

Informática - Curso 2020/2021

Práctica 2 - Juego del NIM

Fecha de entrega: 1 de Febrero (9:00h)

En esta práctica vamos a implementar algunas operaciones del juego del NIM. Ver anexo donde el profesor David de Frutos explica las reglas del juego.

Disponemos de una lista de montones de piezas, cada una con a lo sumo 31, que se inicializará a gusto del usuario en la consola para poder probar lo que se vaya desarrollando. Llamaremos montones a la lista en cuestión. Tendrás que desarrollar las funciones que se describen a continuación. Por supuesto, tienes libertad para implementar otras funciones auxiliares.

Ejercicio 1

Escribir una función llamada `hacer_jugada(mts, m, p)` que tome `p` piezas del montón `m` de las listas de montones `mts`. Se comprobará que la jugada es posible antes de efectuarla. Si la jugada no es posible, la función no hará nada. La función devolverá el estado de los montones tras la jugada.

Ejemplos de ejecución:

```
[In]: montones = [5, 0, 2, 7, 3]
       montones = hacer_jugada(montones, 3, 4)
       print(montones)
```

```
[5, 0, 2, 3, 3]
```

```
[In]: montones = [5, 0, 2, 7, 3]
       montones = hacer_jugada(montones, 0, 6)
       print(montones)
```

```
[5, 0, 2, 7, 3]
```

Ejercicio 2

Escribir una función `cifra_i_base2(n,i)` que calcula la cifra i -ésima del número natural `n` expresado en base 2.

Por ejemplo, el número `n = 39` se transforma en el número `100111` en base 2. Para `i = 1`, el valor devuelto por la función es `1`, mientras que el valor devuelto por la función es `0` en el caso de que `i = 4`.

En realidad no es necesario calcular todas las cifras.

Ejemplos de ejecución:

[In]: cifra_i_base2(0, 0)

[out]: 0

[In]: cifra_i_base2(0, 4)

[out]: 0

[In]: cifra_i_base2(11, 0)

[out]: 1

[In]: cifra_i_base2(11, 1)

[out]: 1

[In]: cifra_i_base2(11, 2)

[out]: 0

[In]: cifra_i_base2(11, 20)

[out]: 0

Ejercicio 3

Escribe una función sumas(mts) que reciba como parámetro de entrada una lista de montones mts y se encargue de hacer las sumas sin llevadas de las cifras en base 2 de los valores en mts. El valor devuelto será el resultado de la suma expresado en base 2 y representado en una lista.

Ten en cuenta que los montones tienen a lo sumo 31 piezas. Un número natural $n \leq 31$ se puede representar en binario con 5 cifras, alguna de las cuales (al "principio" del número) pueden ser 0's.

mts [5, 0, 2, 7, 3]

5 =	00101
0 =	00000
2 =	00010
7 =	00111
3 =	00011
<hr/>	
00011	

Ejemplos de ejecución:

```
[In]: sumas([5, 0, 2, 7, 3])
[Out]: [0, 0, 0, 1, 1]

[In]: sumas([30, 0, 2, 7, 3])
[Out]: [1, 1, 0, 0, 0]

[In]: s = sumas([5, 0, 1, 3, 3])
      print(s)

[0, 0, 1, 0, 0]
```

Ejercicio 4

Escribe una función llamada `terminada(mts)` que compruebe si la partida con montones `mts` ha terminado. En caso de que la partida haya terminado, la función devuelva `True`. Decimos que una partida ha terminado si todos los montones están vacíos, es decir no hay piezas en ellos.

Ejemplo de ejecución:

```
[In]: terminada([30, 0, 2, 7, 3])
[Out]: False

[In]: terminada([0, 0, 0, 0, 3])
[Out]: False

[In]: terminada([0, 0, 0, 0])
[Out]: True
```

Ejercicio 5

Escribe una función llamada `ganadora(mts)` que compruebe si la posición de la partida con montones `mts` es ganadora para el jugador que le toque jugar. Una partida es ganadora si la suma en base 2 de todos los montones no es 0.

(Ver Anexo donde se explica el juego del Nim).

Ejemplo de ejecución

```
[In]: ganadora([5,2,7,3])
[Out]: True

[In]: ganadora([5,1,7,3])
[Out]: False
```

Ejercicio 6

Escribe una función llamada `jugar_ganadora(mts)` que partiendo de una posición ganadora `mts`, devuelva una jugada ganadora (para el jugador que le toque jugar). La función devuelve una tupla con tres elementos (`m`, `h`, `c`) (del montón `m` con `h` piezas, tomo `c` piezas).

(Ver Anexo donde se explica el juego del Nim).

Ejemplo de ejecución (hay varias posibilidades):

```
[In]: jugada_ganadora([5, 0, 1, 3, 3])
[Out]: (0, 5, 4)
```

Ejercicio 7

Escribe una función llamada `jugar_perdedora(mts)` que partiendo de una posición perdedora `mts`, devuelva una jugada cualquiera. La función devuelve una tupla con tres elementos (`m`, `h`, `c`) (del montón `m` con `h` piezas, tomo `c` piezas).

