Memoria final de prácticas VV

María Gallego Martín

Carlos Gómez Robles

Yeray Granada Layos

Daniel Rodríguez Manzanedo

Miguel Ángel López Roche

Github: https://github.com/mariagallegomartin/VVroche

Índice

1. Práctica 0

1.1. Implantación Merge Sort

Ejercicio para hacer en clase.

Dada la siguiente implementación del merge sort:

```
    public static void sort(int arr[], int l, int r)
    {
      if (l < r)
        {
            int m = (l+r)/2;

            sort(arr, l, m);
            sort(arr , m+1, r);

            merge(arr, l, m, r);
        }
    }

    public static void merge(int arr[], int l, int m, int r)
    {
        int n1 = m - l + 1;
        int n2 = r - m;

        int L[] = new int [n1];
        int R[] = new int [n2];

        for (int i=0; i<n1; ++i)
            L[i] = arr[l + i];
        for (int j=0; j<n2; ++j)
            R[j] = arr[m + 1+ j];
        int i = 0, j = 0;

        int k = l;
        while (i < n1 && j < n2)
        {
```

```
            if (L[i] <= R[j])
            {
                arr[k] = L[i];
                i++;
            }
            else
            {
                arr[k] = R[j];
                j++;
            }
            k++;
        }


    }
```

Al ejecutar el código obtenemos las siguientes salidas:

Input: [0, 1, 2, 3, 4] → Output: [0, 1, 2, 3, 4] (correcto)

Input: [4, 3, 2, 1, 0] → Output: [0, 0, 2, 0 ,0] (error)

Solución:

Lo primero que hemos hecho al crear la clase merge sort ha sido crear hacer un main para poder ejecutar las pruebas con los inputs: [0,1,2,3,4] y [4,3,2,1,0] y printear la salida. Al realizer la primera ejecución observamos como, tal cual dice el enunciado, el primer output si lo realiza bien pero el segundo no.

```java
public class MergeSort {
        public static void main(String [ ] args){

            int[] a1 = {0,1,2,3,4};
            int[] a2 = {4,3,2,1,0};

            sort(a1, 0 , a1.length-1);

            for (int i = 0; i < a1.length; i++) {
                System.out.println(a1[i]);
            }

            System.out.println("------------------------------------------------------------");

            sort(a2, 0 , a2.length-1);

            for (int j = 0; j < a2.length; j++) {
                System.out.println(a2[j]);
            }
        }
```

Lo siguiente ha sido comprobar si la función sort, tal cual viene dada, realiza un correcto funcionamiento. Tras las pruebas realizadas confirmamos que ejecuta la division de los array de forma correcta.

```
public static void sort(int arr[], int l, int r)
{
    if (l < r)
    {
        int m = (l+r)/2;

        sort(arr, l, m);
        sort(arr , m+1, r);

        merge(arr, l, m, r);
    }
}
```

Por lo tanto, el error se encuentra en la función de merge. Tras hacer el debugging nos hemos dado cuenta que al tener en el while && cuando cumple una de las condiciones durante el merge sale del bucle, faltando así elementos del array por unir. Esto se ha solucionado mediante dos while que controlen que se copian los elementos restantes del lado derecho y del izquierdo.

Código dado:

```
public static void merge(int arr[], int l, int m, int r)
{
    int n1 = m - l + 1;
    int n2 = r - m;

    int L[] = new int [n1];
    int R[] = new int [n2];

    for (int i=0; i<n1; ++i)
        L[i] = arr[l + i];
    for (int j=0; j<n2; ++j)
        R[j] = arr[m + 1+ j];
    int i = 0, j = 0;

    int k = l;
    while (i < n1 && j < n2)
    {
        if (L[i] <= R[j])
        {
            arr[k] = L[i];
            i++;
        }
        else
        {
            arr[k] = R[j];
            j++;
        }
        k++;
    }
```

Lo añadido:

```
            while (i < n1)
            {
                arr[k] = L[i];
                i++;
                k++;
            }

            while (j < n2)
            {
                arr[k] = R[j];
                j++;
                k++;
            }

        }

    }
```

