Practica 2

Algoritmo Divide y Vencerás

18-03-2021
Metodología e la programación

Alberto García Aparicio
Laura Morales Caro
B1

INDICE

1.Introducción

En esta práctica se pretende hacer un programa que simule una versión más simple del conocido juego "Among Us". El juego trata de una serie de tripulantes con entre 0 y 5 de experiencia y entre 0 y 8 tareas realizadas. Entre los tripulantes hay un impostor también con experiencia y tareas, lo que le diferencia de los tripulantes es que el impostor tiene 2 de ira mientras que el resto tiene 1.

Nuestro trabajo es encontrar al impostor de entre todos los tripulantes por lo que para ello utilizaremos un medidor de ira capaz de medir la ira total de dos grupos. También tenemos que intentar utilizar el medidor de ira lo menos posible.

Una vez hayamos encontrado al impostor tenemos que comprobar el ganador final comparando las tareas y la experiencia con el tripulante de la derecha del impostor. Ganará el que mayor experiencia tenga, en caso de que sea la misma, ganará el que mayor número de tareas realizadas tenga. Si estas dos fueran iguales se daría por ganador al tripulante.

2.Comentarios del código

En este programa tendremos dos clases. La clase **Jugador** tendrá los atributos ira, experiencia y misiones. También tendremos los setters y getters correspondientes de cada atributo.

```
public class Jugador extends Thread {

	int ira;
	int experiencia;
	int misiones;

	public Jugador(int ira, int experiencia, int misiones) {
		super();
		this.ira = ira;
		this.experiencia = experiencia;
		this.misiones = misiones;
	}

	public int getIra() {
		return ira;
	}
	public void setIra(int ira) {
		this.ira = ira;
	}
	public int getExperiencia() {
		return experiencia;
	}
	public void setExperiencia(int experiencia) {
		this.experiencia = experiencia;
	}
	public int getMisiones() {
		return misiones;
	}
	public void setMisiones(int misiones) {
		this.misiones = misiones;
	}
}
```

La clase **AmongUs** se encargará de generar los jugadores, seleccionar a un jugador como impostor de manera aleatoria y encontrar a ese impostor de entre todos los tripulantes. Una vez tengamos al impostor comprobaremos el ganador final.

- ## Método generarJugadores

```
public static int generarJugadores() {
        int cantidadJugadores;
        System.out.println("Introduce la cantidad de jugadores:");
        cantidadJugadores = teclado.nextInt();

        vectorJugadores = new Jugador[cantidadJugadores];

        for (int i = 0; i < vectorJugadores.length; i++) {
                int experiencia = new Random().nextInt(6);
                int misiones = new Random().nextInt(9);
                Jugador jugador = new Jugador(1, experiencia,
misiones);
                vectorJugadores[i] = jugador;
        }
        int impostor = new Random().nextInt(cantidadJugadores);
        vectorJugadores[impostor].setIra(2);
        return cantidadJugadores;
}
```

En este método lo que haremos será pedir la cantidad de jugadores que van a participar, a todos les daremos una experiencia de entre 0 y 5 y unas misiones de entre 0 y 8 y meteremos a todos los jugadores en nuestro vectorJugadores. Una vez tengamos todos nuestros jugadores cogeremos a uno aleatoriamente y le actualizaremos la ira a 2. Después se devuelve la cantidad de jugadores que se había pedido anteriormente.

- ## Método medidorIra

```
public static int medidorIra(int iz, int der) {
        int contadorIra = 0;

        for (int i = iz; i <= der; i++) {
                contadorIra += vectorJugadores[i].getIra();
        }
        return contadorIra;
}
```

En este método iremos sumando las iras de los jugadores y las guardaremos en el contadorIra, que es lo variable que devolverá el metodo.

