EJERCICIOS TEMA 3

Algoritmia y Complejidad

Realizado por:

Gabriel López Cuenca
Miguel Ángel Losada Fernández
Sergio Sanz Sacristán
Álvaro Zamorano Ortega

Ejercicio 4

Se quiere programar un robot para poner tapones de corcho a las botellas de una fábrica de reciclado. Se tienen disponibles N botellas y los N corchos que las tapan (N es constante en el problema), pero hay una serie de problemas:

- **Las botellas son todas distintas entre sí, igual que los corchos: cada botella solo puede cerrarse con un corcho concreto, y cada corcho solo sirve para una botella concreta.**
- **El robot está preparado para cerrar botellas, por lo que lo único que sabe hacer es comparar corchos con botellas. El robot puede detectar si un corcho es demasiado pequeño, demasiado grande, o del tamaño justo para cerrar una botella.**
- **El robot no puede comparar corchos entre sí para "ordenarlos" por grosor, y tampoco puede hacerlo con las botellas.**
- **El robot tiene espacio disponible y brazos mecánicos para colocar botellas y corchos a su antojo, por ejemplo, en distintas posiciones de una mesa, si es necesario. Diseñar el algoritmo que necesita el robot para taponar las N botellas de manera óptima.**

RESOLUCIÓN → colocamos el pivote de un elemento del otro array y colocamos los elementos menores a la izquierda y los mayores a la derecha, comparando corchos con botellas y viceversa (Quicksort), y realizamos el ciclo otra vez, pero esta vez con el pivote siguiente.

- **ARRAY BOTELLAS** → {3,4,1,5}
- **ARRAY CORCHOS** → {5,4,3,1}

- **ORDENAR CORCHOS:**
  - PASO 1-> 3,4,1,5
  - PASO 2-> 3,1,4,5
  - PASO 3-> 1,3,4,5
  - PASO 4-> 1,3,4,5 → corchos ordenados

