Combinatorial Game Theory
Grundy Numbers/Nimbers and Mex

Project Category - Game Theory, Dynamic Programming, Recursion, Memoization

Program Name/ Purpose- Calculation of Grundy-Number/Nimbers and Mex

Explanation-

What is Grundy Number?

Wikipedia says that-
"Grundy Numbers or Nimbers are the ordinal numbers endowed with a new nimber addition and nimber multiplication, which are distinct from ordinal addition and ordinal multiplication."

Without diving deep into the technicalities of the terms described in the above definition, we simply define Grundy Numbers in simple words as-

"*Grundy Number is a number that defines a state of a game. So we can define any impartial game in terms of Grundy Number.*"
[To see what an impartial game means see here- http://www.geeksforgeeks.org/introduction-to-combinatorial-game-theory/]

Later in the next set, we will see how Grundy Numbers determines how any Impartial Game (not just the Game of Nim) can be solved once we have calculated the Grundy Numbers associated with that game using Sprague-Grundy Theorem. But in this article, we will discuss how to calculate Grundy Numbers.

Note that Grundy Numbers are also called as Nimbers.

But before calculating Grundy Numbers we will learn about another term- Mex.

What is Mex ?

'*M*inimum *ex*cludant' a.k.a 'Mex' of a set of numbers is the smallest non-negative number not present in the set.

For example,

$$\text{mex}(\emptyset) = 0$$
$$\text{mex}(\{1, 2, 3\}) = 0$$
$$\text{mex}(\{0, 2, 4, 6, \ldots\}) = 1$$
$$\text{mex}(\{0, 1, 4, 7, 12\}) = 2$$

We will see in our program how to calculate the mex using ***calculateMex()*** function.

Now coming back, how to calculate Grundy Numbers?

We will use this definition- *The Grundy Number/ nimber is equal to 0 for a game that is lost immediately by the first player, and is equal to the mex of the nimbers of all possible next positions for any other game.*

We will look three examples and the program to calculate Grundy Number and Mex in each of them. We will see that calculation of Grundy Numbers is done basically by a recursive function called as- ***calculateGrundy()*** function which uses ***calculateMex()*** function as its sub-routine.

Later on, we will see that since it is a recursive function so we can store/cache the subproblems and use them later to reduce the time complexity of our program. Hence, we will use memoization technique.

Example 1-

So the game is as follows-

"Just like a one-pile version of Nim, the game starts with a pile of n stones, and the player to move may take any positive number of stones. Calculate the Grundy Numbers for this game. The last player to move wins. Which player wins the game?"

Now we recall the definition that-

The Grundy Number/ nimber is equal to 0 for a game that is lost immediately by the first player, and is equal to the mex of the nimbers of all possible next positions for any other game.

So, since if the first player has 0 stones, he will lose immediately, so Grundy (0) = 0

If a player has 1 stones, then he can take all the stones and win. So the next possible position of the game (for the other player) is (0) stones

Hence, Grundy (1) = Mex (0) = 1 [According to the definition of Mex]

Similarly,
If a player has 2 stones, then he can take only 1 stone or he can take all the stones and win. So the next possible position of the game (for the other player) is (1, 0) stones respectively.

Hence, Grundy (2) = Mex (0, 1) = 2 [According to the definition of Mex]

Similarly,
If a player has 'n' stones, then he can take only 1 stone, or he can take 2 stone..... or he can take all the stones and win . So the next possible position of the game (for the other player) is (n-1, n-2,....1) stones respectively.

Hence, Grundy (n) = Mex (0, 1, 2, ...n-1) = n [According to the definition of Mex]

We summarize the first the Grundy Value from 0 to 10 in the below table-

N	0	1	2	3	4	5	6	7	8	9	10
Grundy(n)	0	1	2	3	4	5	6	7	8	9	10

Example 2-

The game is as follows-

"A one-pile version of Nim, the game starts with a pile of n stones, and the player to move may take any positive number of stones upto 3 only. The last player to move wins. Which player wins the game?"

Now we recall the definition that-

The Grundy Number/ nimber is equal to 0 for a game that is lost immediately by the first player, and is equal to the mex of the nimbers of all possible next positions for any other game.

So, since if the first player has 0 stones, he will lose immediately, so Grundy (0) = 0

If a player has 1 stones, then he can take all the stones and win. So the next possible position of the game (for the other player) is (0) stones

Hence, Grundy (1) = Mex (0) = 1 [According to the definition of Mex]
Similarly,
If a player has 2 stones, then he can take only 1 stone or he can take 2 stones and win. So the next possible position of the game (for the other player) is (1, 0) stones respectively.

Hence, Grundy (2) = Mex (0, 1) = 2 [According to the definition of Mex]

Similarly,
Grundy (3) = Mex (0, 1, 2) = 3 [According to the definition of Mex]

But what about 4 stones ?

If a player has 4 stones, then he can take 1 stone or he can take 2 stones or 3 stones, but he can't take 4 stones (see the constraints of the game). So the next possible position of the game (for the other player) is (3, 2, 1) stones respectively.

Hence, Grundy (4) = Mex (1, 2, 3) = 0 [According to the definition of Mex]

So we can define Grundy Number of any n>=4 recursively as-

Grundy (n) = Mex[Grundy (n-1), Grundy (n-2), Grundy (n-3)]

We summarize the first the Grundy Value from 0 to 10 in the below table-

N	0	1	2	3	4	5	6	7	8	9	10
Grundy(n)	0	1	2	3	0	1	2	3	0	1	2

Example 3-

The game is as follows-

"The game starts with a number- 'n' and the player to move divides the number- 'n' with the primes- 2, 3, 6 and then takes the floor. If the integer becomes 0, it is removed. The last player to move wins. Which player wins the game?"

We summarize the first the Grundy Value from 0 to 10 in the below table-
Think how we generated this table

N	0	1	2	3	4	5	6	7	8	9	10
Grundy(n)	0	1	2	2	3	3	0	0	0	0	0

Time Complexity-

The time complexity of recursive solution in each of the above three examples is exponential. By using memoization, we can reduce it to polynomial time complexity.

Auxiliary Space-

If we don't use memoization then the only extra space is taken by the recursion stack. Else if we use memoization, then we have to declare an additional array to cache the results of sub-problems.

References-

https://en.wikipedia.org/wiki/Mex_(mathematics)
https://en.wikipedia.org/wiki/Nimber