



PED 1: Torneo de tenis

Programación y Estructuras de Datos Avanzadas



ADRIANA ARMENTAL TOMÉ

53489431-X

adri.13a@gmail.com

Índice

Enunciado de la practica: Torneo de tenis.....	2
Esquema algorítmico utilizado y como se aplica al problema	3
Forma no recursiva para resolver el problema y costes	7
Opciones posibles para el caso trivial del algoritmo.....	8
Ejemplos de ejecución.....	9
Código fuente	12

Enunciado de la practica: Torneo de tenis

Necesitamos organizar un torneo de tenis con n jugadores en donde cada jugador ha de jugar exactamente una vez contra cada uno de sus posibles $n-1$ competidores, y además ha de jugar un partido cada día, teniendo a lo sumo un día de descanso en todo el torneo. Se supone que hay campos de tenis suficientes para jugar cada día todos los partidos necesarios.

Se pide diseñar para resolverlo un algoritmo basado en el esquema de Divide y Vencerás.

Esquema algorítmico utilizado y como se aplica al problema

El esquema a utilizar es el Divide y Vencerás.

El término Divide y Vencerás en su acepción más amplia es algo más que una técnica de diseño de algoritmos. De hecho, suele ser considerada una filosofía general para resolver problemas y de aquí que su nombre no sólo forme parte del vocabulario informático, sino que también se utiliza en muchos otros ámbitos.

En nuestro contexto, Divide y Vencerás es una técnica de diseño de algoritmos que consiste en resolver un problema a partir de la solución de subproblemas del mismo tipo, pero de menor tamaño. Si los subproblemas son todavía relativamente grandes se aplicará de nuevo esta técnica hasta alcanzar subproblemas lo suficientemente pequeños para ser solucionados directamente. Ello naturalmente sugiere el uso de la recursión en las implementaciones de estos algoritmos.

Aparentemente estas son las características generales de todo diseño recursivo y de hecho el esquema Divide y Vencerás es un caso particular del mismo. Para distinguirlo de otros diseños recursivos que no responden a Divide y Vencerás se han de cumplir las siguientes condiciones:

- Los subproblemas han de tener un tamaño que sea una fracción del tamaño original (un medio, un tercio, etc). No basta simplemente con que sean más pequeños.
- Los subproblemas se generan exclusivamente a partir del problema original. En algunos diseños recursivos, los parámetros de una llamada pueden depender de los resultados de otra previa. En el esquema Divide y Vencerás, no.
- La solución del problema original se obtiene combinando los resultados de los subproblemas entre sí, y posiblemente con parte de los datos originales. Otras posibles combinaciones no encajan en el esquema.
- El (los) caso(s) base no son necesariamente los casos triviales. Como veremos más adelante podría utilizarse como caso base (incluso debería utilizarse en ocasiones) un algoritmo distinto al algoritmo recursivo Divide y Vencerás.

Puesto en forma de código, el esquema Divide y Vencerás tiene el siguiente aspecto:

```
template <class Problema, class Solución>
Solución divide-y-vencerás (Problema x) {
Problema x_1,...,x_k;
Solución y_1,...,y_k;
    if (base(x))
        return método-directo(x);
    else {
        (x_1,..., x_k) = descomponer(x);
        for (i=1; i<=k; i++)
            y_i = divide-y-vencerás(x_i);
        return combinar(x, y_1,..., y_k);
    }
}
```

Los tipos Problema, Solución, y los métodos base, método-directo, descomponer y combinar, son específicos de cada problema resuelto por el esquema.

Para saber si la aplicación del esquema Divide y Vencerás a un problema dado resultará en una solución eficiente o no, se deberá utilizar la recurrencia en la que el tamaño del problema disminuía mediante división:

$$T(n) = \begin{cases} c_1, & 0 \leq n < n_0 \\ a * T(n/b) + c * n^k, & n \geq n_0 \end{cases}$$

Recordemos que la solución de la misma era:

$$T(n) = \begin{cases} O(n^k), & a < b^k \\ O(n^k * \log n), & a = b^k \\ O(n^{\log_b a}), & a > b^k \end{cases}$$

Para obtener una solución eficiente, hay que conseguir a la vez:

- que el tamaño de cada subproblema sea lo más pequeño posible, es decir, maximizar b.
- que el número de subproblemas generados sea lo más pequeño posible, es decir, minimizar a.
- que el coste de la parte no recursiva sea lo más pequeño posible, es decir minimizar k.

La recurrencia puede utilizarse para anticipar el coste que resultará de la solución Divide y Vencerás, sin tener por qué completar todos los detalles. Si el coste sale igual o peor que el de un algoritmo ya existente, entonces no merecerá la pena aplicar Divide y Vencerás.

Para solucionar el problema del enunciado se proceden por partes, considerando los siguientes casos:

- a) Si n es potencia de 2, implementaremos un algoritmo para construir un cuadrante de partidas del torneo que permita terminarlo en n-1 días.
- b) Dado cualquier n>1, implementaremos un algoritmo para construir un cuadrante de partidas del torneo que permita terminarlo en n-1 días si n es par, o en n días si n es impar.

N POTENCIA DE 2

En el primer caso suponemos que n es una potencia de 2. El caso más simple se produce cuando sólo tenemos dos jugadores, cuya solución es fácil pues basta enfrentar uno contra el otro.

