1. Introduction to Combinatorial Game Theory

Properties:

1. There is no move by chance
2. All the information is known by both players (no moves of pieces are hidden, unlike most of the card games for example)
3. Once the game ends, we know the winner
4. Both players play optimally.

Examples:

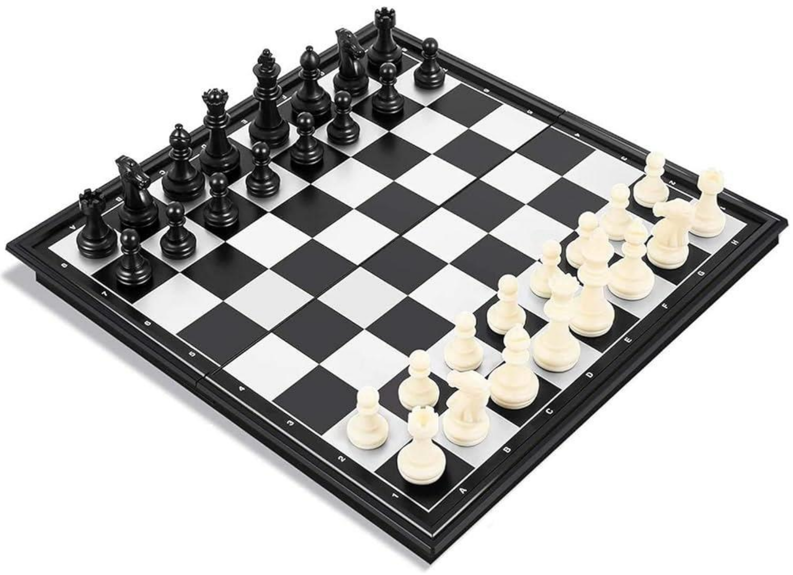


Figure 1-1 Chess

Figure 1-2 Shogi

And so many other games such as Go, Dots and Boxes, tic tac toe, checkers and **Nim** game.

In these games, 2 players take turns and the last player to move is the winner!

(The player who is NOT able to make a move, loses the game).

Let's start with a simple game example to show some principles:

Alice and Bob play a game, where there is a pile with 21 chips, each player can take 1, 2 or 3 chips in their turn. Alice Goes first.

Let's see for every number of chips, whether it's a **winning or losing position** w.r.t Alice (the first player).

0	1	2	3	4	5	6	7	8	9
L	W	W	W	L	W	W	W	L	W

Let's take a dive into that!

0 → Alice cannot move, so he loses

1 → Alice can take the whole pile, leaving nothing to Bob, so Alice wins

2 → Alice can take the whole pile, leaving nothing to Bob, so Alice wins

3 → Alice can take the whole pile, leaving nothing to Bob, so Alice wins

4 → Whatever move Alice make, he goes to either 3, 2 or 1 which are all winning positions, that means the next move Bob will win, so Alice loses

5 → Alice can take 1 pile and go to 4, which is a losing position for Bob,

So, Alice wins

6 → Alice can take 2 piles and go to 4, which is a losing position for Bob,

So, Alice wins

7 → Alice can take 3 piles and go to 4, which is a losing position for Bob,

So, Alice wins

8 → Whatever move Alice make, he goes to either 7, 6 or 5 which are all winning positions, that means the next move Bob will win, so Alice loses

9 → Alice can take 1 pile and go to 8, which is a losing position for Bob,

So, Alice wins.

That means if the number is divisible by 4, Bob wins otherwise, Alice wins.

Conclusion:

- If all moves at some position lead to winning positions, that position is a losing position.
- If at least one move at some position lead to a losing position, that position is a winning position.

Formally, we call losing positions "P positions" because the Previous player wins or in other words, good for the Previous player, who just had a turn.

