UNIVERSIDAD DE CASTILLA-LA MANCHA

ESCUELA SUPERIOR DE INFORMÁTICA

GRUPO:B2

PRÁCTICA 2. ALGORITMO DIVIDE Y VENCERÁS: AMONG US

Trabajo realizado por (grado de participación):

Miguel de las Heras Fuentes → 50%

Isaac González del Pozo → 50%

Asignatura: Metodología de la Programación

Grupo: B2

Titulación: Grado en Ingeniería Informática

Fecha: 12/03/2021

Contenido

1. ESTRATEGIA UTILIZADA

En este apartado expondremos la estrategia que hemos utilizado para encontrar al impostor en base al estudio del atributo “ira” de todos los participantes del juego.

Para resolver este problema hemos utilizado un algoritmo divide y vencerás que estará compuesto por dos métodos. El método principal recursivo llamado “buscarImpostor” y un método auxiliar llamado “medidorIra”.

La función del método medidorIra será decir si en el intervalo que se le ha pasado como parámetro está el impostor o no. Para ello, se le pasarán como parámetros la lista principal de jugadores(lista), una posición inicial(li) que corresponde a la primera posición del intervalo que vamos a analizar, y una posición final(ls) que corresponde a la última posición del intervalo que vamos a analizar.

Para comprobar si el impostor está dentro del intervalo, utilizaremos un bucle for que ira recorriendo todas las posiciones del array lista desde la posición li hasta la posición ls, e iremos comparando el atributo ira de la posición i con el valor 2 mediante “lista[i].getIra()==2”. En caso de que esta comprobación se cumpla se retornará el valor 2. En caso contrario, iremos avanzando posiciones hasta llegar a la posición ls y retornar el valor 1.

```
public static int medidorIra(Jugador[] lista, int li, int ls) {
        int ira=1;

        for (int i = li; i <= ls; i++) {
                if (lista[i].getIra()==2) {
                        ira=2;
                        return ira;
                }
        }
        return ira;
}
```

El método principal buscarImpostor, nos permitirá ir dividiendo la lista de jugadores de dos en dos hasta que al final quede un intervalo de una posición que será la del impostor. Para ello, primero deberemos comprobar si el número de posiciones es par o impar. Si es impar, dividiremos el intervalo en parte izquierda y parte derecha dejando la posición del medio sola. Cada parte será pasada como parámetro en las llamadas al método medidorIra para comprobar en que parte está el impostor. el intervalo que devuelva el valor 2 de la llamada, será pasado como parámetro en una llamada recursiva al método BuscarImpostor y repetiremos el proceso. En caso de que ninguna de las dos llamadas al método medidorIra devuelva el valor 2, significará que el impostor será el jugador del medio, de manera que terminaremos el método y devolveremos esa posición.

En el caso de que el número de jugadores que se le pasan como parámetro al método buscarImpostor sea par, llevaremos a cabo el mismo proceso, pero en lugar de dejar al jugador del medio solo, lo incluiremos en la primera mitad.

```
public static int buscarImpostor(Jugador[]lista, int li, int ls) {
		int pos=-1;
		if (li==ls) {
			pos=li;
		}else{
			int mitad= (li+ls)/2;
			if ((ls-li)%2!=0) {
				int iraIzq= medidorIra(lista, li, mitad-1);
				int iraDrch= medidorIra(lista, mitad+1, ls);
				if (iraIzq==2) {
					pos=buscarImpostor(lista,li,mitad-1);
				}else if(iraDrch==2)
					pos=buscarImpostor(lista,mitad+1,ls);
				else
					pos=mitad;
			}else {
				int iraIzq= medidorIra(lista, li, mitad);
				int iraDrch= medidorIra(lista, mitad+1, ls);
				if (iraIzq==2) {
					pos=buscarImpostor(lista, li, mitad);
				}else
					pos=buscarImpostor(lista,mitad+1,ls);
			}
		}
		return pos;
	}
```

2. COMPLEJIDAD TEÓRICA: ALGORITMO BUSCARIMPOSTOR

$T(n) = t\left(\frac{n}{2}\right) + 1$

$n = 2^m \quad m = \log n$

$t(2^m) = t\left(\frac{2^m}{2}\right) + 1$

$t_m - t_{m-1} = 1$

$X^m - X^{m-1} = 1$

$X^{m-1}(X-1) = 1 \qquad b^m p(m)^d$

$b = 1^m \quad d = 0$

$(X-1)(X-1) = 0 \qquad r = 1(doble)$

$t_m = C_1(1)^m + C_2 m(1)^m$

Sustituimos

$C_2 * \log_2 n$

Pertenece al orden $O(log\ n)$

3. COMPLEJIDAD EMPÍRICA DE LOS RESULTADOS

- 3 jugadores

```
Introduzca el número de participantes(minimo 3)
3
Lista de participantes:

Jugador 0 experiencia=3, numTareas=2, ira=1
Jugador 1 experiencia=1, numTareas=0, ira=2
Jugador 2 experiencia=3, numTareas=0, ira=1

El impostor es el Jugador 1 experiencia=1, numTareas=0, ira=2
El tiempo que ha tardado en encontrar al impostor es: 70800 nanosegundos
El ganador es el jugador: Jugador 2 experiencia=3, numTareas=0, ira=1
```

