Teoria de Juego

[Γα=Ω5]

Game states

Consideremos un juego con `n` palitos. Los jugadores A y B se mueven alternados donde A empieza. En cada movimiento, el jugador tiene que quitar 1, 2 o 3 palitos del montón y el jugador que quite el último palito gana el juego.

Por ejemplo, si n = 10, el juego puede desarrollarse de la siguiente manera:

```
A → quita 2 palitos (quedan 8 palitos).
B → quita 3 palitos (quedan 5 palitos).
A → quita 1 palito (quedan 4 palitos)
B → quita 2 palitos (quedan 2 palitos).
A → quita 2 palitos y gana
```

Este juego consta de los estados 0, 1, 2,..., n, donde el número del estado corresponde al número de palitos que quedan.

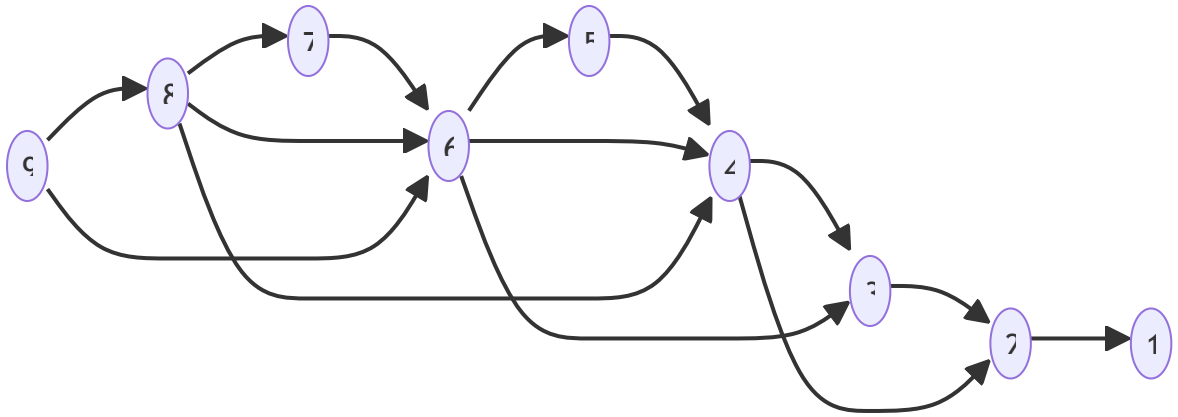
Winning and Losing states (con el caso anterior)

- **Estado ganador** (W): El jugador actual puede asegurar la victoria al mover de manera óptima.
- **Estado perdedor** (L): El jugador actual perderá si el oponente juega de manera óptima.
- **Estado inicial:** El estado `0` es un estado perdedor (L) porque no hay movimientos posibles.
- **Regla general:** Si hay un movimiento que lleva a un estado perdedor para el oponente, el estado actual es ganador (W). Si no, es perdedor (L).
- **Clasificación:** En este juego, un estado es perdedor (L) si el número de palos es divisible por 4; de lo contrario, es ganador (W).
- **Estrategia óptima:** Siempre deja al oponente un número de palos divisible por 4. Si el número inicial es divisible por 4, el oponente tiene la ventaja.

Estado	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
Resultado	L	W	W	W	L	W	W	W	L	W	W	W	L	W	W	W

State graph

Ahora consideremos otro juego de palitos, donde en cada estado `k` es posible eliminar cualquier número `x` de palitos tal que `x` sea menor que `k` y divisor de `k`.



El grafo muestra los estados `1 … 9` del juego como un grafo de estados, donde los nodos representan los estados y las aristas los movimientos permitidos entre ellos.

Por ejemplo:

- En el estado `8`, podemos eliminar `1`, `2` o `4` palitos.
- En el estado `7`, la única opción permitida es eliminar `1` palito.

El estado final en este juego siempre es el estado `1`, que es un **estado perdedor**, porque no hay movimientos válidos.

Tabla de clasificación:

Estado (k)	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Resultado	L	W	L	W	L	W	L	W	L

Sorprendentemente, en este juego:

- Todos los números **pares** son **estados ganadores** (W).
- Todos los números **impares** son **estados perdedores** (L).

