Advanced Nim

IE616 Mini-Project

Atishay Jain (210050026)
Aryan Mathe (210050021)
Chaitanya Aggarwal (210050038)

210050026@iitb.ac.in
210050021@iitb.ac.in
210050038@iitb.ac.in

Contents

PART

I

Nim

Section 1

Standard Nim

Lets start with discussion on the standard game of Nim, which is a impartial game played between 2 players.

Definition 1
Nim is an impartial game in which a position consists of l piles of stones of sizes $a_1, a_2, ..., a_l$. A player removes one or more (up to all) stones from a chosen pile in its move. The last player to take a stone wins.

There are different types of positions in Nim. For any non-negative integer α let $*\alpha$ denote a Nim position consisting of a single pile of α stones. Then, we denote a position with piles of sizes $a_1, a_2, ..., a_l$ by the sum notation $*\alpha_1 + *\alpha_2 + ... + *\alpha_l$.

Subsection 1.1

MEX (Minimum Excludant)

Suppose you have a set of numbers.

Definition 2
Minimum excludant (MEX) of a set is defined as the least non-negative number which is not present in the set.

In other words, if we consider the complement set, then MEX is the minimum non-negative element in it. Such MEX values also have applications in greedy graph coloring algorithms.

Example

- $MEX(\phi) = 0$
- $MEX(1, 2, 5, 100) = 0$
- $MEX(1, 5, 9, 13) = 0$
- $MEX(0, 2, 4, 6, 8...) = 1$
- $MEX(0, 1, 2, 3..., \omega) = \omega + 1$
- $MEX(0, 1, 2, 3...) = \omega$

Remark Here, ω represents the ordinal limit of natural numbers.

Subsection 1.2
Grundy Number ($G(N)$)

It is a number that we use to define the state of a combinatorial game. We can also use it to define any impartial game like Nim also. The calculation of Grundy number is based on MEX calculation.

Definition 3
Grundy Number is 0 for a game, which is immediately lost by player who makes first move, else the Grundy Number is the MEX of all the numbers which are possible in next positions by making a move in the game

Example

Lets take a simple example to understand G(N). A game of standard Nim with a single pile of N coins and 2 players. We will try to find $G(N)$ for different values of N -

- $N = 0$: first player has nothing to pick, he lost immediately $\implies G(N) = 0$
- $N = 1$: first player will pick this 1 coin and 0 coin will be left then which have $G(0) = 0$. So, $G(1) = MEX(G(0)) = 1$
- $N = 2$: There are 2 cases, if first player picks 1 coin then we will be left with a single coin with $G(1) = 1$ or if the first player picks both coins, then we will be left with 0 coins with $G(0)0$. Therefore, $G(2) = MEX(G(0), G(1)) = 2$
- $N = 3$: After making cases similar to $N = 2$ case, we get $G(3) = 3$, and in general $G(N) = N$

We know that we use the XOR (exclusive-OR) of Grundy Number of size of piles to solve the standard game of Nim. In the next section, we will see a modified version of Nim called Advanced Nim or Zero move Nim. We will try to solve it using a similar but modified approach and write a code for its implemetation as well. We have also designed a Dynamic Programming based implementation for solving it, which we will review later.

PART

II

Advanced Nim

Section 2

Zero Move Nim Game

In this modified version of Nim, we include a new **Zero move** rule.

Definition 4
Zero move rule is such that it allows the players to remove zero coins in a move and such a move can be used only once per “pile" by either of the players.

For solving this version of Nim, lets start by deducing a way to find Grundy Number for it -

Subsection 2.1

Grundy Number for Advanced Nim

Lets start with some notation. We say the move of picking 0 number of coins as zero move. We will denote nF as the pile of size n in which no zero moves are left, that is the zero move of this pile which was allowed to be used only once has been used. The pile of size n in which zero moves are left will be represented by nT.

Remark Note that we can reach the state of nF from nT for a value of n, but not reverse.