- Con 10 jugadores

```
Introduzca el número de participantes(minimo 3)
10
Lista de participantes:

Jugador 0 experiencia=4, numTareas=4, ira=1
Jugador 1 experiencia=0, numTareas=1, ira=1
Jugador 2 experiencia=4, numTareas=6, ira=1
Jugador 3 experiencia=0, numTareas=7, ira=1
Jugador 4 experiencia=3, numTareas=4, ira=1
Jugador 5 experiencia=2, numTareas=0, ira=1
Jugador 6 experiencia=2, numTareas=5, ira=1
Jugador 7 experiencia=0, numTareas=1, ira=1
Jugador 8 experiencia=4, numTareas=1, ira=1
Jugador 9 experiencia=2, numTareas=1, ira=2

El impostor es el Jugador 9 experiencia=2, numTareas=1, ira=2
El tiempo que ha tardado en encontrar al impostor es: 66600 nanosegundos
El ganador es el Jugador 0 experiencia=4, numTareas=4, ira=1
```

- Con 100 jugadores

```
Jugador 95 experiencia=4, numTareas=2, ira=1
Jugador 96 experiencia=4, numTareas=2, ira=1
Jugador 97 experiencia=4, numTareas=5, ira=1
Jugador 98 experiencia=4, numTareas=2, ira=1
Jugador 99 experiencia=0, numTareas=2, ira=1

El impostor es el Jugador 0 experiencia=4, numTareas=2, ira=2
El tiempo que ha tardado en encontrar al impostor es: 165800 nanosegundos
El ganador es el Jugador 1 experiencia=1, numTareas=6, ira=1
```

- Con 1 000 jugadores

```
Jugador 996 experiencia=1, numTareas=2, ira=1
Jugador 997 experiencia=0, numTareas=0, ira=1
Jugador 998 experiencia=0, numTareas=7, ira=1
Jugador 999 experiencia=2, numTareas=0, ira=1

El impostor es el Jugador 555 experiencia=2, numTareas=5, ira=2
El tiempo que ha tardado en encontrar al impostor es: 140400 nanosegundos
El ganador es el impostor: Jugador 555 experiencia=2, numTareas=5, ira=2
```

- Con 10 000 jugadores

```
Jugador 9996 experiencia=2, numTareas=4, ira=1
Jugador 9997 experiencia=4, numTareas=5, ira=1
Jugador 9998 experiencia=2, numTareas=6, ira=1
Jugador 9999 experiencia=1, numTareas=2, ira=1

El impostor es el Jugador 7632 experiencia=4, numTareas=6, ira=2
El tiempo que ha tardado en encontrar al impostor es: 873200 nanosegundos
El ganador es el Jugador 7633 experiencia=4, numTareas=7, ira=1
```

- Con 100 000 jugadores

```
Jugador 99997 experiencia=3, numTareas=1, ira=1
Jugador 99998 experiencia=1, numTareas=2, ira=1
Jugador 99999 experiencia=0, numTareas=7, ira=1

El impostor es el Jugador 78601 experiencia=0, numTareas=5, ira=2
El tiempo que ha tardado en encontrar al impostor es: 4059800 nanosegundos
El ganador es el impostor: Jugador 78601 experiencia=0, numTareas=5, ira=2
```

- Con 1 000 000 jugadores

```
Jugador 999997 experiencia=3, numTareas=7, ira=1
Jugador 999998 experiencia=0, numTareas=4, ira=1
Jugador 999999 experiencia=1, numTareas=5, ira=1

El impostor es el Jugador 930274 experiencia=3, numTareas=2, ira=2
El tiempo que ha tardado en encontrar al impostor es: 14901000 nanosegundos
El ganador es el Jugador 930275 experiencia=2, numTareas=6, ira=1
```

- Con 10 000 000 jugadores

```
Jugador 9999997 experiencia=1, numTareas=0, ira=1
Jugador 9999998 experiencia=1, numTareas=5, ira=1
Jugador 9999999 experiencia=3, numTareas=0, ira=1

El impostor es el Jugador 3033780 experiencia=1, numTareas=0, ira=2
El tiempo que ha tardado en encontrar al impostor es: 85603300 nanosegundos
El ganador es el Jugador 3033781 experiencia=4, numTareas=4, ira=1
```

- Con 1 000 000 000 jugadores

```
Introduzca el número de participantes(minimo 3)
1000000000
Exception in thread "main" java.lang.OutOfMemoryError: Java heap space
        at main.main(main.java:44)
```

No se puede ejecutar ya que son demasiados jugadores, tal y como dice el mensaje de excepción OutOfMemoryError que significa que se ha ocupado el máximo tamaño de memoria.

Como podemos observar en nuestros resultados, vemos que el tiempo empleado no es directamente proporcional a los jugadores introducidos porque como hemos calculado en el apartado anterior, la complejidad teórica del algoritmo es logarítmica, no lineal.