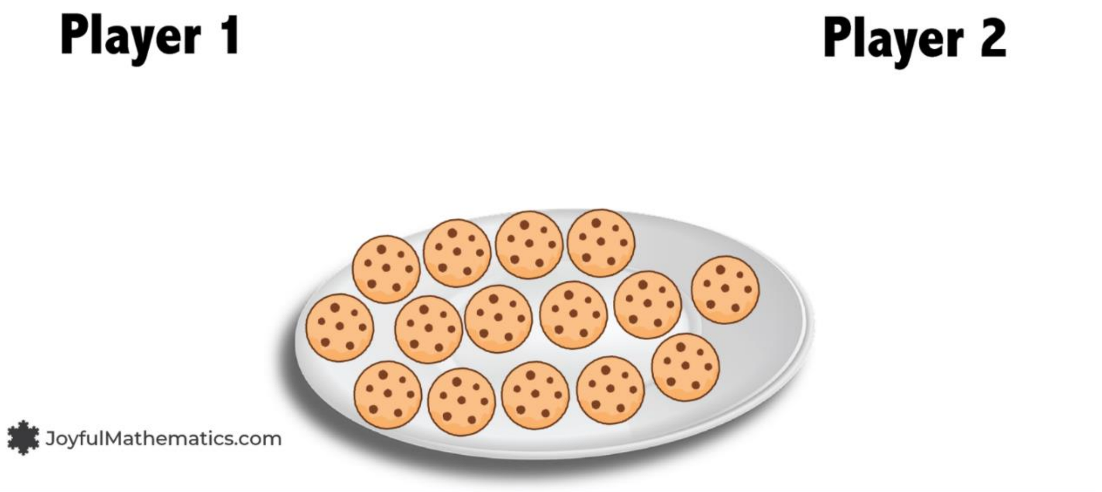
we call winning positions "N-position" because they are good for the Next player, who is about to have a turn.
An N-position has the property that there is at least one move to a P-position.

2. The game of Nim

Two players and K piles of chips

On each turn, a player can remove some (non-zero) number of stones of one pile.

The player who cannot make a move, loses.



Let's analyze a simple version when **K = 1**

For the first player:

Then if number of chips $= 0$, he cannot make any move, thus the first player loses. i.e., it's a P-position (losing position)

And if number of chips $\neq 0$, the first player always takes the whole pile, thus the first player wins. i.e., it's an N-position (winning position)

Let's analyze a *harder* version when **K = 2**

One pile contains $\boldsymbol{n}$ chips, the other contains $\boldsymbol{m}$ chips

For the first player:

Then if $n = m$, the first player loses. i.e., it's a P-position (losing position)

Why?

If the first player makes a move, the second player will always copy that move, thus keeping the two piles equal all the time, that guarantees that the second player will always make the last move and the first player loses

Otherwise, if $n \neq m$, the first player always can always take some number x from the larger pile, making them equal.

Thus, going to the state where $n = m$, which is a P-position (losing position), thus the first player wins. i.e., it's an N-position (winning position)

For $K \geq 3$, it would be way harder to analyze! Let's see a representation that may help us in the analysis of some general K

Given a game of Nim whose current game state consists of $\boldsymbol{n}$ piles with $\boldsymbol{p_1, p_2, p_3, \cdots, p_n}$, then the current game state is a winning state if the Nim-sum, $\boldsymbol{p_1 \oplus p_2 \oplus p_3 \oplus \cdots \oplus p_n}$, is non-zero, and a losing state if it is equal to zero, where $\boldsymbol{\oplus}$ is the bitwise XOR operation. Proof:

1. The empty game is a losing state with **Nim-sum** = 0. This follows from our observations earlier.
2. If the current **Nim-sum** is 0, then any move will make it **non-zero**. Recall that a move replaces p_i with $(p_i - s)$. So, keeping in mind that the XOR-inverse of each integer is itself, that means the new Nim-sum after a move is made becomes $(p_1 \oplus p_2 \oplus p_3 \oplus \cdots \oplus p_n) \oplus p_i \oplus (p_i - s)$; we remove p_i from the Nim-sum, then add in $(p_i - s)$. But, if the current Nim-sum was already 0, then this is just $p_i \oplus (p_i - s)$, and this is only equal to 0 when $p_i = (p_i - s)$. But the rules state you have to always take a positive integer of stones on your turn, so $s \neq 0$, and it is impossible for this new Nim-sum to still be 0.