Si n>2, aplicaremos la técnica de Divide y Vencerás para construir la tabla pedida suponiendo que tenemos calculada ya una solución para la mitad de los jugadores, esto es, que tenemos relleno el cuadrante superior izquierdo de la tabla. En este caso los otros tres cuadrantes no son difíciles de rellenar, como puede observarse en la siguiente figura, y en donde se han tenido en cuenta la siguientes consideraciones para su construcción:

1. El cuadrante inferior izquierdo debe enfrentar a los jugadores de número superior entre ellos, por lo que se obtiene sumando $n/2$ a los valores del cuadrante superior izquierdo.
2. El cuadrante superior derecho enfrenta a los jugadores con menores y mayores números, y se puede obtener enfrentando a los jugadores numerados 1 a $n/2$ contra $(n/2)+1$ a n respectivamente en el día $n/2$, y después rotando los valores $(n/2)+1$ a n cada día.
3. Análogamente, el cuadrante inferior derecho enfrenta a los jugadores de mayor número contra los de menor número, y se puede obtener enfrentando a los jugadores $(n/2)+1$ a n contra 1 a $n/2$ respectivamente en el día $n/2$, y después rotando los valores 1 a n cada día, pero en sentido contrario a como lo hemos hecho para el cuadrante superior derecho.

N PAR

Si n es par llamaremos m al número $n \div 2$. Utilizando la técnica de Divide y Vencerás vamos a encontrar una forma de resolver el problema para n jugadores suponiendo que lo tenemos resuelto para m . Distinguiremos dos casos:

Si m es par, sabemos que existe una solución para enfrentar a esos m jugadores entre sí en $m-1$ días. Éste va a constituir el cuadrante superior izquierdo de la solución para n . Los otros tres cuadrantes se van a construir de igual forma al caso anterior cuando n es potencia de 2.

Si m es impar, su solución necesita m días. Esto va a volver a constituir el cuadrante superior izquierdo de la solución para el caso de n jugadores, pero ahora nos encontramos con que tiene algunos ceros, que necesitaremos rellenar convenientemente.

El cuadrante inferior izquierdo va a construirse como anteriormente, es decir, va a enfrentar a los jugadores de número superior entre ellos, por lo que se obtiene sumando $n/2$ a los valores del cuadrante superior que no sean cero.

El cuadrante superior derecho enfrenta a los jugadores con menores y mayores números y se va a obtener de forma similar a como lo construíamos antes, solo que no va a enfrentar a los jugadores 1 a $n/2$ contra $(n/2)+1$ a n en el día $n/2$, sino en cada una de las posiciones vacías del cuadrante superior izquierda. Los demás días del cuadrante superior derecho sí se van a obtener rotando esos valores cada día.

Análogamente, el cuadrante inferior derecho enfrenta a los jugadores de mayor número contra los de menor número, y se va a obtener enfrentando a los jugadores $(n/2)+1$ a n contra 1 a $n/2$ respectivamente, pero no en el día $n/2$, sino ocupando las posiciones vacías del cuadrante inferior izquierdo. Los valores restantes sí se obtendrán como antes, rotando los valores 1 a n cada día, pero en sentido contrario a como lo hemos hecho para el cuadrante superior.

N IMPAR

Por último si n es impar y sabemos resolver el problema para un número par de jugadores existe una solución al problema en n días, que se construye a partir de la

solución al problema para $n+1$ jugadores. Si n es impar entonces $n+1$ es par, y sea $S[1..n+1][1..n]$ el cuadrante solución para $n+1$ jugadores. Entonces podemos obtener el cuadrante solución para n jugadores $T[1..n][1..n]$ como:

$$T[jug, dia] = \begin{cases} S[jug, dia], & S[jug, dia] \neq n+1 \\ 0, & S[jug, dia] = n+1 \end{cases}$$

Es decir, utilizamos la convención de que un 0 en la posición $[i,j]$ de la tabla indica que el jugador i descansa (no se enfrenta a nadie) el día j , y aprovechamos este hecho para construir el cuadrante solución pedido.

En pseudocódigo sería

tipo tabla=vector[1..n,1..n-1] de 1..n

algoritmo formarTabla(sal t:tabla; ent inf,sup:1..n)

{Forma la parte de tabla correspondiente a los enfrentamientos de los participantes inf..sup}

variable medio:1..n

principio

 si inf:=sup-1 entonces

 t[inf,1]:=sup; t[sup,1]:=inf

 sino

 medio:=(inf+sup) div 2;

 formarTabla(t,inf,medio);

 formarTabla(t,medio+1,sup);

 completarTabla(t,inf,medio, medio,sup-1,medio+1);

 completarTabla(t,medio+1,sup, medio,sup-1,inf)

 fsi

fin

algoritmo completarTabla(ent/sal t:tabla; ent eqInf,eqSup, díaInf,díaSup, eqInic:1..n)

{Rellena t[eqInf..eqSup,díaInf..díaSup] con permutaciones cíclicas empezando en eqInic}

principio

 para j:=díaInf hasta díaSup hacer

 t[eqInf,j]:=eqInic+j-díaInf

 fpara;

 para i:=eqInf+1 hasta eqSup hacer

 t[i,díaInf]:=t[i-1,díaSup];

 para j:=díaInf+1 hasta díaSup hacer

 t[i,j]:=t[i-1,j-1]

 fpara

 fpara

fin

Forma no recursiva para resolver el problema y costes

La forma no recursiva de resolver el problema sería mediante la fuerza bruta, es decir, a raíz del número de jugadores n tendríamos que realizar $n!$ combinaciones que equivalen a diferentes formas de rellenar la tabla que es evidentemente mucho menos eficiente que la técnica divide y vencerás que lo que descompone el problema en un conjunto de subproblemas más pequeños. En nuestro caso, el coste cuadrático se debe a que, al combinar las soluciones para obtener la solución para el problema original se realizan dos bucles anidados completando el resto de la tabla con los mismos valores existentes en la otra mitad.

La solución del problema puede representarse en una tabla de dimensión $n \times (n-1)$. El elemento (i, j) -ésimo de la tabla, $1 \leq i \leq n$, $1 \leq j < n$, contiene el número del participante contra el que el participante i -ésimo compite el día j -ésimo.