- **ORDENAR BOTELLAS:**
  - PASO 1-> 1,3,4,5
  - PASO 2-> 1,3,4,5
  - PASO 3-> 1,3,4,5
  - PASO 4-> 1,3,4,5 → botellas ordenadas

```java
public class Tema3_EJ4 {

    public static ArrayList<Integer> ordenado = new ArrayList<>();

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        ArrayList<Integer> corchos = new ArrayList<>();
        ArrayList<Integer> botellas = new ArrayList<>();

        //Añadir los mismos datos a los arrays
        corchos.add(5);
        corchos.add(4);
        corchos.add(3);
        corchos.add(1);
        botellas.add(3);
        botellas.add(4);
        botellas.add(1);
        botellas.add(5);

        corchos = ordenarCorchos(corchos, botellas, 0);
        System.out.print("Corchos: ");
        System.out.println(corchos);

        botellas = ordenarBotellas(botellas, corchos, 0);
        System.out.print("Botellas: ");
        System.out.println(botellas);

    }
    public static ArrayList<Integer> ordenarCorchos(ArrayList<Integer> corchos,
            ArrayList<Integer> botellas, int indice) {
        int pivote, primero;
        ArrayList<Integer> auxiliar = new ArrayList<>();

        //Si el pivote es la ultima botella
        if (indice == botellas.size() - 1) {
            pivote = botellas.get(indice);
            auxiliar.add(pivote);
            primero = 0;
            for (int i = 0; i < botellas.size(); i++) {
                //Si el corcho es menor que la botella pivote
                if (corchos.get(i) < pivote) {
                    //Se añaden a aux desde el principio
                    auxiliar.add(primero, corchos.get(i));
                    primero++;

                    //Si el corcho es mayor que la botella pivote
                } else if (corchos.get(i) > pivote) {
                    //Se añaden a aux desde el final
                    auxiliar.add(corchos.get(i));
                }
            }
            return auxiliar;
```

```
        } else {
            pivote = botellas.get(indice);
            auxiliar.add(pivote);
            primero = 0;
            for (int i = 0; i < botellas.size(); i++) {
                //Si el corcho es menor que la botella pivote
                if (corchos.get(i) < pivote) {
                    //Se añaden a aux desde el principio
                    auxiliar.add(primero, corchos.get(i));
                    primero++;
                    //Si el corcho es mayor que la botella pivote
                } else if (corchos.get(i) > pivote) {
                    //Se añaden a aux desde el final
                    auxiliar.add(corchos.get(i));
                }
            }

            //funcion recursiva para el pivote siguiente
            return ordenarCorchos(auxiliar, botellas, indice + 1);
        }
    }

    public static ArrayList<Integer>
            ordenarBotellas(ArrayList<Integer> botellas,
                    ArrayList<Integer> corchos, int indice) {
        int pivote, primero;
        ArrayList<Integer> auxiliar = new ArrayList<>();

        //Si el pivote es el ultimo corcho
        if (indice == corchos.size() - 1) {
            pivote = corchos.get(indice);
            auxiliar.add(pivote);
            primero = 0;
            for (int i = 0; i < corchos.size(); i++) {
                //Si la botella es menor que el corcho pivote
                if (botellas.get(i) < pivote) {
                    //Se añaden a aux desde el principio
                    auxiliar.add(primero, botellas.get(i));
                    primero++;
                    //Si la botella es mayor que el corcho pivote
                } else if (botellas.get(i) > pivote) {
                    //Se añaden a aux desde el final
                    auxiliar.add(botellas.get(i));
                }
            }
            return auxiliar;
```

```
        } else {
            pivote = corchos.get(indice);
            auxiliar.add(pivote);
            primero = 0;
            for (int i = 0; i < corchos.size(); i++) {
                //Si la botella es menor que el corcho pivote
                if (botellas.get(i) < pivote) {
                    //Se añaden a aux desde el principio
                    auxiliar.add(primero, botellas.get(i));
                    primero++;
                    //Si la botella es mayor que el corcho pivote
                } else if (botellas.get(i) > pivote) {
                    //Se añaden a aux desde el final
                    auxiliar.add(botellas.get(i));
                }
            }

            //funcion recursiva para el pivote siguiente
            return ordenarBotellas(auxiliar, corchos, indice + 1);
        }
    }
}
```