Por lo que la estrategia optima es dejar al oponente en un **estado impar**, ya que todos los estados impares son perdedores.

Nim game

El **juego de Nim** es fundamental en teoría de juegos, ya que muchas estrategias se pueden generalizar a partir de este.

Reglas básicas:

- Hay **n montones**, cada uno con cierto número de palos.
- Los jugadores alternan turnos y, en cada turno, deben elegir un montón con palos y retirar **cualquier cantidad de palos** de ese montón.
- El ganador es quien retira el **último palo**.

Estado del juego:

- Representado como $[x_1, x_2, \ldots, x_n]$, donde x_k es el número de palos en el montón k.
- Ejemplo: $[10, 12, 5]$ representa tres montones con 10, 12 y 5 palos respectivamente.
- El estado final siempre es $[0, 0, \ldots, 0]$, que es un **estado perdedor**, ya que no quedan movimientos posibles.

Análisis:

1. **Nim sum (s):**
   - Se calcula como el XOR de los elementos del estado:
     $s = x_1 \oplus x_2 \oplus \ldots \oplus x_n$
   - Un estado es:
     - **Perdedor** si $s = 0$
     - **Ganador** si $s \neq 0$
2. **Estados perdedores:**
   - El estado $[0, 0, \ldots, 0]$ es siempre perdedor ($s = 0$).
   - En cualquier otro estado perdedor, **todo movimiento** lleva a un estado ganador.
3. **Estados ganadores:**
   - Desde un estado ganador, siempre es posible realizar un movimiento que lleve al oponente a un estado perdedor.
   - La regla clave es encontrar un montón k tal que: $x_k \oplus s < x_k$.
     En este caso, se ajusta x_k a $x_k \oplus s$, garantizando un estado perdedor para el oponente.

Ejemplo práctico:

Estado inicial: $[10, 12, 5]$

1. **Nim sum:** $10 \oplus 12 \oplus 5 = 3$
   - Representación binaria:

```
10: 1010
12: 1100
 5: 0101
 ----------
 3: 0011
```

   - Como $s \neq 0$, es un **estado ganador**.
2. **Movimiento óptimo:**
   - La posición más significativa del $s = 3$ (0011) es el segundo bit.
   - El montón 10 (1010) tiene un bit 1 en esa posición.
   - Reducimos el tamaño del montón 10 a $10 \oplus 3 = 9$.

3. **Nuevo estado:** $[9, 12, 5]$:
   - Nim sum: $9 \oplus 12 \oplus 5 = 0$

```
 9: 1001
12: 1100
 5: 0101
 ----------
 0: 0000
```

   - Este es un **estado perdedor**.

Estado	Nim Sum (s)	Resultado	Comentario
$[10, 12, 5]$	$10 \oplus 12 \oplus 5 = 3$	Ganador	Hay un movimiento que lleva al estado perdedor: cambiar 10 a 9.
$[9, 12, 5]$	$9 \oplus 12 \oplus 5 = 0$	Perdedor	Ningún movimiento puede evitar que el oponente gane si juega óptimamente.
$[0, 0, 0]$	0	Perdedor	Estado final del juego, no hay movimientos posibles.

Sprague–Grundy theorem

El teorema de Sprague-Grundy generaliza la estrategia utilizada en el nim a todos los juegos que cumplan los siguientes requisitos:

- Dos jugadores mueven alternativamente.
- El juego consta de estados y los movimientos posibles en un estado no dependen de a quién le toque el turno.
- El juego termina cuando un jugador no puede hacer un movimiento.
- El juego seguramente terminará tarde o temprano.

Los jugadores tienen información completa sobre los estados y los movimientos permitidos y no hay aleatoriedad en el juego.

La idea es calcular para cada estado del juego un número de Grundy que corresponda al número de palitos en un montón de nim. Cuando conocemos los números de Grundy de todos los estados, podemos jugar el juego como el juego del nim.