3. If the current Nim-sum is non-zero, then there always exists a move to make it equal to 0. Suppose the current Nim-sum is $N > 0$. We need to choose p_i and s such $p_i \oplus (p_i - s) = N$, since the constraint is $0 < s \leq p_i$. Consider the fact that N, since it is nonzero, must have a greatest significant bit, the g_{th} bit. Furthermore, at least one of the piles p_i will have the g_{th} bit as a 1 as well; if all the piles had a 0 there, then it couldn't have been the greatest significant bit of their XORsum after all. So, we choose one such pile as our p_i, and we construct our $p_i - s$ as the following:

- All bits greater than the g_{th} bit we keep the same as p_i.
- The g_{th} bit is set to 0.
- All bits less than the g_{th} bit we make match those of $N \oplus p_i$.

Notice that this construction is always less than p_i, since we know their first difference is at the g_{th} bit, therefore this is a valid choice of $p_i - s$, from which we can easily recover what the s should be. We can see that the $p_i \oplus (p_i - s) = N$ because

- All bits in N greater than the g_{th} bit were already 0 (since the g_{th} bit was defined to be the greatest significant bit). Then, we know that all bits in $p_i \oplus (p_i - s)$ greater than the g_{th} bit are also 0, as that was how we constructed it.
- The g_{th} bit in N is 1, by definition as the greatest significant bit, and the g_{th} bit in $p_i \oplus (p_i - s)$ is also 1, since it is 1 in p_i and 0 in $p_i - s$.
- All bits in $p_i \oplus (p_i - s)$ less than the g_{th} bit are guaranteed to match that of N, since $p_i \oplus (p_i - s) = p_i \oplus (p_i \oplus N) = N$ in this range.

So, $N \oplus p_i \oplus (p_i - s) = 0$,
and thus the new Nim-sum after the move has become 0.

Now, with these tools in hand, we can see that:

- If the current turn player has a zero state, they have no choice but to 'disturb' it into a nonzero state.
- The next player then, given a nonzero state, always is able to 'restore' it back into a zero state.
- Thus, if the players know what they are doing, the first player is helpless if the Nim-sum is 0. The second player can always ensure that the Nim-sum is 0 if and only if it is the first player's turn. There are only a finite number of moves in a game of Nim (obvious, but you can prove it by induction, using the fact that pile sizes only get smaller). Eventually, the game ends when the last stone is taken, meaning the Nim-sum is 0. So, we know that it is the first player's turn when the game ends, thus the second player can ensure that the first player loses.
- Similarly, if the current turn player has a nonzero state, they can restore it to a zero state and pass the board over to the second player, who by our logic, must be the loser, thus the first player is the winner.

Conclusion:

If $p_1 \oplus p_2 \oplus p_3 \ldots \oplus p_n = 0$, the first player loses

Otherwise, if $p_1 \oplus p_2 \oplus p_3 \ldots \oplus p_n \neq 0$, the first player wins

3. Graph games

A directed graph G is a pair of (V, N) $such\ that\ V\ is\ a\ set\ and\ N\ is\ a\ subset\ of\ V$

Then, $g: V\{0, 1, 2, \dots\}\ is\ the\ Sprague\ Grundy\ function\ iff$

$g(x) = mex\{g(y) \in N(x)\ for\ all\ x \in V$

The important Sprague-Grundy theorem states that **these games are equivalent to playing Nim, but instead of getting the Nim-sum by taking the XOR of the piles, we take the XOR of their Grundy numbers.**

Let's apply this on the very first simple game we analyzed:

Alice and Bob play a game, where there is a pile with 9 chips, each player can take 1, 2 or 3 chips in their turn. Alice Goes first. Let's evaluate the **Grundy number** of each position (number of chips)

x	0	1	2	3	4	5	6	7	8	9
G	0	1	2	3	0	1	2	3	0	1

0 → Alice can move to ONLY 0, so mex is 0

1 → Alice can move to $x: \{0\}, with\ G: \{0\}\ so\ mex = 1$

2 → Alice can move to $x: \{0,1\}, with\ G: \{0,1\}\ so\ mex = 2$

3 → Alice can move to $x: \{0,1,2\}, with\ G: \{0,1,2\}\ so\ mex = 3$

4 → Alice can move to $x: \{1,2,3\}, with\ G: \{1,2,3\}\ so\ mex = 0$

5 → Alice can move to $x: \{2,3,4\}, with\ G: \{2,3,0\}\ so\ mex = 1$

6 → Alice can move to $x: \{3,4,5\}, with\ G: \{3,0,1\}\ so\ mex = 2$

7 → Alice can move to $x: \{4,5,6\}, with\ G: \{0,1,2\}\ so\ mex = 3$

8 → Alice can move to $x: \{5,6,7\}, with\ G: \{1,2,3\}\ so\ mex = 0$

9 → Alice can move to $x: \{6,7,8\}, with\ G: \{2,3,0\}\ so\ mex = 1$

The $G(9) = \ 1 \neq \ 0$

So, the first player (Alice) wins.

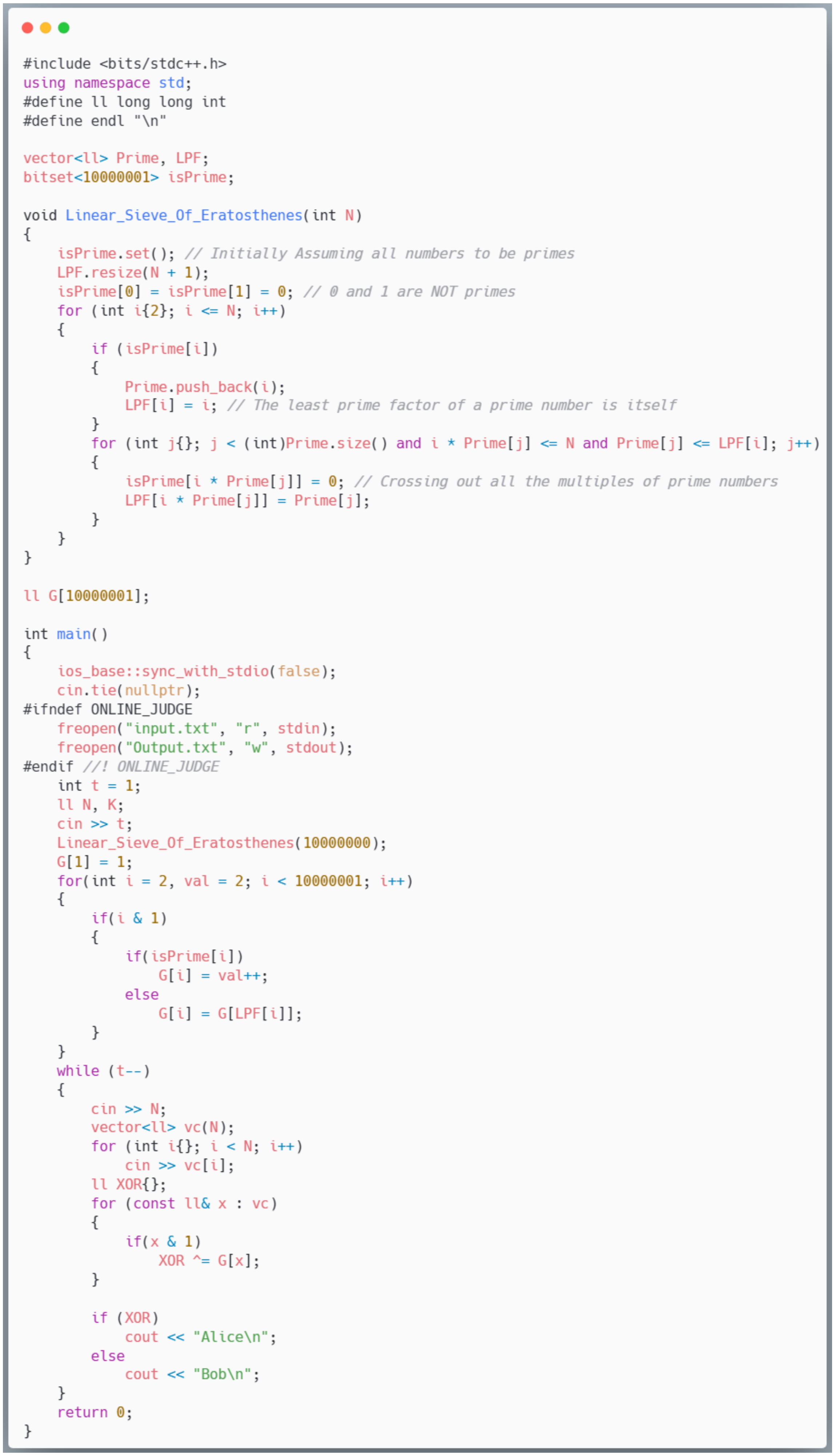
The complexity in these kinds of programming problems in contests usually then comes down to finding some efficient formula for the **Grundy** numbers. The Sprague-Grundy theorem is actually far more general, and says that *any impartial combinatorial game* is essentially Nim, due to the Grundy numbers. Why? Well, the argument in our proof would work with any graph, not just Nim-games with piles. Any game which can be represented as a DAG (Directed Acyclic Graph) can also be formulated in terms of **Grundy** numbers, and thus Nim!