- **Método recursividad**

```
public static int recursividad(int iz, int der) {
	int iralz, iraDer, posicionTraidor = 0;

	if (iz == der) {
		posicionTraidor = iz;
		System.out.println("Solo queda una posición donde puede
	estar el impostor");
	} else {
		int mitad = (iz + der) / 2;
		iraDer = medidorIra(mitad + 1, der);
		// PAR
		if (((der + 1) - iz) % 2 != 0) {
			iralz = medidorIra(iz, mitad - 1);
			contador++;
			if (iralz > iraDer)
				posicionTraidor = recursividad(iz, mitad - 1);
			else if (iralz < iraDer)
				posicionTraidor = recursividad(mitad + 1, der);
			else{
				posicionTraidor = mitad;
				System.out.println("El impostor esta en
medio");
			}
		}
		// IMPAR
		else {
			iralz = medidorIra(iz, mitad);
			contador++;
			if (iralz > iraDer)
				posicionTraidor = recursividad(iz, mitad);
			else if (iralz < iraDer)
				posicionTraidor = recursividad(mitad + 1, der);
		}
	}
	return posicionTraidor;
}
```

El impostor sabemos que esta entre el rango 0 y n por lo que pondremos el inicio a 0 y el final a n para ir buscando al impostor mediante divisiones entre dos. Si la longitud de jugadores es impar se mide la izquierda y la derecha sin contar el del medio, en el caso de que la ira sea igual en los dos lados el impostor es el del medio, si no es el caso seleccionaremos la parte que tenga más ira para seguir haciendo la recursividad. En caso de ser par se mide simplemente los dos lados y se va al que tenga más ira para seguir la recursividad.

- **Método posicionContrincante**

```
public static int posicionContrincante(int impostorUbi) {
		int contrincante;

		if (impostorUbi == vectorJugadores.length - 1) {
			contrincante = 0;
		} else {
			contrincante = impostorUbi + 1;
		}
		return contrincante;
}
```

En este método buscamos la posición del tripulante que será quien compararemos al impostor. Si el impostor está en la última posición del array, el contrincante será el primer tripulante del array. En caso contrario, el contrincante será el tripulante de la derecha del impostor.

- **Método ganador**

```
public static void ganador(int impostorUbi, int contrincante) {

		if (vectorJugadores[impostorUbi].getMisiones() >
vectorJugadores[contrincante].getMisiones()) {
			System.out.println("Gana el impostor");
		} else if (vectorJugadores[impostorUbi].getMisiones() <
vectorJugadores[contrincante].getMisiones()) {
			System.out.println("Gana la tripulación");
		} else {
			if (vectorJugadores[impostorUbi].getExperiencia() >
vectorJugadores[contrincante].getExperiencia()) {
				System.out.println("Gana el impostor");
			} else if
(vectorJugadores[impostorUbi].getExperiencia() <
vectorJugadores[contrincante].getExperiencia()) {
				System.out.println("Gana la tripulación");
			} else {
				System.out.println("Gana la tripulación");
}
```

A este método le pasamos la posición del impostor en el vector Jugadores y la posición del contrincante. Obtenemos las misiones del impostor con vectorJugadores[impostorUbi].getMisiones() y las del contrincante con vectorJugadores[contrincante].getMisiones(), si el impostor tiene más misiones que el tripulante, ganará el impostor. En caso contrario, ganará la tripulación. Si el impostor y el tripulante tienen el mismo número de misiones se mirará la experiencia y se aplicara la misma regla, gana quien tiene más

experiencia. Si tienen la misma experiencia le daremos la victoria a la tripulación.

- ## Método imprimirInformacionPosTareasExp

```
public static void imprimirInformacionPosTareasExp(int impostorUbi, int
contrincante, long tiempo) {
		System.out.println("\nPosición en el que se encuentra el impostor: "
+ impostorUbi);
		System.out.println("Numero de tareas realizadas por impostor: " +
vectorJugadores[impostorUbi].getMisiones());
		System.out.println("Cantidad de experiencia del impostor: " +
vectorJugadores[impostorUbi].getExperiencia() + "\n");
		System.out.println("Posición en el que se encuentra el tripulante: "
+ contrincante);
		System.out.println("Numero de tareas realizadas por tripulante: " +
vectorJugadores[contrincante].getMisiones());
		System.out.println("Cantidad de experiencia del tripulante: " +
vectorJugadores[contrincante].getExperiencia() + "\n");
		System.out.println("El número de veces que se ha utilizado el
medidor de ira es: " + contador);
		System.out.println("Tiempo que se ha tardado en encontrar al
impostor en nanosegundos: " + tiempo + "\n");
}
```

Este método servirá para imprimir toda la información relacionada con el impostor y su contrincante, es decir, su posición, tareas, misiones, y experiencia.