Los casos base, $n = 2$ o $n = 1$, se resuelven trivialmente en tiempo constante. solución nos da pues los parámetros de coste $a = 2$, $b = 2$ y $k = 2$, que conducen a un coste esperado de $O(n^2)$. No puede ser menor puesto que la propia planificación consiste en rellenar $O(n^2)$ celdas. Mientras que resolver el problema por fuerza bruta sería un coste exponencial en función del número de combinaciones que se tendrían que realizar.

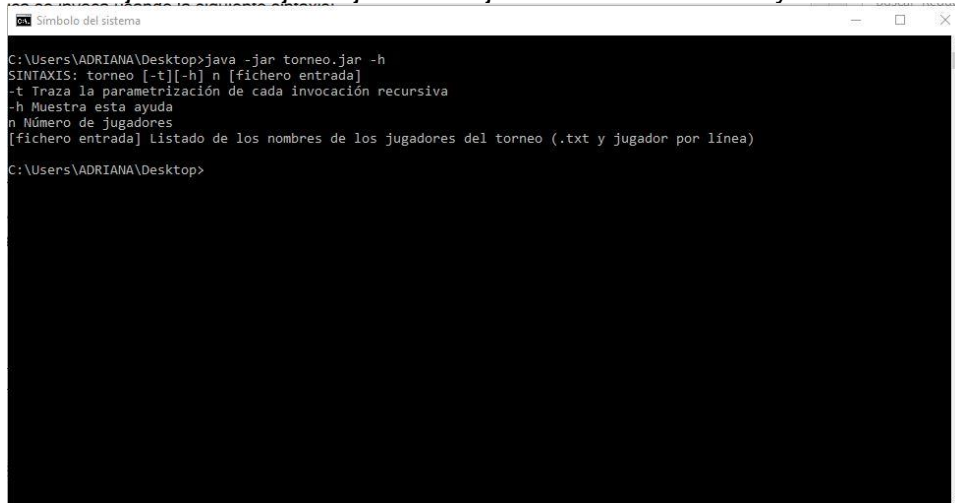
Opciones posibles para el caso trivial del algoritmo

Si tenemos un problema de tamaño n iríamos reduciendo por la mitad en cada llamada. Así para llegar desde n hasta 1, si este fuera el caso trivial, se producirán los tamaños de problema sucesivos $n/2$, $n/4$, $n/8$, $n/16$, ..., 4, 2, 1. Eso son exactamente $\log_2 n$ términos. Muchos menos que si vamos reduciendo de uno en uno $n-1$, $n-2$, ..., 3, 2, 1 donde hay $n-1$ términos. Y como $\log_2 n < n-1$ habremos reducido el coste del recursivo significativamente.

Ejemplos de ejecución

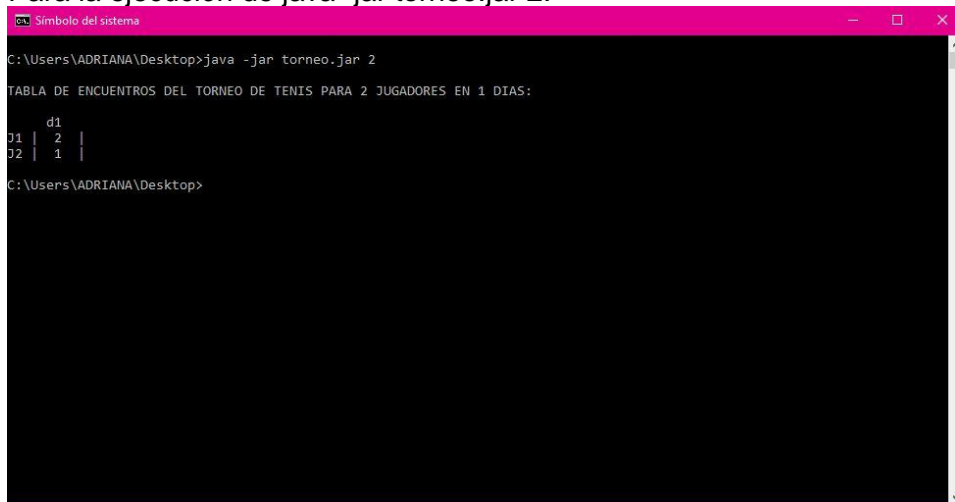
El programa se ejecuta usando la secuencia `java -jar`.

Para la ejecución de `java -jar torneo.jar -h`. Se muestra la ayuda.



```
Simbolo del sistema
C:\Users\ADRIANA\Desktop>java -jar torneo.jar -h
SINTAXIS: torneo [-t][-h] n [fichero entrada]
-t Traza la parametrización de cada invocación recursiva
-h Muestra esta ayuda
n Número de jugadores
[fichero entrada] listado de los nombres de los jugadores del torneo (.txt y jugador por línea)
C:\Users\ADRIANA\Desktop>
```

Para la ejecución de `java -jar torneo.jar 2`.



```
Simbolo del sistema
C:\Users\ADRIANA\Desktop>java -jar torneo.jar 2
TABLA DE ENCUENTROS DEL TORNEO DE TENIS PARA 2 JUGADORES EN 1 DIAS:

      d1
D1 | 2 |
D2 | 1 |
C:\Users\ADRIANA\Desktop>
```

Para la ejecución de `java -jar torneo.jar 4`.



```
Simbolo del sistema
C:\Users\ADRIANA\Desktop>java -jar torneo.jar 4
TABLA DE ENCUENTROS DEL TORNEO DE TENIS PARA 4 JUGADORES EN 3 DIAS:

      d1  d2  d3
D1 | 2 | 3 | 4 |
D2 | 1 | 4 | 3 |
D3 | 4 | 1 | 2 |
D4 | 3 | 2 | 1 |
C:\Users\ADRIANA\Desktop>
```

Para la ejecución de java -jar torneo.jar 5. En este caso como es un número impar de jugadores, habrá días de descanso.

```

C:\Users\ADRIANA\Desktop>java -jar torneo.jar 5
TABLA DE ENCUENTROS DEL TORNEO DE TENIS PARA 5 JUGADORES EN 5 DIAS:

```

	d1	d2	d3	d4	d5
d1	2	3	4	5	-
d2	1	5	3	-	4
d3	-	1	2	4	5
d4	5	-	1	3	2
d5	4	2	-	1	3

```

C:\Users\ADRIANA\Desktop>

```

Para la ejecución de java -jar torneo.jar 6.

```

C:\Users\ADRIANA\Desktop>java -jar torneo.jar 6
TABLA DE ENCUENTROS DEL TORNEO DE TENIS PARA 6 JUGADORES EN 5 DIAS:

```

	d1	d2	d3	d4	d5
d1	2	3	4	5	6
d2	1	5	3	6	4
d3	6	1	2	4	5
d4	5	6	1	3	2
d5	4	2	6	1	3
d6	3	4	5	2	1

```

C:\Users\ADRIANA\Desktop>

```

Para la ejecución de java -jar torneo.jar jugadores.txt. Solo se permite este formato de archivo pasado como parámetro.

```

C:\Users\ADRIANA\Desktop>java -jar torneo.jar jugadores.txt
TABLA DE ENCUENTROS DEL TORNEO DE TENIS PARA 6 JUGADORES EN 5 DIAS:

```

	d1	d2	d3	d4	d5
Raul	Pedro	Paula	Sara	Carol	Miguel
Pedro	Raul	Carol	Paula	Miguel	Sara
Paula	Miguel	Raul	Pedro	Sara	Carol
Sara	Carol	Miguel	Raul	Paula	Pedro
Carol	Sara	Pedro	Miguel	Raul	Paula
Miguel	Paula	Sara	Carol	Pedro	Raul

```

C:\Users\ADRIANA\Desktop>

```

Para la ejecución de la traza de java -jar torneo.jar -t 8. Se muestra la traza de la tabla del torneo para 8 jugadores.

```

C:\Users\ADRIANA\Desktop>java -jar torneo.jar -t 8

TABLA DE ENCUENTROS DEL TORNEO DE TENIS PARA 2 JUGADORES EN 1 DIAS:

    d1
J1 | 2 |
J2 | 1 |
TABLA DE ENCUENTROS DEL TORNEO DE TENIS PARA 4 JUGADORES EN 3 DIAS:

    d1    d2    d3
J1 | 2 | 3 | 4 |
J2 | 1 | 4 | 3 |
J3 | 4 | 1 | 2 |
J4 | 3 | 2 | 1 |
TABLA DE ENCUENTROS DEL TORNEO DE TENIS PARA 8 JUGADORES EN 7 DIAS:

    d1    d2    d3    d4    d5    d6    d7
J1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
J2 | 1 | 4 | 3 | 6 | 7 | 8 | 5 |
J3 | 4 | 1 | 2 | 7 | 8 | 5 | 6 |
J4 | 3 | 2 | 1 | 8 | 5 | 6 | 7 |
J5 | 6 | 7 | 8 | 1 | 4 | 3 | 2 |
J6 | 5 | 8 | 7 | 2 | 1 | 4 | 3 |
J7 | 8 | 5 | 6 | 3 | 2 | 1 | 4 |
J8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |

C:\Users\ADRIANA\Desktop>

```

Para la ejecución de la traza de java -jar torneo.jar -t jugadores.txt. Se muestra la traza de la tabla del torneo para 8 jugadores.

```

C:\Users\ADRIANA\Desktop>java -jar torneo.jar -t jugadores.txt

TABLA DE ENCUENTROS DEL TORNEO DE TENIS PARA 2 JUGADORES EN 1 DIAS:

    d1
Raul | Pedro |
Pedro | Raul |
TABLA DE ENCUENTROS DEL TORNEO DE TENIS PARA 4 JUGADORES EN 3 DIAS:

    d1    d2    d3
Raul | Pedro | Paula | Sara |
Pedro | Raul | Sara | Paula |
Paula | Sara | Raul | Pedro |
Sara | Paula | Pedro | Raul |
TABLA DE ENCUENTROS DEL TORNEO DE TENIS PARA 6 JUGADORES EN 5 DIAS:

    d1    d2    d3    d4    d5
Raul | Pedro | Paula | Sara | Carol | Miguel |
Pedro | Raul | Carol | Paula | Miguel | Sara |
Paula | Miguel | Raul | Pedro | Sara | Carol |
Sara | Carol | Miguel | Raul | Paula | Pedro |
Carol | Sara | Pedro | Miguel | Raul | Paula |
Miguel | Paula | Sara | Carol | Pedro | Raul |

C:\Users\ADRIANA\Desktop>

```

Y para otros casos, como por ejemplo el máximo (30) y mínimo (2) de jugadores, otros formatos de archivo, o otros argumentos se marca el error y se finaliza el programa.

```

C:\Users\ADRIANA\Desktop>java -jar torneo.jar hola
Argumento no válido

C:\Users\ADRIANA\Desktop>java -jar torneo.jar jugadores.pdf
Argumento no válido

C:\Users\ADRIANA\Desktop>java -jar torneo.jar
Debe introducir un valor

C:\Users\ADRIANA\Desktop>java -jar torneo.jar 1
El valor debe ser mayor a 1

C:\Users\ADRIANA\Desktop>java -jar torneo.jar 34
El valor excede el tamaño máximo permitido. Ingrese un número no mayor a 30

C:\Users\ADRIANA\Desktop>java -jar torneo.jar -t -h 6
Número de parámetros inválido

C:\Users\ADRIANA\Desktop>java -jar torneo.jar -t jugadores.pdf
Argumentos no válidos

C:\Users\ADRIANA\Desktop>

