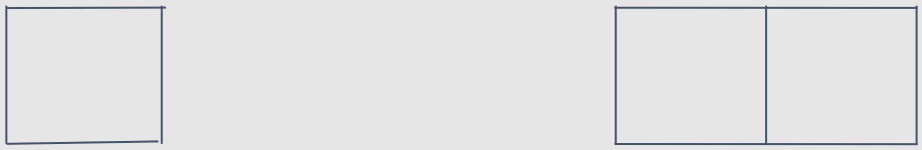
Example

g(6) = mex(g(5), g(4) xor g(1), g(3) xor g(2), … , g(4), g(3) xor g(1), g(2) xor g(2)) -> we have g(2) xor g(2), it's always 0, so g(6) is win and we can go to g(2) xor g(2)

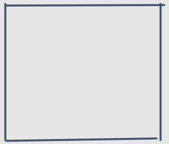
my 1 turn

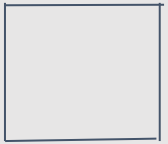


your 1 turn

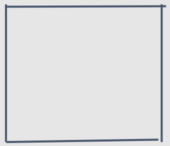


my 2 turn





your 2 turn



my 3 turn. I won!!

Game 5

There is a staircase with n steps (numbered from 1 to n), and on the i-th step there are a_i coins. In one move, it is allowed to move some non-zero number of coins from the i-th to the i-1-th step. The player who cannot make a move loses.

Game 5

If we try to solve the problem naively, it turns out that we have two operations in one move:

1) Decrease the pile a_i
2) Increase the pile a_(i - 1)

So, we won't be able to do it, as in Nim we can only perform one operation.

Nim is bad

[3, 2, 3, 2] -> [3, 3, 2, 2]

Game 5

Let's solve Nim for either even or odd piles depending on the numbering.

I will solve only for even ones, that is, (a_0, a_2, … ,a_4).

Now we have two requests:

1) Increase a pile from the set
2) Decrease a pile from the set

So let's solve the regular Nim.

Game 5

a_0 xor a_2 xor a_4 xor .. xor a_(last even)

Example

[3, 2, 2, 3]

3 xor 2 = 1

[3, 2, 3, 2]

3 xor 3 = 0

[5, 0, 3, 2]

5 xor 3 = 6

[3, 0, 3, 2]

3 xor 3 = 0

Example

[3, 1, 2, 2]

3 xor 2 = 1

[2, 1, 2, 2]

2 xor 2 = 0

[2, 1, 4, 0]

2 xor 4 = 6

Example

[2, 3, 2, 0]

[5, 0, 2, 0]

[2, 0, 2, 0]

[2, 2, 0, 0]

[0, 2, 0, 0]

[1, 1, 0, 0]

[0, 1, 0, 0]

Moore’s nim

There are n piles of stones of size a_i. Also, a natural number k is given. In one move, a player can reduce the sizes of one to k piles (i.e., simultaneous moves in several piles at once are now allowed). The player who cannot make a move loses.

Obviously, when k=1, Moore's Nim turns into regular Nim.

Example

k=2

a = [1, 2, 3]

my turn = [1, 2, 3] -> [1, 1, 1] -> any -> I win

K-xor

k-xor

k = 2 - regular xor

$0 + 0 = 0$

$1 + 0 = 1$

$0 + 1 = 1$

$1 + 1 = 0$

K-xor

k-xor

You work with binary numbers, you sum up digits in each position and then take sum % k

For example

3 (3-xor) 5 (3-xor) 1 = 011 3-xor 101 3-xor 001 = 113 = 110 2 (3-xor) 0 (3-xor) 2 = 020

Moore’s nim

The solution to such a problem is actually very simple. If $(k+1)$-xor = 0, then the current position is losing; otherwise, it's winning (and from it, there is a move to a position with zero value).

Example

k=2

a = [1, 2, 3]

01 (k+1)-xor 10 (k+1)-xor 11 = 22

[1, 2, 3] -> [1, 1, 1]

01 (k+1)-xor 01 (k+1)-xor 01 = 00

Moore's nim

Like for Nim, the proof consists of describing the players' strategies: on one hand, we show that from a game with a zero value, we can only move to a game with a non-zero value, and on the other hand, we show that from a game with a non-zero value, there exists a move to a game with a zero value.