Ejercicio 7

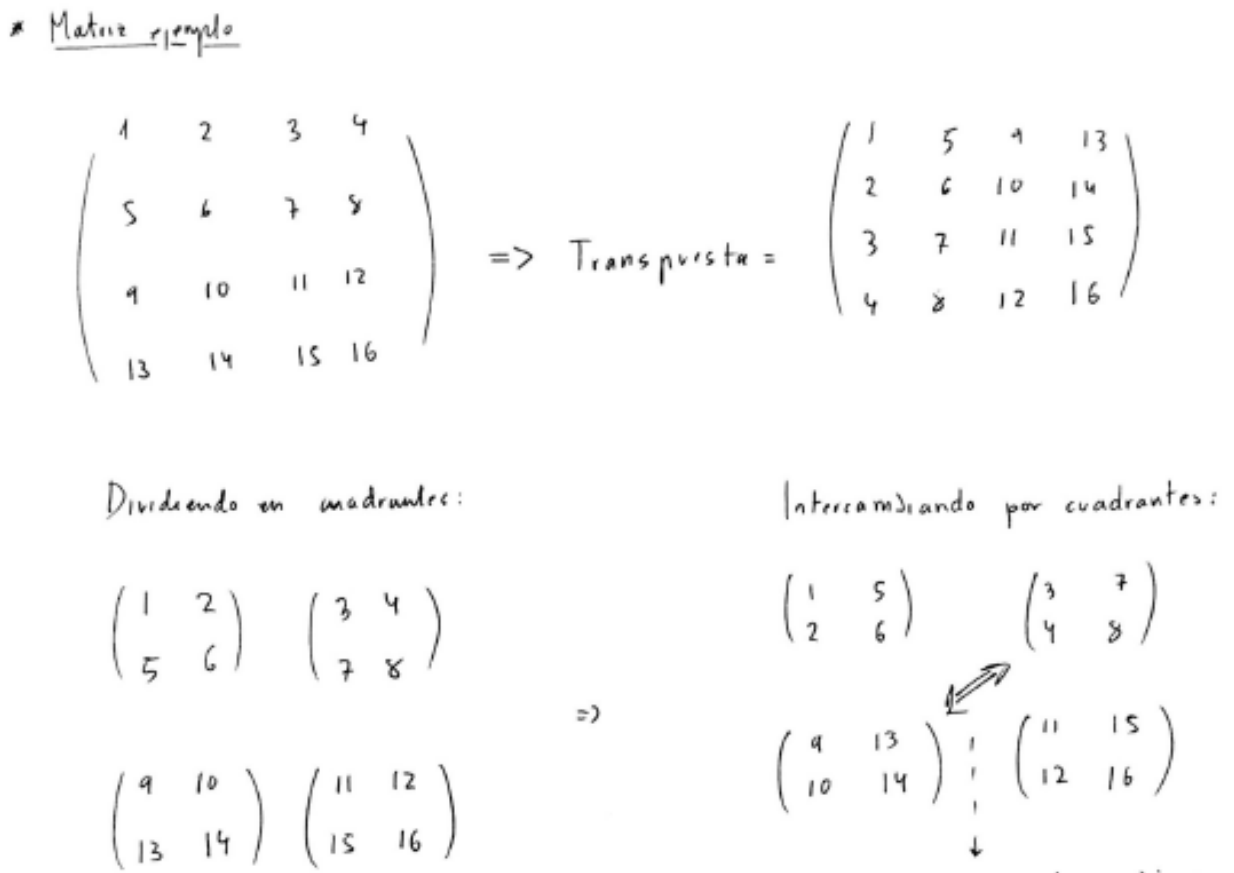
En Abecelandia, ciudad famosa por sus N bellas plazas y que puede que conozcas, tienen un curioso sistema de carreteras: desde cada plaza sale una calle a todas las otras plazas que comienzan con una letra que se encuentre en su nombre (por ejemplo, de la plaza Aro salen calles que llevan a las plazas que empiezan por R, como las plaza Ruido y Reto, o por O, como la plaza Osa, pero no salen calles a plazas como Duende, Cascada o Tiara).

Las calles son de sentido único (de la plaza Aro se puede ir a la plaza Ruido, pero no al revés ya que no cumple la regla de las letras; obviamente, otras plazas como Aro y Osa están conectadas entre sí en ambas direcciones). Todas estas conexiones entre las N plazas están recogidas en un callejero, representado por una matriz de adyacencia de tamaño NxN; así, el valor de Callejero[p, q] indica si se puede ir de la plaza p a la plaza q.

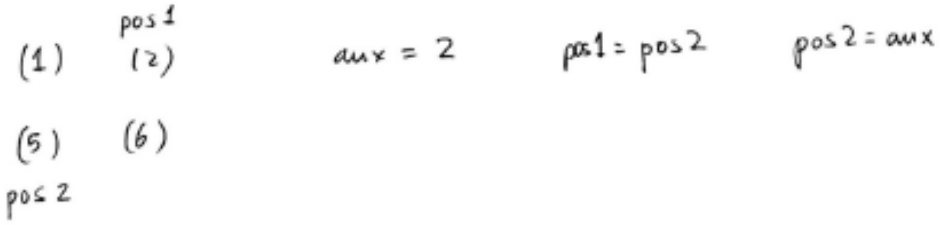
Se acerca el día 26 de Abril, festividad de San Isidoro de Sevilla (patrón de las Letras y, casualmente, de los informáticos), y en el Ayuntamiento de Abecelandia han decidido que para celebrarlo van a invertir la dirección de todas las calles que conectan sus plazas. En ese día no se podrá circular de Aro a Ruido, pero si se permitirá ir de Ruido a Aro; obviamente, Aro y Osa seguirán conectadas entre sí.