(Ver Anexo donde se explica el juego del Nim).

Ejemplo de ejecución (hay varias posibilidades):

```
[In]: jugada_perdedora([5,1,7,3])
[Out]: (0, 5, 1)
```

Ejercicio 8

Escribe una función llamada `juega_maquina(mts)` que cuando le toque jugar a la máquina, seleccione y haga una jugada inteligente. La función devolverá el estado de los montones tras la jugada.

(Ver Anexo donde se explica el juego del Nim).

Ejemplo de ejecución (hay varias posibilidades):

```
[In]: juega_maquina([5,0,2,7,3])
[Out]: [5,0,1,7,3]
```

El juego del NIM

1) Descripción del juego

El juego del NIM es un juego muy sencillo al que sin embargo resulta difícil "jugar bien" si no se tienen ciertos conocimientos matemáticos relativos a la representación en base 2 de los números (naturales).

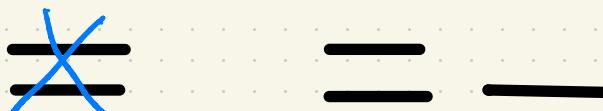
El tablero del juego viene dado por una determinada serie de montones con piezas iguales. Por ejemplo :





Elijo un moutón y elimino
un cierto número de piezas (al
menos una)

Mi rival hace lo mismo



Vuelvo a jugar yo

y ahora él



Jueg yo

y ahora él

~~X~~

Con la última pieza yo

¡He ganado!

2) ¿Cómo tengo que jugar para ganar seguro?

Ante todo tenemos que "asumir" que no siempre podré hacerlo, pues si yo puedo hacerlo es porque puedo hacer una jugada ganadora que deja a mi rival en una posición perdedora. Así que si fuera yo quien tuviera que jugar en esa posición, tampoco podría hacer nada por ganar... de momento.

De momento, porque para ir a ganar seguro cuando estoy en una posición ganadora tengo que elegir acertadamente mi jugada.

De hecho, si acabo de ganar la partida anterior ha sido sólo porque mi rival no ha salido a jugar tras mi primera jugada:



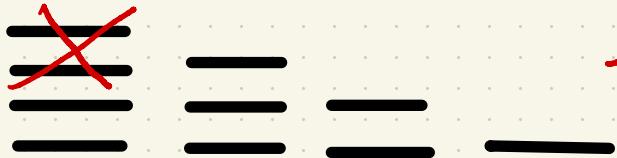
Elijo un montón y elimino
un cierto número de piezas

Mi rival hace lo mismo que antes
¡Jugando MAL!



Elijo un montón y elimino
un cierto número de piezas

Mi rival **debería haber**
jugado ASI o ...



Elijo un moutón y elimino
un cierto número de piezas

O también podría haber
hecho esto



Para **GANAR** aquí **TENGO QUE**
HACER esto

He dejado a mi rival en
la misma posición **perdedora**
en la que él me dejaba arriba

- 3) ¿ Pero cómo he de jugar en general ?
¿ Cuáles son todas las posiciones **ganadoras / perdedoras** ?

ALGORITMO para detectar posiciones GANADORAS / PERDEDORAS

Obviamente lo único que importa en una posición es cuántos montones de cada tamaño tenemos. Representaremos cada posición con la notación

$t_1^{k_1} \dots t_p^{k_p}$, donde los t_i 's son los tamaños **distintos** que tenemos , y los $k_i > 0$ la **cantidad** de montones de tamaño t_i que tenemos . Así , la situación inicial que teníamos era

$4^1 3^1 2^1 1^1$, la alcanzada tras mi primera jugada era

$3^1 2^2 1^1$, y la **perdedora** que tanto yo como mi rival

le hemos dejado "al otro", es $3^1 2^1 1^1$.

- Posiciones **ganadoras / perdedoras "triviales"**

- 1^n con n IMPAR es **ganadora**:

(En los siguientes jugados nos alternamos teniendo que coger **1 pieza** y termina haciéndolo el **mismo** que empieza)

- 1^n con n IMPAR es **perdedora**:

(Ahora termina el que **NO empezó**)

En una posición **trivial** sólo se puede ir jugando "de una forma", por lo que el **final** está **predeterminado**, hagamos lo que hagamos.

- Posiciones ganadoras / perdedoras NO triviales
- i) Escribimos en base 2 los tamaños t_i repetidos k_i veces :

$4^1 3^1 2^1 1^1$

$1 \ 0 \ 0$

$3^1 2^2 1^1$

$1 \ 1$

$1 \ 1$

$1 \ 0$

$1 \ 0$

$1 \ 0$

1

1

$3^1 2^1 1^1$

$1 \ 1$

$1 \ 0$

1

ii) Sumamos en base 2 ¡ pero sin hacer "llevados" !

$$4^1 3^1 2^1 1^1$$

$$\begin{array}{r} 100 \\ 11 \\ 10 \\ 1 \end{array}$$

$$3^1 2^1 1^1$$

$$\begin{array}{r} 11 \\ 10 \\ 10 \\ 1 \end{array}$$

$$3^1 2^1 1^1$$

$$11$$

$$\begin{array}{r} 100 \\ \hline \end{array}$$

$$10$$

$$\begin{array}{r} 10 \\ \hline \end{array}$$

$$1$$

$$\hline$$

$$00$$

Las posiciones **perdedoras** son las que suman "**todo ceros**" ; las demás son por tanto todas **ganadoras**.

iii) Realizando los jugados que indicamos antes: consecuencias.