```

Código fuente

Clase encuentros.java:

```
import java.io.BufferedReader;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;

class encuentros {

    int maxJugadores = 30; //número máximo de jugadores permitidos
    int encuentros[][] = new int[maxJugadores][maxJugadores];
    //matriz de dos dimensiones para jugadores y dias
    int lineas = 0; //lineas que se leen del archivo en caso de
    haberlo
    String nomJugadores[] = new String[lineas]; //array de los
    nombres del archivo en caso de haberlo

    //función para saber si un número es potencia de dos
    public static boolean potenciaDos (int n) {
        int potencia = 2;
        while(potencia <= n) {
            if (n == potencia)
                return true;
            potencia *= 2;
        }
        return false;
    }

    //función para leer archivo
    public void leeArchivo (File archivo) throws
    FileNotFoundException, IOException {
        String cadena;
        int i=0;
        FileReader f = new FileReader(archivo);
        BufferedReader b = new BufferedReader(f);
        //primero contamos las lineas
        while((cadena = b.readLine())!=null) {
            lineas = lineas+1;
        }
        b.close();
        //volvemos a leer el archivo
        FileReader f2 = new FileReader(archivo);
        BufferedReader b2 = new BufferedReader(f2);
        //metemos los nombres en un array
        nomJugadores = new String[lineas];
        while((cadena = b2.readLine())!=null) {
            nomJugadores[i] = cadena;
            i++;
        }
        b2.close();
    }

    //función que crea la tabla de encuentros a partir del número de
    jugadores
    public void tablaTorneo (int n) {
        int jugador, dia;
        //si solo hay 2 jugadores solo se jugará un día
        if (n==2) {
```

```

        encuentros[0][0]=2;
        encuentros[1][0]=1;
    }
    //si es potencia de dos
    else if (potenciaDos(n)) {
        //iremos dividiendo hasta llegar a solo dos jugadores
        tablaTorneo(n/2);
        //cuando ya los tenemos, llenamos el cuadrante
superior derecho
        for (jugador=0; jugador<n/2; jugador++) {
            for (dia=n/2-1; dia<n-1; dia++) {
                encuentros[jugador][dia] = jugador+1 +
dia+1;

                if (encuentros[jugador][dia] > n)
                    encuentros[jugador][dia] =
encuentros[jugador][dia] - n/2;
            }
        }
        //llenamos el cuadrante inferior derecho
        for (jugador=n/2; jugador<n; jugador++) {
            for (dia=n/2-1; dia<n-1; dia++) {
                encuentros[jugador][dia] = jugador+1 -
(dia+1);

                if (encuentros[jugador][dia] <= 0)
                    encuentros[jugador][dia] =
encuentros[jugador][dia] + n/2;
            }
        }
        //llenamos el cuadrante inferior izquierdo
        for (jugador=n/2; jugador<n; jugador++) {
            for (dia=0; dia < n/2 - 1; dia++) {
                encuentros[jugador][dia] =
encuentros[jugador-n/2][dia] + n/2;
            }
        }
    }
    //si no es potencia de 2 pero el número de jugadores es par
    else if (n%2 == 0) {
        //iremos dividiendo hasta llegar a solo dos jugadores
        tablaTorneo(n/2);
        //cuando ya los tenemos, llenamos el cuadrante
superior derecho
        for (jugador=0; jugador<n/2; jugador++) {
            for (dia=n/2; dia<n-1; dia++) {
                encuentros[jugador][dia] = jugador+1 +
dia+1;

                if (encuentros[jugador][dia] > n)
                    encuentros[jugador][dia] =
encuentros[jugador][dia] - n/2;
            }
        }
        //llenamos el cuadrante inferior derecho
        for (jugador=n/2; jugador<n; jugador++) {
            for (dia=n/2; dia<n-1; dia++) {
                encuentros[jugador][dia] = jugador+1 -
(dia+1);

                if (encuentros[jugador][dia] <= 0)
                    encuentros[jugador][dia] =
encuentros[jugador][dia] + n/2;
            }
        }
    }
}

```

```

        //llenamos el cuadrante inferior izquierdo
        for (jugador=n/2; jugador<n; jugador++) {
            for (dia=0; dia<n/2; dia++) {
                if (encuentros[jugador-n/2][dia] != 0)
                    encuentros[jugador][dia] =
encuentros[jugador - n/2][dia] + n/2;
            }
        }
        //llenamos el cuadrante superior izquierdo
reemplazando los valores 0
        for (jugador=0; jugador<n/2; jugador++) {
            for (dia=0; dia<n/2; dia++) {
                if (encuentros[jugador][dia] == 0)
                    encuentros[jugador][dia] =
jugador+1 + n/2 ;
            }
        }
        //llenamos el cuadrante inferior izquierdo
reemplazando los valores 0
        for (jugador=n/2; jugador<n; jugador++) {
            for (dia=0; dia<n/2; dia++) {
                if (encuentros[jugador][dia] == 0)
                    encuentros[jugador][dia] =
jugador+1 - n/2;
            }
        }
    }
    //si el número de jugadores es impar
    else {
        //si n es impar, le sumamos 1 para volverlo par y que
en la llamada recursiva podamos llegar a 2 jugadores
        tablaTorneo(n+1);
        //se eliminan los valores que sobran por llamar a la
función con n+1 que serán los días de descanso
        for (jugador=0; jugador<n; jugador++) {
            for (dia=0; dia<n; dia++) {
                if (encuentros[jugador][dia]==n+1)
                    encuentros[jugador][dia] = 0;
            }
        }
    }
}

//función para sacar la tabla por pantalla con número
public void imprimirTabla(int n) {
    int dias, i, j, sdias, sjugador;
    //si n es par la cantidad de dias es n-1, si es impar la
cantidad de dias es n y hay día de descanso para los jugadores
    if(n%2 == 0)
        dias = n-1;
    else
        dias = n;

    System.out.println("\nTABLA DE ENCUENTROS DEL TORNEO DE
TENIS PARA "+n+" JUGADORES EN "+dias+" DIAS:\n");

    System.out.print("  ");
    for(i=0; i<dias; i++) {
        sdias = i+1;
        if(i+1 < 10)
            System.out.print("   d"+sdias+" ");
    }
}