Remark Note that $G(N)$ is actually same as $G(nT)$

Example Again we will look into the same previous example setting to understand -

- $N = 0$: $G(0) = 0$, as first player loses immediately
- $N = 1$: There are 2 cases here -
  1. If $nF = 1$: This means pile has no zero moves left, so first player picks 1 coin and thus $G(1F) = MEX(0) = 1$
  2. If $nT = 1$: There are 2 moves possible. First player can pick 1 coin and then 0 coins left, which is same as above and has $G(0)$. Second possibility is that first player makes *zero move*, now game reaches $1F$ state again with grundy number as $G(1F)$. So, $G(1T) = MEX(0, G(1F)) = 2$

  So, overall $G(1) = G(nT) = 2$
- $N = 2$: There are 2 cases here -
  1. $nF = 2$: This means a pile of size 2 and no zero moves left. So either first player picks 2 coins resulting in $G(0)$ state or, he can pick 1 coin resulting in $1F$ state. Therefore $G(2F) = MEX(0, G(1F)) = 2$

2. $nT = 2$: There are 3 possibilited here. First player can pick 2 coin resulting in $G(0)$ state, or he can pick 1 coin resulting in new $1T$ state. Another possibility is that he can make a zero move resulting in $2F$ state. Therefore, $G(2T) = MEX(0, G(1T), G(2F)) = MEX(0, 2, 2) = 1$

Hence, $G(2) = G(2T) = 1$

- $N = 3$: There are 2 cases -

1. $nF = 3$: After making similar cases as shown in previous calculation, here we will get $G(3F) = MEX(0, G(1F), G(2F)) = MEX(0, 1, 2) = 3$
2. $nT = 3$: Similarly here we will consider a zero move and get $G(3T) = MEX(0, G(1T), G(2T), G(3F)) = MEX(0, 2, 1, 3) = 4$

Therefore, we get $G(3) = G(3T) = 4$

Remark In $N = 0$ case, we cannot pick a 0 coin as a zero move because zero move is defined for piles of nonzero size only

Remark Note that in $N = 2$ case, you cannot reach $1F$ from current state of $2T$ because it will require 2 simultanoues moves of picking one coin and a zero move, which is not possible

So, from the example, we can say that for a pile of size N, if it is in nF state, then

$$G(nF) = MEX(0, G(1F), G(2F), ...G((n-1)F)) = n$$

or if is in nT state, then

$$G(nT) = MEX(0, G(1T), G(2T), ...G((n-1)T), G(nF))$$

After performing calcultaion for more values of N and since there is a pattern in $G(1T)$ and $G(1F)$, we can easily deduce that for in general for a pile of size N,

Formulae

$$G(N) = \begin{cases} N+1 \ , \ NisOdd \\ \\ N-1 \ , \ NisEven \end{cases}$$

Subsection 2.2

Winning Strategy and States

Here, also we have used cumulative XOR for finding the winning state of the game. These are based on an assumption considering that both the players play optimally, that is they do not make any moves which may reduce the winning chances of themselves.

2.2.1 Winning State

Properties The player to play first move will win the game if and only if the cumulative XOR (also known as **Nim Sum**) of the Grundy Numbers of sizes of all the piles is **non-zero**.

2.2.2 Losing State

Properties The player to play first move will lose the game if and only if the cumulative XOR (also known as **Nim Sum**) of the Grundy Numbers of sizes of all the piles is **zero**.

Remark Note that Grundy number here refers to the modified Grundy Number which we calculated above for Advanced Nim

Example

1. `Input` : Size of piles of coins are = { 6, 4, 8 }
   `Output` : Player 1 wins!
   `Explanation` : Grundy numbers for each pile are { 5, 3, 7 } so $NimSum = (5 \oplus 3 \oplus 7) = 1$, which is nonzero

2. `Input` : Size of piles of coins are = { 4, 9, 10 }
   `Output` : Player 2 wins!
   `Explanation` : Grundy numbers for each pile are { 3, 10, 9 } so $NimSum = (3 \oplus 10 \oplus 9) = 0$