Firstly, let's show that from a game with a zero value, we can only move to a game with a non-zero value. This is quite clear: if the sum modulo k+1 was equal to zero, then after changing from one to k bits, we couldn't obtain a zero sum again.

Example to proof

k=2

[1, 2, 3] -> [1, 1, 1]

01 (k+1)-xor 01 (k+1)-xor 01 = 00

If you change 0 bits -> 00

If you change 1 bit -> 20

If you change 2 bits -> 10

Example

k=2

a = [1, 2, 3, 3, 2]

001 (k+1)-xor 010 (k+1)-xor 011 (k+1)-xor 011 (k+1)-xor 010 =

000

a = [1, 2, 2]

001 (k+1)-xor 011 (k+1)-xor 010 = 022

Moore’s nim

To show that the transition from non-zero xor to zero xor exists we will just make this transition.

Let’s imagine (x_0 (k + 1-xor) x_1 (k + 1-xor) .. x_n) = s

s = 0221

Example

k=2

a = [1, 2, 3, 4, 2]

001 (k+1)-xor 010 (k+1)-xor 011 (k+1)-xor 100 (k+1)-xor 010 =

102

biggest bit is 2, so we take 4, because 2^2 = 4 and we will

change it to 3, because 100 xor 101 = 001 = 1

[1, 2, 3, 4, 2] -> [1, 2, 3, 1, 2]

Example

a = [1, 2, 3, 1, 2]

001 (k+1)-xor 010 (k+1)-xor 011 (k+1)-xor 001 (k+1)-xor 010 =

000

turn is finished

Example

k=2

a = [5, 2, 3, 4, 2]

101 (k+1)-xor 010 (k+1)-xor 011 (k+1)-xor 100 (k+1)-xor 010 =

202

biggest bit is 2, so we take 4, because 2^2 = 4 and we will

change it to 3, because 100 xor 101 = 001 = 1

[1, 2, 3, 4, 2] -> [5, 2, 3, 1, 2]

Example

a = [5, 2, 3, 1, 2]

101 (k+1)-xor 010 (k+1)-xor 011 (k+1)-xor 001 (k+1)-xor 010 =

100

5 = 101 xor 100 -> 1

a = [1, 2, 3, 1, 2]

k-xor = 0

Example

a = [1, 2, 3, 1, 2]

01 3-xor 10 3-xor 11 3-xor 01 3-xor 10 = 00

misère nim

The nim game we discussed earlier is also known as "normal nim." In contrast, there is "misère nim," where the player who makes the last move loses (rather than wins).

misère nim

the solution is the same

if (x_0 xor x_1 xor x_2 .. xor x_{n-1}) = 0, then you lose

otherwise you win.

x = [1, 1, 1.. .. 1] - vice versa

Example with ones

regular nim: [1] - You win

regular nim: [1, 1] - You lose

misère nim: [1] - You lose

misère nim: [1, 1] - You win

Example

[1, 2, 4] -> (1 xor 2 xor 4) = 7 - winning

[1, 2, 3] -> (1 xor 2 xor 3) = 0 - losing

[1, 2, 1] -> (1 xor 2 xor 1) = 2 - winning

CHANGE - in regular nim you do this->

[1, 0, 1] -> (1 xor 0 xor 1) = 0 - lose

Example

[1, 2, 4] -> (1 xor 2 xor 4) = 7 - winning

[1, 2, 3] -> (1 xor 2 xor 3) = 0 - losing

[1, 2, 1] -> (1 xor 2 xor 1) = 2 - winning

CHANGE - in misere nim [1, 0, 1] is winning condition

[1, 1, 1] -> (1 xor 1 xor 1) = 1 - lose

Example

[1, 1, 1] = 1 xor 1 xor 1 = 1

0

[1, 1, 1] -> You cant have more than 1 step

Example

From big game to the [1….1] you play regular nim

and then you make stupid turns

Interesting solution

$(x\_0 \text{ xor } x\_1 \text{ xor } x\_2 \ldots x\_n\text{-}1 \text{ xor } z)$

$z = (x\_0 = x\_1 = .. = x\_n - 1 = 1)$

Example

[1] -> 1 xor 1 = 0

[1, 1] -> 1 xor 1 xor 1 = 1

[3, 1] -> 1 xor 3 = 2

All codes

graph game - https://ideone.com/CBwdpE