```

```

        else
            System.out.print("  d"+sdias+"  ");
    }

    //imprimimos los valores
    if(n%2 != 0) {
        for(i=0; i<dias; i++) {
            sjugador = i+1;
            if(i+1 < 10)
                System.out.print("\nJ"+sjugador);
            else
                System.out.print("\nJ"+sjugador);

            for(j=0; j<dias; j++) {
                if(encuentros[i][j] == 0)
                    System.out.print(" | - ");
                else if(encuentros[i][j] < 10)
                    System.out.print(" | "
"+encuentros[i][j]+" ");
            }
            else
                System.out.print(" | "
"+encuentros[i][j]+" ");
            }
            System.out.print(" | ");
        }
    }
    else {
        for(i=0; i<dias+1; i++) {
            sjugador = i+1;
            if(i+1 < 10)
                System.out.print("\nJ"+sjugador);
            else
                System.out.print("\nJ"+sjugador);

            for(j=0; j<dias; j++) {
                if(encuentros[i][j] == 0)
                    System.out.print(" | - ");
                else if(encuentros[i][j] < 10)
                    System.out.print(" | "
"+encuentros[i][j]+" ");
            }
            else
                System.out.print(" | "
"+encuentros[i][j]+" ");
            }
            System.out.print(" | ");
        }
    }
}

//función para sacar la tabla por pantalla con nombres
public void imprimirTablaConNombres(int n, String[] nombres) {
    int dias, i, j, sdias;
    //si n es par la cantidad de dias es n-1, si es impar la
cantidad de dias es n y hay dia de descanso para los jugadores
    if(n%2 == 0)
        dias = n-1;
    else
        dias = n;

    System.out.println("\nTABLA DE ENCUENTROS DEL TORNEO DE
TENIS PARA "+n+" JUGADORES EN "+dias+" DIAS:\n");
}

```



```

        System.out.print(" ");
        for(i=0; i<dias; i++) {
            sdias = i+1;
            if(i+1 < 10)
                System.out.print("      d"+sdias+" ");
            else
                System.out.print("      d"+sdias+" ");
        }

//imprimimos los valores
        if(n%2 != 0) {
            for(i=0; i<dias; i++) {
                if(i+1 < 10)
                    System.out.print("\n"+nombres[i]);
                else
                    System.out.print("\n"+nombres[i]);

                for(j=0; j<dias; j++) {
                    if(encuentros[i][j] == 0)
                        System.out.print(" | - ");
                    else if(encuentros[i][j] < 10)
                        System.out.print(" | 
"+nombres[encuentros[i][j]-1]+" ");
                    else
                        System.out.print(" | 
"+nombres[encuentros[i][j]-1]+" ");
                }
                System.out.print(" | ");
            }
        }
        else {
            for(i=0; i<dias+1; i++) {
                if(i+1 < 10)
                    System.out.print("\n"+nombres[i]);
                else
                    System.out.print("\n"+nombres[i]);
                for(j=0; j<dias; j++) {
                    if(encuentros[i][j] == 0)
                        System.out.print(" | - ");
                    else if(encuentros[i][j] < 10)
                        System.out.print(" | 
"+nombres[encuentros[i][j]-1]+" ");
                    else
                        System.out.print(" | 
"+nombres[encuentros[i][j]-1]+" ");
                }
                System.out.print(" | ");
            }
        }
    }

//función que crea la tabla de encuentros a partir del número de
jugadores y muestra la traza
    public void trazaTorneo (int n) {
        int jugador, dia;
        //si solo hay 2 jugadores solo se jugará un día
        if (n==2) {
            encuentros[0][0]=2;
            encuentros[1][0]=1;
        }
        //si es potencia de dos

```

```

        else if (potenciaDos(n)) {
            //iremos dividiendo hasta llegar a solo dos jugadores
            trazaTorneo(n/2);
            //cuando ya los tenemos, llenamos el cuadrante
superior derecho
            for (jugador=0; jugador<n/2; jugador++) {
                for (dia=n/2-1; dia<n-1; dia++) {
                    encuentros[jugador][dia] = jugador+1 +
dia+1;

                    if (encuentros[jugador][dia] > n)
                        encuentros[jugador][dia] =
encuentros[jugador][dia] - n/2;
                }
            }
            //llenamos el cuadrante inferior derecho
            for (jugador=n/2; jugador<n; jugador++) {
                for (dia=n/2-1; dia<n-1; dia++) {
                    encuentros[jugador][dia] = jugador+1 -
(dia+1);

                    if (encuentros[jugador][dia] <= 0)
                        encuentros[jugador][dia] =
encuentros[jugador][dia] + n/2;
                }
            }
            //llenamos el cuadrante inferior izquierdo
            for (jugador=n/2; jugador<n; jugador++) {
                for (dia=0; dia < n/2 - 1; dia++) {
                    encuentros[jugador][dia] =
encuentros[jugador-n/2][dia] + n/2;
                }
            }
        }
        //si no es potencia de 2 pero el número de jugadores es par
        else if (n%2 == 0) {
            //iremos dividiendo hasta llegar a solo dos jugadores
            trazaTorneo(n/2);
            //cuando ya los tenemos, llenamos el cuadrante
superior derecho
            for (jugador=0; jugador<n/2; jugador++) {
                for (dia=n/2; dia<n-1; dia++) {
                    encuentros[jugador][dia] = jugador+1 +
dia+1;

                    if (encuentros[jugador][dia] > n)
                        encuentros[jugador][dia] =
encuentros[jugador][dia] - n/2;
                }
            }
            //llenamos el cuadrante inferior derecho
            for (jugador=n/2; jugador<n; jugador++) {
                for (dia=n/2; dia<n-1; dia++) {
                    encuentros[jugador][dia] = jugador+1 -
(dia+1);

                    if (encuentros[jugador][dia] <= 0)
                        encuentros[jugador][dia] =
encuentros[jugador][dia] + n/2;
                }
            }
            //llenamos el cuadrante inferior izquierdo
            for (jugador=n/2; jugador<n; jugador++) {
                for (dia=0; dia<n/2; dia++) {
                    if (encuentros[jugador-n/2][dia] != 0)