A good practice on this would be the problem Not a Nim Problem

Hint:

Try to find the Grundy function of the first 10 numbers and deduce a pattern!

My full source code will be attached in the next page.



```cpp
#include <bits/stdc++.h>
using namespace std;
#define ll long long int
#define endl "\n"

vector<ll> Prime, LPF;
bitset<10000001> isPrime;

void Linear_Sieve_Of_Eratosthenes(int N)
{
    isPrime.set(); // Initially Assuming all numbers to be primes
    LPF.resize(N + 1);
    isPrime[0] = isPrime[1] = 0; // 0 and 1 are NOT primes
    for (int i{2}; i <= N; i++)
    {
        if (isPrime[i])
        {
            Prime.push_back(i);
            LPF[i] = i; // The least prime factor of a prime number is itself
        }
        for (int j{}; j < (int)Prime.size() and i * Prime[j] <= N and Prime[j] <= LPF[i]; j++)
        {
            isPrime[i * Prime[j]] = 0; // Crossing out all the multiples of prime numbers
            LPF[i * Prime[j]] = Prime[j];
        }
    }
}

ll G[10000001];

int main()
{
    ios_base::sync_with_stdio(false);
    cin.tie(nullptr);
#ifndef ONLINE_JUDGE
    freopen("input.txt", "r", stdin);
    freopen("Output.txt", "w", stdout);
#endif //! ONLINE_JUDGE
    int t = 1;
    ll N, K;
    cin >> t;
    Linear_Sieve_Of_Eratosthenes(10000000);
    G[1] = 1;
    for(int i = 2, val = 2; i < 10000001; i++)
    {
        if(i & 1)
        {
            if(isPrime[i])
                G[i] = val++;
            else
                G[i] = G[LPF[i]];
        }
    }
    while (t--)
    {
        cin >> N;
        vector<ll> vc(N);
        for (int i{}; i < N; i++)
            cin >> vc[i];
        ll XOR{};
        for (const ll& x : vc)
        {
            if(x & 1)
                XOR ^= G[x];
        }

        if (XOR)
            cout << "Alice\n";
        else
            cout << "Bob\n";
    }
    return 0;
}
```