4¹3¹2¹1¹

$3^1 2^1 1^1$

11

10

1

¡ pierdo !

00

Las posiciones **perdedoras** son las que suman "**todo ceros**" ; las demás son por tanto todas **ganadoras**.

- ¿ Cómo jugar en una posición **GANADORA** (no trivial) ?
¡ Convirtiéndola en **perdedora** ! (trivial o no trivial)
- i) Tomamos el valor numérico de la "suma sin llevados" calculada (100 en el 1º ejemplo ; 10 en el 2º)
 - ii) Escogemos **cualquier** montón que tenga un 1 en la posición más a la izqda en la que la suma tiene un 1.
(Evidentemente , tiene que haber alguno , pues todo 0's sumaría 0)
 - iii) Y ahora partiendo de esa posición (incluida) , y hacia la drcha , numerándolos con **0** más a la drcha ,
por cada 1 de la sum , con 1 también en esa
posición en el montón **QUITO** **2^{pos}** piezas , y
con 0 en esa posición en el montón **REPONGO** **2^{pos}**
piezas .

- ¿ Cómo jugar en una posición **GANADORA** (trivial) ?

En realidad **NO** es necesario fijarse en si la posición es o no **trivial**. Las posiciones ganadoras triviales ¡ también suman $1 \neq 0$!, así que al aplicar el algoritmo general seguimos cogiendo un montón cualquiera y tomando 1 pieza (¡ lo único que podemos hacer !) dejando claramente una posición **perdedora trivial** como vimos antes.

- ¿Cómo jugar en una posición **PERDEDORA** (no trivial)?
Hagamos lo que hagamos vamos a dejarle al rival una posición **ganadora**, por lo que si "sabe jugar" o acierta "por casualidad" o "intuición" con una jugada "ganadora" cada vez que le toque jugar en lo sucesivo "estamos perdidos". Para intentar "prevenir" lo mejor posible "lo segundo" lo aconsejable es hacer una jugada que "simplifique poco" (o sea cogemos **pocas** piezas), pero también es deseable dejar en lo posible un montón de un tamaño del que hasta ahora habría una cantidad **PAR** de montones, y elegir también un tamaño t_i con t_i **PAR**. La razón es que cada **pareja** de montones iguales **SUMA todo ceros**, lo que simplifica el "cálculo mental" de la suma (de un jugador "humano") -

- Elección entre varios jugadores **ganadores** (cuando los haya)

ya hemos dicho que "dería igual" cuál hacer de cara a ganar. Pero si vemos a jugar "muchas veces" con un jugador "que de momento no sabe jugar bien" lo aceptable sería "**verificar**" cuando se juega. Incluso, si no nos importa perder en ocasiones, convendría "**arraigarnos**" a jugar mal al principio (no siguiendo la estrategia adrede), para que así nuestro rival "no pueda saber" cómo hay que jugar a base de fijarse en cómo lo hacemos nosotros (que si queremos "terminar" ganando empejaremos a jugar bien cuando nos peseza "que ya queda poco" para acabar, o terminaremos perdiendo por "pascnos de listos").

ii) Aplicación del algoritmo en los ejemplos anteriores y otros

$$4^1 3^1 2^1 1^1$$

$$3^1 2^1 1^1$$

$$\begin{array}{r} 11 \\ 10 \\ 1 \\ \hline 00 \end{array}$$

$$\begin{array}{r} 100 \\ 11 \\ 10 \\ 1 \\ \hline \end{array}$$

$$100$$

$$3^1 2^1 1^1$$



$$2^2 1^2$$

$$\begin{array}{r} 10 \\ 10 \\ 1 \\ 1 \\ \hline 00 \end{array}$$

$$\begin{array}{r} 11 \\ 10 \\ 10 \\ 1 \\ \hline \end{array}$$



$6^1 \ 5^1 \ 4^1 \ 3^1 \ 2^1 \ 1^1$

110

10

100

11

10

1

000

~~$\rule{1cm}{0.4pt}$~~

$$\begin{array}{r} -2^2 + 2^1 - 2^0 \\ \hline -3 \end{array}$$

110

101

100

11

10

1

111

~~$\rule{1cm}{0.4pt}$~~

$$\begin{array}{r} -2^2 - 2^1 + 2^0 \\ \hline -5 \end{array}$$

1
 101

100

11

10

1

000

110
 101

11

11

10

1

000

$$\begin{array}{r} -2^2 + 2^4 + 2^0 \\ \hline -1 \end{array}$$