3.Complejidad de los algoritmos

Vamos a utilizar la siguiente fórmula para calcular la complejidad de nuestro programa: T(n)=aT(nb) +g(n) -> a≥1, b>1.

- n es la cantidad de jugadores
- a es el número de subproblemas en la recursión
- n/b es el tamaño de cada subproblema (todos tienen el mismo tamaño)
- g(n) es el coste del trabajo hecho fuera de las llamadas recursivas: n^k

En nuestro programa a = 1 ya que se hace una llamada. b = 2 ya que dividimos en mitades. Si consideramos la operación de pesar como unitaria g(n)=O(1) entonces k=0. Entones:

$$a=b^k \Rightarrow O(n^0 \log n) \Rightarrow O(\log n)$$

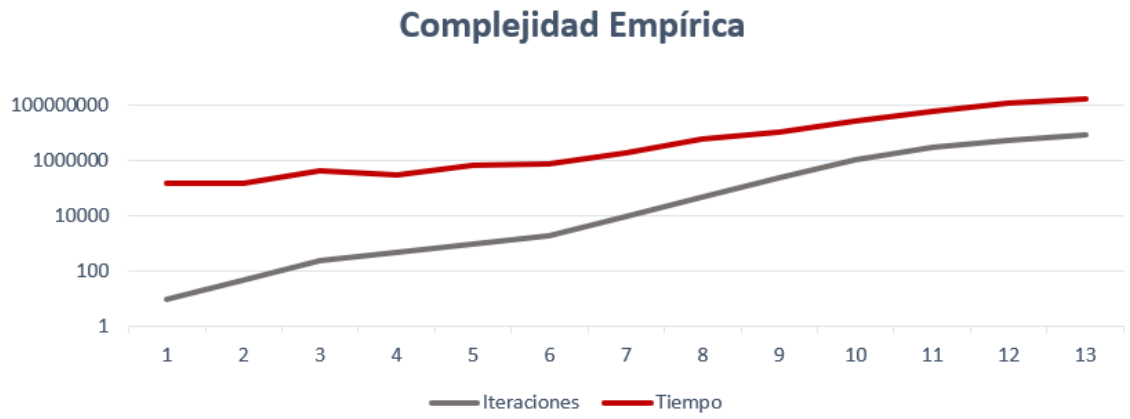
Dentro de esta llamada recursiva tenemos la llamada a otro metodo “medidorIra” el cual se realiza dos veces en cada llamada al metodo anterior, y es un metodo que tiene una complejidad es de O(n), donde esta es $n/2^i$.

$$T(1)=n/2$$
$$T(2)=n/4$$
$$T(3)=n/8$$
$$T(4)=n/16$$
$$T(i)=n/2^i$$

Con lo cual la formula en su totalidad se quedara como una de complejidad O(n*log(n))

4.Complejidad Empírica

Iteraciones	Tiempo
10	154700
50	157000
250	404800
500	302900
1000	692200
2000	771800
10000	1947900
50000	6067600
250000	11053500
1000000	26039300
3000000	56416200
5500000	113873200
8000000	171986000



5.Bibliografía

https://www.fdi.ucm.es/profesor/gmendez/docs/edi0910/_02-Recursion.pdf
https://elvex.ugr.es/decsai/algorithms/slides/2%20Eficiencia.pdf
https://www.dit.upm.es/~pepe/doc/adsw/tema1/Complejidad.pdf