```

```

encuentros[jugador][dia] =
encuentros[jugador - n/2][dia] + n/2;
    }
    }
    //llenamos el cuadrante superior izquierdo
reemplazando los valores 0
    for (jugador=0; jugador<n/2; jugador++) {
        for (dia=0; dia<n/2; dia++) {
            if (encuentros[jugador][dia] == 0)
                encuentros[jugador][dia] =
jugador+1 + n/2 ;
        }
    }
    //llenamos el cuadrante inferior izquierdo
reemplazando los valores 0
    for (jugador=n/2; jugador<n; jugador++) {
        for (dia=0; dia<n/2; dia++) {
            if (encuentros[jugador][dia] == 0)
                encuentros[jugador][dia] =
jugador+1 - n/2;
        }
    }
    }
    //si el número de jugadores es impar
    else {
        //si n es impar, le sumamos 1 para volverlo par y que
        en la llamada recursiva podamos llegar a 2 jugadores
        trazaTorneo(n+1);
        //se eliminan los valores que sobran por llamar a la
        función con n+1 que serán los días de descanso
        for (jugador=0; jugador<n; jugador++) {
            for (dia=0; dia<n; dia++) {
                if (encuentros[jugador][dia]==n+1)
                    encuentros[jugador][dia] = 0;
            }
        }
    }
    imprimirTabla(n);
}

//función que crea la tabla de encuentros a partir del número de
jugadores y muestra la traza
public void trazaTorneoNombres (int n, String[] nombres) {
    int jugador, dia;
    //si solo hay 2 jugadores solo se jugará un día
    if (n==2) {
        encuentros[0][0]=2;
        encuentros[1][0]=1;
    }
    //si es potencia de dos
    else if (potenciaDos(n)) {
        //iremos dividiendo hasta llegar a solo dos jugadores
        trazaTorneoNombres(n/2, nombres);
        //cuando ya los tenemos, llenamos el cuadrante
superior derecho
        for (jugador=0; jugador<n/2; jugador++) {
            for (dia=n/2-1; dia<n-1; dia++) {
                encuentros[jugador][dia] = jugador+1 +
dia+1;

                if (encuentros[jugador][dia] > n)

```

```

encuentros[jugador][dia] =
encuentros[jugador][dia] - n/2;
    }
    }
    //llenamos el cuadrante inferior derecho
    for (jugador=n/2; jugador<n; jugador++) {
        for (dia=n/2-1; dia<n-1; dia++) {
            encuentros[jugador][dia] = jugador+1 -
(dia+1);
            if (encuentros[jugador][dia] <= 0)
                encuentros[jugador][dia] =
encuentros[jugador][dia] + n/2;
        }
    }
    //llenamos el cuadrante inferior izquierdo
    for (jugador=n/2; jugador<n; jugador++) {
        for (dia=0; dia < n/2 - 1; dia++) {
            encuentros[jugador][dia] =
encuentros[jugador-n/2][dia] + n/2;
        }
    }
    }
    //si no es potencia de 2 pero el número de jugadores es par
    else if (n%2 == 0) {
        //iremos dividiendo hasta llegar a solo dos jugadores
        trazaTorneoNombres(n/2, nombres);
        //cuando ya los tenemos, llenamos el cuadrante
superior derecho
        for (jugador=0; jugador<n/2; jugador++) {
            for (dia=n/2; dia<n-1; dia++) {
                encuentros[jugador][dia] = jugador+1 +
dia+1;
                if (encuentros[jugador][dia] > n)
                    encuentros[jugador][dia] =
encuentros[jugador][dia] - n/2;
            }
        }
        //llenamos el cuadrante inferior derecho
        for (jugador=n/2; jugador<n; jugador++) {
            for (dia=n/2; dia<n-1; dia++) {
                encuentros[jugador][dia] = jugador+1 -
(dia+1);
                if (encuentros[jugador][dia] <= 0)
                    encuentros[jugador][dia] =
encuentros[jugador][dia] + n/2;
            }
        }
        //llenamos el cuadrante inferior izquierdo
        for (jugador=n/2; jugador<n; jugador++) {
            for (dia=0; dia<n/2; dia++) {
                if (encuentros[jugador-n/2][dia] != 0)
                    encuentros[jugador][dia] =
encuentros[jugador - n/2][dia] + n/2;
            }
        }
        //llenamos el cuadrante superior izquierdo
reemplazando los valores 0
        for (jugador=0; jugador<n/2; jugador++) {
            for (dia=0; dia<n/2; dia++) {
                if (encuentros[jugador][dia] == 0)

```

```

                                encuentros[jugador][dia] =
jugador+1 + n/2 ;
                                }
                                }
                                //llenamos el cuadrante inferior izquierdo
reemplazando los valores 0
                                for (jugador=n/2; jugador<n; jugador++) {
                                    for (dia=0; dia<n/2; dia++) {
                                        if (encuentros[jugador][dia] == 0)
                                            encuentros[jugador][dia] =
jugador+1 - n/2;
                                        }
                                    }
                                }
                                //si el número de jugadores es impar
                                else {
                                    //si n es impar, le sumamos 1 para volverlo par y que
en la llamada recursiva podamos llegar a 2 jugadores
                                    trazaTorneoNombres(n+1, nombres);
                                    //se eliminan los valores que sobran por llamar a la
función con n+1 que serán los días de descanso
                                    for (jugador=0; jugador<n; jugador++) {
                                        for (dia=0; dia<n; dia++) {
                                            if (encuentros[jugador][dia]==n+1)
                                                encuentros[jugador][dia] = 0;
                                        }
                                    }
                                }
                                }imprimirTablaConNombres(n, nombres);
                                }
}

