Partida de póker

Tenemos N jugadores (numerados desde 0 hasta N-1) con un fajo de K billetes cada uno. Todos tienen billetes auténticos, pero el jugador 0 es un tahúr que juega con billetes falsos.

Nos proporcionan una lista de la forma:

```
0 1 10
1 2 15
2 0 2
1 2 20
1 0 2
2 0 5
```

Cada línea significa que el primer jugador ha pagado al segundo la cantidad que aparece en tercera posición. Es decir, la primera línea indica que el jugador 0 ha pagado al jugador 1 10 billetes. Se entiende que cada jugador, cuando paga, coge los billetes de encima de su fajo de billetes y, cuando cobra, los coloca también encima.

Se trata de hacer un programa que lea la lista de pagos, simule todo el juego, y termine devolviendo la cantidad de billetes auténticos que ha ganado el jugador 0.


```
public class Juego {

    public int jugar(int nJugadores,
                     int nBilletesInicial,
                     ArrayList<Pago> pagos)
    // Precondición:  2 <=  nJugadores <= 20
    //                nBilletesInicial >= 1
    //                Todos los pagos de la lista “pagos” son correctos,
    //                 es decir, un jugador siempre tiene los billetes
    //                 necesarios para realizar sus pagos
    // Postcondición: el resultado es el número de billetes verdaderos
    //                 del jugador 0 al acabar la partida

}

public class Pago {
    int pagador;
    int cobrador;
    int cantidad;
}
```

Se pide:

1. Encontrar la estructura de datos adecuada para simular la partida.
2. Realizar el diseño e implementación del algoritmo.
3. Complejidad.
4. Código del programa.

Solución

1. Una estructura de datos adecuada para este problema es un array de pilas de booleanos. El
elemento i-ésimo del array se corresponde con el fajo de billetes del i-ésimo jugador. En este caso concreto el array contiene 3 elementos.
2. El diseño del algoritmo es:
a) Incializar las pilas de booleanos
b) Simular los pagos que se encuentran en el fichero
c) Contar los billetes auténticos del jugador A
a) Por cada jugador hacer
Crear pila vacía
Empilar 100 billetes ("**false**" para A, "**true**" para B y C)
b) Mientras no se termine de leer el fichero hacer
leer **j1**, leer **j2,** leer **cant**
empilar en la pila de **j2** la cantidad de billetes **cant** que se desempilan de la pila de **j1**
c) **cont** a 0
Mientras haya billetes en la pila de A hacer
si el billete es "**true**" aumentar **cont**
desempilar de la pila de A
3. La complejidad del algoritmo depende del número de lineas (número de pagos) que contenga el
fichero. Si N es el número de líneas del fichero, el algoritmo es de O(N). En este caso concreto,
el número de jugadores es fijo y la cantidad de billetes que manejan también.
4. Codificación en Java

```java
public class PokerPilas {

   public static int conversion(String c) {
     if (c.equals("A")) return 0;
     else if (c.equals("B")) return 1;
     else return 2;
   }

   public static void main(String[] args){
     PilaSimpleLinkedList<Boolean>[] jugadores =
                                    new PilaSimpleLinkedList[3];
     // Inicializar las colas
     for (int i = 0; i < 3; i++) {
       jugadores[i] = new PilaSimpleLinkedList<Boolean>();
       for (int j = 1; j <= 100; j++) {
         jugadores[i].push((i!=0));
       }
     }

     // leer de fichero
     Scanner sa = new Scanner(new File("pagos.txt"));
     while (sa.hasNext()) {
       int j1 = conversion(sa.next());
       int j2 = conversion(sa.next());
       int cant = sa.nextInt();
       for (int i = 1; i <= cant; i++)
         jugadores[j2].push(jugadores[j1].pop());
     }

     // contabilizar el dinero que ha ganado A
     int cont = 0;
     while (!jugadores[0].isEmpty()) {
       if (jugadores[0].pop())
         cont++;
     }

     System.out.println("ha ganado: " + cont);
   }
}
```