SUBSECTION 2.3
Code Implementation for Advanced Nim

2.3.1 Using Dynamic Programming

We implemented the dynamic programming approach to determine the outcome of such play using **Sqrauge Grundy** themorem and **MEX Principle**. The explaination of working of our code is given in the comments -

```cpp
#include <iostream>
#include <bits/stdc++.h>
using namespace std;

#define LOG(x) cout << x << endl

// This function finds the MEX value of a set of values which is
// present in a list named "arr" using a simple algoritm.
// In that algorithm we iterate through the elements of the array,
// and negate the element which has index equal to the current element
// After doing this process for all teh elements we simply finds the
// index which has a non negative element adn returns it's index.
int find_mex(int index, vector<vector<int>>& dp){
    vector<int>arr = dp[1];
    arr[index] = dp[0][index];
    int i=0;
```

```
17      // Handling the case for 0.
18      for(int i=0; i<=index; i++){
19          arr[i] = arr[i] + 1;
20      }
21      // Negating the elements
22      for(int i=0; i<=index; i++){
23          int temp = arr[i];
24          if(abs(temp)-1 <= index){
25              arr[abs(temp) - 1] = -arr[abs(temp) - 1];
26          }
27      }
28      // Finding the index which has +ve element.
29      for(i=0; i<=index; i++){
30          if(arr[i] > 0){
31              break;
32          }
33      }
34      return i;
35  }
36
37  int get_Nimber_AdvNim(vector<int>& pile){
38      // Getting the max_size of the DP array.
39      int max_val = INT_MIN;
40      for(auto i:pile){
41          max_val = max(i, max_val);
42      }
43      // Creating the vector(array)
44      vector<vector<int>> dp(2);
45      // dp[0] ==> An array which has the nimber equivalent
46      // to the standard NIM game
47      dp[0] = vector<int>(max_val+1, 0);
48      // dp[0] ==> An array which consists of the nimber equivalent
49      // of the Advanced NIM game
50      dp[1] = vector<int>(max_val+1, 0);
51      for(int i=0; i<max_val+1; i++){
52          dp[0][i] = i;
53      }
54      // Finding the Nimber value for all the pile sizes less that
55      // and equal to the max pile size.
56      for(int i=1; i<dp[0].size(); i++){
57          dp[1][i] = find_mex(i, dp);
58      }
59      // Finding the final nimber equivalent of all the piles using "XOR SUM".
60      int final_ans = 0;
61      for(auto i:pile){
62          final_ans^=dp[1][i];
63      }
```

```
64      // returning the final nimber equivalent of the entire game.
65      return final_ans;
66  }
67
68  int main(){
69      cout << "Input Number of Piles: ";
70      int num_of_pile;
71      cin >> num_of_pile;
72      LOG("");
73      cout << "Input Pile Sizes for each of them: ";
74      vector<int> piles(num_of_pile);
75      for(int i=0; i<num_of_pile; i++){
76          cin >> piles[i];
77      }
78      int final_ans = get_Nimber_AdvNim(piles);
79      LOG("");
80      LOG("Nimber Equivalent: " << final_ans);
81      // If the nimber value is "0" then that's a P position and Second Player wins.
82      if(final_ans == 0){
83          LOG("");
84          LOG("Second Player Wins !!");
85      }
86      // If the nimber value is non zero that that's a N position and First Player Wins.
87      else{
88          LOG("");
89          LOG("First Player Wins !!");
90      }
91      return 0;
92  }
```

2.3.2 Using Theoretical result of Grundy Number

```
1   #include <iostream>
2   #include <bits/stdc++.h>
3   using namespace std;
4
5   // Simple implementation using the result obtained
6   // of Grundy number and winning state
7   void game(vector<int>& piles){
8       int Nimsum = 0;
9       for (int i = 0; i < piles.size(); i++){
10          // If the size of the pile is odd then it's Grundy Number will be G(N+1)
11          if (piles[i] % 2 == 1)
12              Nimsum ^= (piles[i] + 1);
13          // If the size of the pile is even then it's Grundy Number will be G(N-1)
14          else
15              Nimsum ^= (piles[i] - 1);
16      }
```

```
    // If the Nim sum is non-zero then A will win the game
    if (Nimsum != 0){
        cout << "First Player Wins !!" << endl;
    }
    // If the Nim sum is zero then B will win the game
    else{
        cout << "Second Player Wins !!" << endl;
    }
}

int main(){
    cout << "Input Number of Piles: ";
    int num_of_pile;
    cin >> num_of_pile;
    cout << "Input Pile Sizes for each of them: ";
    vector<int> piles(num_of_pile);
    for(int i=0; i<num_of_pile; i++){
        cin >> piles[i];
    }
    game(piles);
}
```

2.3.3 Running Instructions

If the name of the C++ file with this code is advanced_nim_game.cpp, then compile it by running -

```
g++ -o game advanced_nim_game.cpp
```

This will create a binary file named `game` and then you can execute that by command -

```
./game
```

The codes we have written are interactive and will ask for the input of number of piles and number of coins in each pile, and then provide you the winning player.

Section 3

References

- https://cp-algorithms.com/game_theory/sprague-grundy-nim.htmlcrosses-crosses
- Game Theory A Playful Introduction by Matt DeVos, Deborah A. Kent
- https://www.scaler.com/topics/data-structures/advanced-nim-game/
- https://www.geeksforgeeks.org/combinatorial-game-theory-set-2-game-nim/