Grundy number

El número de Grundy para un estado de juego se define como:

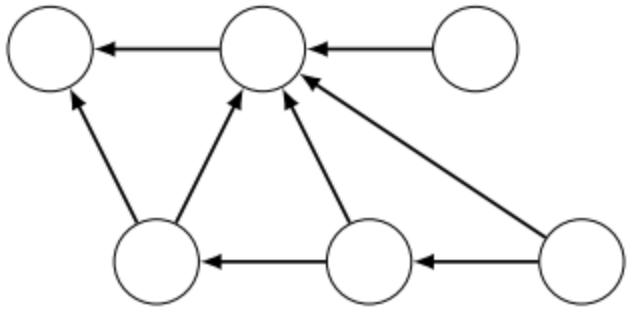
Grundy(state) = `mex({g1, g2, ..., gn})`

Donde:

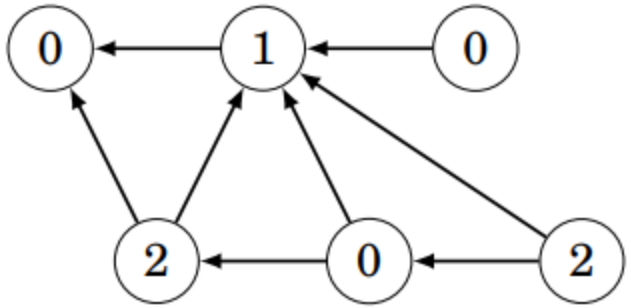
- $g_1, g_2, \ldots, g_n$ son los números de Grundy de los estados a los que se puede mover.
- La función `mex` (Minimum EXcluded) da el menor número no negativo que **no está** en el conjunto.

Ejemplo

- `mex({0, 1, 3}) = 2`, ya que 2 es el menor número no negativo que no pertenece al conjunto, Si no hay movimientos posibles desde un estado (es un estado perdedor), su número de Grundy es 0 `mex(Ø) = 0`



Los números de Grundy son los siguientes:



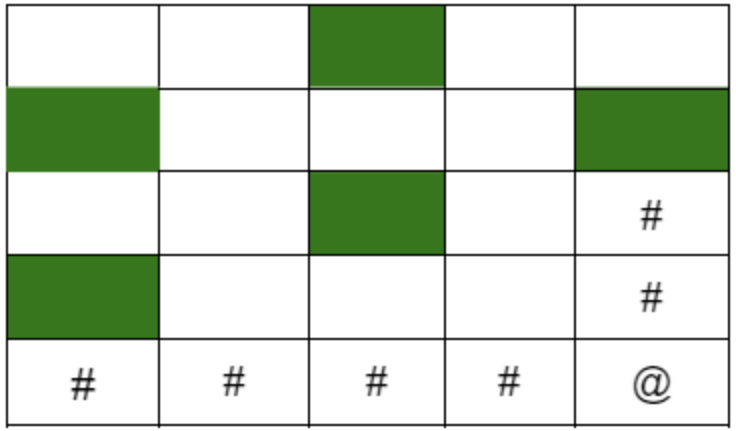
- **Estado perdedor:**
  - El número de Grundy de un **estado perdedor** es siempre **0**.
  - Ejemplo: Si no hay movimientos posibles, como en el estado final de un juego, su número de Grundy es (0).
- **Estado ganador:**
  - Un **estado ganador** tiene un número de Grundy **positivo**.
  - Si el número de Grundy es (x > 0), se puede mover a estados cuyos números de Grundy incluyan (0, 1, ..., x-1).

Ejemplo de Grundy

considere un juego en el que los jugadores mueven una figura en un laberinto.

- Cada cuadrado del laberinto es un piso o una pared.
- En cada turno, el jugador tiene que mover la figura una cierta cantidad de pasos hacia la izquierda o hacia arriba.
- El ganador del juego es el jugador que realiza el último movimiento.

La siguiente imagen muestra un posible estado inicial del juego, donde `@` denota la figura y `#` denota un cuadrado donde se puede mover.