2. Práctica 1

2.1. Ejercicio 1

Dado el siguiente Código de ordenación por el método de la burbuja:

```
public class BubbleSort {
	/**
	 * This method the BubbleSort method. Returns
	 * the array in increasing order.
	 *
	 * @param array The array to be sorted
	 * @param size The count of total number of elements in array
	 **/

	public static <T extends Comparable<T>> void BS(T array[], int size) {
		boolean swap;
		int last = size - 1;

		do
		{
			swap = false;
			for (int count = 0; count < last-1; count++)
			{
				int comp = array[count].compareTo(array[count + 1]);
				if (comp > 0)
				{
					T temp = array[count];
					array[count] = array[count + 1];
					array[count + 1] = temp;
					swap = true;
				}
			}
		} while (swap);
	}
}
```

Al ejecutar el código con los valores de entrada obtenemos estos resultados:
[1 0 0 0 2 0] --> [0 0 0 1 2 0]
[8 7 6 5 4 9] --> [4 5 6 7 8 9]

Rellena la siguiente plantilla aplicando el método científico para depuración:

Hipótesis	El código se ejecuta correctamente
Predicción	Al ejecutar el código recibiremos una salida
Experimento	Ejecutar el código
Observación	No ejecuta
Conclusión	Hipótesis rechazada

Hipótesis	Es necesario el método main para que funcione el código
Predicción	Al ejecutar el código recibiremos una salida
Experimento	Creamos el método main y ejecutamos el código

Observación	Se obtiene una salida
Conclusión	Hipótesis confirmada

Hipótesis	Al introducir un array desordenado, el programa nos devuelve el mismo array ordenado de forma ascendente
Predicción	Obtenemos el array ordenado
Experimento	Ejecutamos el código pasando el array [1 0 0 0 2 0]
Observación	El array no se encuentra ordenado de forma ascendente
Conclusión	Hipótesis rechazada

Hipótesis	El do while no funciona correctamente
Predicción	Al ejecutar el código obtenemos un array no ordenado de forma ascendente
Experimento	Ejecutamos el código utilizando el Debbuging para comprobar si el bucle so while recorre el array hasta que todos los elementos están ordenados de forma ascendente
Observación	El array no se encuentra ordenado pero no por fallo del do while
Conclusión	Hipótesis rechazada

Hipótesis	El for no recorre todos los elementos del array
Predicción	Si pasamos un array, el elemento que ocupa la última posición no es comparado
Experimento	Observar hasta que elemento del array recorre el for
Observación	No alcanza el último elemento del array
Conclusión	Hipótesis confirmada

Hipótesis	La variable last es la que produce el error
Predicción	Al ajustar last = size recorrerá todos los elementos del array
Experimento	Cambiamos last = size -1 por last = size
Observación	El for recorre todos los elementos del array
Conclusión	Hipótesis confirmada

Código final:

** Las distintas versiones previas al código final se encuentra en git indicado en la portada en la carpeta VVroche/Parte1/BubbleSort. En este documento solo mostraremos el código ya depurado.

```
public class BubbleSort {
    /**
     * This method the BubbleSort method. Returns
     * the array in increasing order.
     *
     * @param array The array to be sorted
     * @param size The count of total number of elements in array
     **/

    public static <T extends Comparable<T>> void BS(T array[], int size) {
        boolean swap;
        int last = size;

        do
        {
            swap = false;
            for (int count = 0; count < last-1; count++)
            {
                int comp = array[count].compareTo(array[count + 1]);
                if (comp > 0)
                {
                    T temp = array[count];
                    array[count] = array[count + 1];
                    array[count + 1] = temp;
                    swap = true;
                }
            }
        } while (swap);
    }

    public static void main (String[] args){
        Integer [] solucion = {8,7,6,5,4,9};
        BubbleSort bs = new BubbleSort();
        bs.BS(solucion,solucion.length);
        for(int i=0; i<solucion.length; i++){
            System.out.println(solucion[i]);
        }

    }
}
```

2.2. Ejercicio 2

Dado el siguiente Código Fuente:

```
public class Goldbach {
	/**
	 * Checks if the number n is prime. Negative numbers and zero are declared
	 * to be non-prime.
	 *
	 * @param n
	 *            Number to be tested.
	 * @return True if n is prime, false if not.
	 */
	public static boolean isPrime(Integer n) {
		for (int p = 2; p < Math.sqrt(n)-1; p++) {
			if (n % p == 0) {
				return false;
			}
		}

		return true;
	}

	/**
	 * Checks Goldbach conjecture for n, that is, checks if the even number n
	 * can be written as the sum of two prime numbers.
	 *
	 * If n is odd, it returs automatically true.
	 *
	 * @param n
	 *            Number to be checked.
	 * @return True if n can be decomposed as sum of prime numbers, false if
	 *         not. Also prints the decomposition if true.
	 */
	public static boolean checkGoldbach(Integer n) {
		for (int p = 1; p < n / 2; p++) {
			if (isPrime(p) && isPrime(n - p)) {
				System.out.println(n + "=" + p + " & " + (n - p));
				return true;
			}
		}

		return false;
	}
}
```

Al ejecutar el código con los valores de entrada obtenemos estos resultados:
13=2 & 11 → CORRECTO!
23=4 & 19 → INCORRECTO!