```

Clase torneo.java:

```
import java.io.File;
import java.io.IOException;
import java.nio.file.Files;

public class torneo {
    //función para saber si un valor es numérico
    public static boolean isNumeric(String cadena) {
        boolean resultado;
        try {
            Integer.parseInt(cadena);
            resultado = true;
        } catch (NumberFormatException excepcion) {
            resultado = false;
        }
        return resultado;
    }

    public static void main(String[] args) {
        int n = 0; //variable para coger el número de jugadores
        encuentros e = new encuentros(); //Objetos de la clase
        //si no hay parametros
        if(args.length == 0){
            System.out.println("Debe introducir un valor");
            System.exit(0);
        }

        //si solo hay un parametro
        else if(args.length == 1){
            //si el valor es -t
            String valor = args[0];
            if (valor.equals("-t")) {
                System.out.println("Debe introducir un número de
jugadores también");
                System.exit(0);
            }
            //si el valor es -h
            else if(valor.equals("-h")) {
                System.out.println("SINTAXIS: torneo [-t][-h] n
[fichero entrada]\n"
                                + "-t Traza la parametrización de cada
invocación recursiva\n"
                                + "-h Muestra esta ayuda\n"
                                + "n Número de jugadores\n"
                                + "[fichero entrada] Listado de los
nombres de los jugadores del torneo (.txt y jugador por línea)");
                System.exit(0);
            }
            else {
                //comprobar si existe fichero en la ruta actual
                File archivo = new File(valor);
                if (archivo.exists()) {
                    try {
                        String tipodeArchivo =
Files.probeContentType(archivo.toPath());
                        if(tipodeArchivo.equals("text/plain"))
                        {
                            e.leeArchivo(archivo);
                        }
                    }
                }
            }
        }
    }
}
```

```

do {
    //tiene que haber más
    de un jugador
    if(e.lineas <= 1) {
        System.out.println("El valor debe ser mayor a 1\n\n");
        System.exit(0);
    }
    else if (e.lineas >
e.maxJugadores) {
        System.out.println("El valor excede el tamaño máximo permitido.
Ingrese un número no mayor a "+e.maxJugadores);
        System.exit(0);
    }
    while(e.lineas <= 1 ||
e.lineas > e.maxJugadores);

    //creamos la tabla de
    encuentros
    e.tablaTorneo(e.lineas);

    //y la dibujamos
e.imprimirTablaConNombres(e.lineas, e.nomJugadores);
    }
    catch (IOException ioException) {
        System.out.println("Error: " +
ioException.getMessage());
        System.exit(0);
    }
    //si el valor no es numérico (número de
jugadores)
    else if (isNumeric(valor) != true) {
        System.out.println("Argumento no válido");
        System.exit(0);
    }
    //si es numérico
    else {
        n = Integer.parseInt(valor);
        do {
            //tiene que haber más de un jugador
            if(n <= 1) {
                System.out.println("El valor
debe ser mayor a 1\n\n");
                System.exit(0);
            }
            else if (n > e.maxJugadores) {
                System.out.println("El valor
excede el tamaño máximo permitido. Ingrese un número no mayor a
"+e.maxJugadores);
                System.exit(0);
            }
        }
        while(n <= 1 || n > e.maxJugadores);

        //creamos la tabla de encuentros
        e.tablaTorneo(n);

```

```

        //y la dibujamos
        e.imprimirTabla(n);
    }
}

//si solo hay dos parametros
else if(args.length == 2){
    //si el valor es -t
    String valor1 = args[0];
    String valor2 = args[1];
    //comprobar si existe fichero en la ruta actual
    File archivo = new File(valor2);
    if (valor1.equals("-t") && archivo.exists()) {
        try {
            String tipodeArchivo =
Files.probeContentType(archivo.toPath());
            if(tipodeArchivo.equals("text/plain")) {
                e.leeArchivo(archivo);
                do {
                    //tiene que haber más de un
jugador
                    if(e.lineas <= 1) {
                        System.out.println("El
valor debe ser mayor a 1\n\n");
                        System.exit(0);
                    }
                    else if (e.lineas >
e.maxJugadores) {
                        System.out.println("El
valor excede el tamaño máximo permitido. Ingrese un número no mayor a
"+e.maxJugadores);
                        System.exit(0);
                    }
                } while(e.lineas <= 1 || e.lineas >
e.maxJugadores);

                //creamos la traza de encuentros
                e.trazaTorneoNombres(e.lineas,
e.nomJugadores);
            }
        }
        catch (IOException ioException) {
            System.out.println("Error: " +
ioException.getMessage());
            System.exit(0);
        }
    }

    else if(valor1.equals("-t") && isNumeric(valor2) !=
true) {
        System.out.println("Argumentos no válidos");
        System.exit(0);
    }

    else if(valor1.equals("-t") && isNumeric(valor2) ==
true) {
        n = Integer.parseInt(valor2);
    }
}

```



```

        do {
            //tiene que haber más de un jugador
            if(n <= 1) {
                System.out.println("El valor debe
ser mayor a 1\n\n");
                System.exit(0);
            }
            else if (n > e.maxJugadores) {
                System.out.println("El valor excede
el tamaño máximo permitido. Ingrese un número no mayor a
"+e.maxJugadores);
                System.exit(0);
            }
        } while(n <= 1 || n > e.maxJugadores);

        //creamos la traza de encuentros
        e.trazaTorneo(n);

    }

    else {
        System.out.println("Argumentos no válidos");
        System.exit(0);
    }

    //si hay más de dos parámetro
    else{
        System.out.println("Número de parámetros inválido");
        System.exit(0);
    }

    System.out.println();
    System.exit(0);
}
}

```