Diseñar formalmente un algoritmo Divide y Vencerás estándar que, teniendo por dato el callejero de la ciudad (representado por la matriz de adyacencia), obtenga el nuevo callejero valido para el dia de San Isidoro de Sevilla, indicando las estructuras de datos que se utilicen.

A pesar de que el callejero está formado por 0 y 1, realizamos a mano la matriz transpuesta de números cualesquiera para observar el algoritmo, en el cual nos damos cuenta de que además de cambiar filas por columnas, es necesario intercambiar el cuadrante superior derecho por el inferior izquierdo, y que de esta forma la matriz final queda como la deseada.



* En el código, volver a dividir cada cuadrante



```java
public class Tema3_EJ7 {

    /**
     * @param args the command line arguments
     */
    private static int N;

    public static void main(String[] args) {
        System.out.print("Introducir potencia de 2 como longitud del "
                + "callejero: ");
        Scanner s = new Scanner(System.in);
        N = s.nextInt();

        System.out.println();

        int[][] callejero = generarCallejero();
        System.out.println("Imprimiendo callejero original");
        imprimirM(callejero);

        filasColumnas(callejero, 0, N - 1, 0, N - 1);
        System.out.println("Imprimiendo callejero transpuesto");
        imprimirM(callejero);
    }

    public static void cambioCuadrante(int[][] matriz, int filaIniA,
            int colIniA, int filaIniB, int colIniB, int dimen) {
        //Intercambia el 3º cuadrante por el 2º independientemente del
        //tamaño de la matriz
        for (int i = 0; i < dimen; i++) {
            for (int j = 0; j < dimen; j++) {
                //aux es el 3º cuadrante
                int aux = matriz[filaIniA + i][colIniA + j];
                //se intercambia el 3º por el 2º
                matriz[filaIniA + i][colIniA + j]
                        = matriz[filaIniB + i][colIniB + j];
                //se coloca en el 2º el valor de auxiliar
                matriz[filaIniB + i][colIniB + j] = aux;
            }
        }
    }
```

```
    public static void filasColumnas(int[][] matriz, int filaInicio,
            int filaFin, int colInicio, int colFin) {
        //Cuando la matriz es lxl, filaInicio = filaFin, no hay que
        //intercambiar
        if (filaInicio < filaFin) {
            int filaMedio = (filaInicio + filaFin) / 2;
            int colMedio = (colInicio + colFin) / 2;

            //Dividir la matriz en cuadrantes mediante los índices, a la mitad
            //tanto por columnas como filas. Tantas veces hasta que lleguemos
            //al caso base
            filasColumnas(matriz, filaInicio, filaMedio, colInicio, colMedio);
            filasColumnas(matriz, filaInicio, filaMedio, colMedio + 1, colFin);
            filasColumnas(matriz, filaMedio + 1, filaFin, colInicio, colMedio);
            filasColumnas(matriz, filaMedio + 1, filaFin, colMedio + 1, colFin);

            //Llamada a intercambiar con los valores para cambiar el
            //3º cuadrante por el 2º
            cambioCuadrante(matriz, filaMedio + 1, colInicio, filaInicio,
                    colMedio + 1, filaFin - filaMedio);
        }
    }

    public static void imprimirM(int[][] vector) {
        //Método para imprimir la matriz
        for (int i = 0; i < N; i++) {
            for (int j = 0; j < N; j++) {
                System.out.print(vector[i][j] + " ");
                System.out.print("\t");
            }
            System.out.println();
        }
        System.out.println();
    }

    public static int[][] generarCallejero() {
        int[][] callejero = new int[N][N];
        for (int i = 0; i < N; i++) {
            for (int j = 0; j < N; j++) {
                if (i == j) { //Diagonal principal de 0, ya que no hay calle
                    //a la misma calle
                    callejero[i][j] = 0;
                } else {
                    //El resto de posiciones se rellena aleaoriamente
                    int m = (int) Math.floor(Math.random() * 2);
                    callejero[i][j] = m;
                }
            }
        }

        return callejero;
    }

}
```