Los estados del juego son todos los cuadrados del suelo del laberinto. En el laberinto anterior, los números de Grundy son los siguientes:

0	1		0	1
	0	1	2	
0	2		1	0
	3	0	4	1
0	4	1	3	2

En el juego del laberinto, cada estado equivale a un montón (heap) en el juego **Nim**, y su número de Grundy determina si es un estado ganador o perdedor. Por ejemplo, el cuadrado inferior derecho, con un número de Grundy de 2, es un estado ganador desde el cual es posible llegar a un estado perdedor y asegurar la victoria. A diferencia de Nim, se puede mover a estados con un número de Grundy mayor, pero el oponente siempre podrá contrarrestar esos movimientos, asegurando que desde un estado perdedor no es posible escapar.

Minimax Algorithm

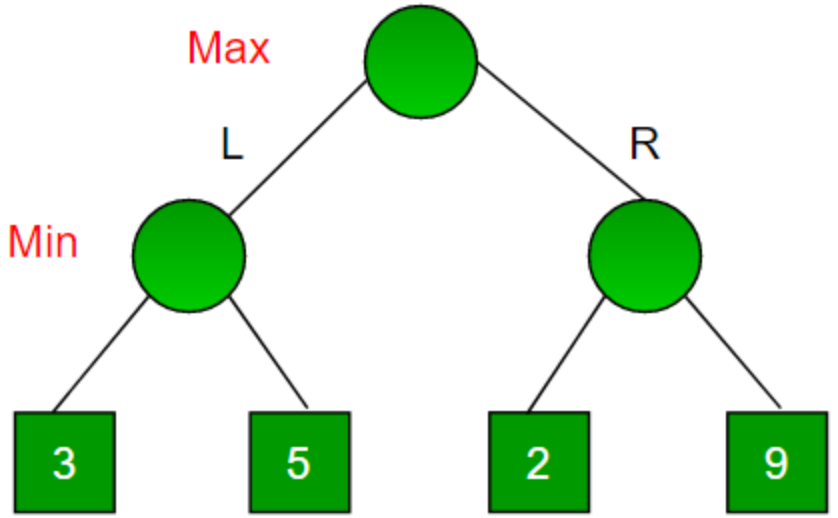
El algoritmo **Minimax** es una técnica de retroceso usada en la teoría de juegos y en la toma de decisiones para encontrar el movimiento óptimo en un juego de dos jugadores. Es ampliamente utilizado en juegos como **Tic-Tac-Toe, Backgammon, Mancala, Chess**, entre otros.

1. **Jugadores:**
   - **Maximizador:** Busca maximizar la puntuación (valores positivos).
   - **Minimizador:** Busca minimizar la puntuación (valores negativos).
2. **Valor del estado del tablero:**
   - Cada estado del tablero tiene un valor asociado, determinado por una **heurística**.
   - **Estado favorable al maximizador:** Valor positivo.
   - **Estado favorable al minimizador:** Valor negativo.
3. **Asunción básica:**
   - Ambos jugadores juegan de manera **óptima**.
4. **Funcionamiento:**
   - Explora todas las posibles jugadas a través de un árbol de decisiones.
   - Retrocede desde las hojas del árbol (estado final) hacia la raíz, tomando decisiones óptimas en cada nivel.

Ejemplo práctico:

Juego con 4 estados finales:

Los caminos para alcanzar estos estados parten de la raíz hacia las 4 hojas de un árbol binario perfecto. Supongamos que eres el jugador maximizador y tienes el primer turno (es decir, estás en la raíz del árbol). Tu oponente, el minimizador, juega en el siguiente nivel. ¿Qué movimiento harías como jugador maximizador, considerando que tu oponente también juega de forma óptima?

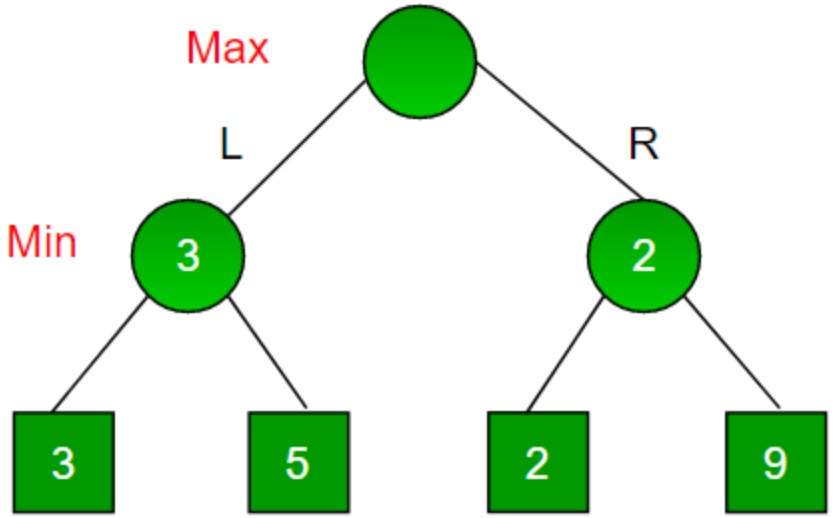


Dado que este es un algoritmo basado en **retroceso (backtracking)**, evalúa todos los movimientos posibles, retrocede y luego toma una decisión.

1. **El maximizador va a la IZQUIERDA:**
    - Ahora es el turno del minimizador.
    - El minimizador tiene una elección entre **3** y **5**.
    - Como minimizador, definitivamente elegirá el menor de ambos, es decir, **3**.
2. **El maximizador va a la DERECHA:**
    - Ahora es el turno del minimizador.
    - El minimizador tiene una elección entre **2** y **9**.
    - Como minimizador, definitivamente elegirá el menor de ambos, es decir, **2**.

Siendo el maximizador, elegirías el valor más alto, que es **3**. Por lo tanto, el movimiento óptimo para el maximizador es ir a la **IZQUIERDA**, y el valor óptimo es **3**.

Ahora, el árbol de juego se ve como se muestra a continuación:



❝ Observaciones:

- Aunque en el subárbol derecho hay un valor más alto (**9**), **el minimizador nunca lo seleccionará**, ya que juega de forma óptima y elegirá el menor valor (**2**).
- El maximizador siempre debe considerar que su oponente jugará de manera ideal.

Codigo en C++

Devuelve el valor óptimo que el maximizador puede obtener. `depth` es la profundidad actual en el árbol de juego. `nodeIndex` es el índice del nodo actual en `scores[]`. `isMax` es verdadero si el movimiento actual es del maximizador, de lo contrario, es falso. `scores[]` almacena las hojas del árbol de juego. `h` es la altura máxima.

```
int minimax(int depth, int nodeIndex, bool isMax, vector<int> scores, int h) {
        // Condición de terminación: se alcanza un nodo hoja
        if (depth == h) return scores[nodeIndex];
        // Si el movimiento actual es del maximizador, encuentra el valor máximo alcanzable
        if (isMax)
                return max(minimax(depth + 1, nodeIndex * 2, false, scores, h),
                                minimax(depth + 1, nodeIndex * 2 + 1, false, scores, h));
        // Si el movimiento actual es del minimizador, encuentra el valor mínimo alcanzable
        else
                return min(minimax(depth + 1, nodeIndex * 2, true, scores, h),
                                minimax(depth + 1, nodeIndex * 2 + 1, true, scores, h));
}
```

función main

```
int main() {
        // la cantidad de elementos debe ser par
        vector<int> scores = {3, 5, 2, 9, 12, 5, 23, 23};
        int n = scores.size();
        int h = log2(n);
        int res = minimax(0, 0, true, scores, h);
        cout << "El valor optimo es: " << res << endl;
        return 0;
}
```

Problema

- **768E** Game of Stones ↑

Referencias

- Akshay, L. (2022). *Minimax Algorithm in Game Theory | Set 1 (Introduction)*. Recuperado de https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-1-introduction/
- Belwariar, R. (2024). *Combinatorial Game Theory | Set 4 (Sprague – Grundy Theorem)*. https://www.geeksforgeeks.org/combinatorial-game-theory-set-4-sprague-grundy-theorem/
- kartik. (2024). *Game Theory*. Recuperado de https://www.geeksforgeeks.org/game-theory/
- Laaksonen, A. (2018). *Competitive Programmer's Handbook*. Recuperado de https://cses.fi/book/book.pdf
- Qi,B & Brebenel, M. (s.f.). *Game Theory*. Recupeardo de https://usaco.guide/adv/game-theory?lang=cpp