Rellena la siguiente plantilla aplicando el método científico para depuración:

Hipótesis	El código se ejecuta correctamente
Predicción	Al ejecutar, obtendremos una salida
Experimento	Ejecutamos el código
Observación	No se obtiene ninguna salida
Conclusión	Hipótesis rechazada

Hipótesis	Se necesita un método main para su correcto funcionamiento
Predicción	Obtendremos una salida
Experimento	Creamos un método main
Observación	Obtenemos una salida
Conclusión	Hipótesis aceptada

Hipótesis	El for del método checkGoldBach recorre todas las posibilidades
Predicción	For recorre todos los valores entre 1 y n/2 inclusive
Experimento	Ejecutar el método con n=2
Observación	El for no se ejecuta porque p=1 y y n/2=1 por lo que ya se cumpliría la condición de salida del for y, por lo tanto, nunca comprueba n/2
Conclusión	Hipótesis rechazada

Hipótesis	El for del método checkGoldBach recorre todas las posibilidades
Predicción	For recorre todos los valores entre 1 y n/2 inclusive
Experimento	En el for, cambiamos la condición de salida por p < N/2 +1 y ejecutar el método con n=2
Observación	Se entra al for y comprueba p=1 y p=N/2 (en este caso ambos p=1) que es el único valor posible para n=2
Conclusión	Hipótesis aceptada

Hipótesis	El for del método is prime recorre todas las posibilidades
Predicción	El For recorre los valores entre 2 y sqtr(n) inclusive
Experimento	Ejecutar el método con n=4
Observación	El for no se ejecuta porque p=2 y y sqtr(n)=2 por lo que ya se cumpliría la condición de salida del for sqtr(n) – 1 y, por lo tanto, nunca comprueban todos los valores
Conclusión	Hipótesis rechazada

Hipótesis	El for del método is prime recorre todas las posibilidades
Predicción	El For recorre los valores entre 2 y sqtr(n) inclusive

Experimento	Cambiar la condición de salida por p < Math.sqtr(n) + 1 y añadir en el if la condición de que n!=p. Ejecutamos el método con n=5
Observación	El for comprueba p=2 y sqtr(4), en este caso ambos p=2, el único divisor posible para este caso y realiza el método correctamente
Conclusión	Hipótesis aceptada

Código final:

** Las distintas versiones previas al código final se encuentra en git indicado en la portada en la carpeta VVroche/Parte1/Goldbach. En este documento solo mostraremos el código ya depurado.

```
public class Goldbach {
	/**
	 * Checks if the number n is prime. Negative numbers and zero are declared
	 * to be non-prime.
	 *
	 * @param n
	 *            Number to be tested.
	 * @return True if n is prime, false if not.
	 */
	public static boolean isPrime(Integer n) {
		for (int p = 2; p < Math.sqrt(n)+1; p++) {
			if ((n!=p)&&(n % p == 0)) {
				return false;
			}
		}

		return true;
	}

	/**
	 * Checks Goldbach conjecture for n, that is, checks if the even number n
	 * can be written as the sum of two prime numbers.
	 *
	 * If n is odd, it returs automatically true.
	 *
	 * @param n
	 *            Number to be checked.
	 * @return True if n can be decomposed as sum of prime numbers, false if
	 *         not. Also prints the decomposition if true.
	 */
```

```
	public static boolean checkGoldbach(Integer n) {
		for (int p = 1; p < n / 2+1; p++) {
			if (isPrime(p) && isPrime(n - p)) {
				System.out.println(n + "=" + p + " & " + (n - p));
				return true;
			}
		}
		return false;
	}
	public static void main(String [] args) {
		Goldbach gb = new Goldbach();
		gb.checkGoldbach(5);
	}
```

2.3. Ejercicio 3

Dada la siguiente implementación de una pila.

```
public class Stack {
	private Integer[] arrayStack;
	private Integer MAX_SIZE = 5;
	private Integer top;

	/**
	 * Create a stack LIFO with finite deep.
	 */
	public Stack() {
		this.arrayStack = new Integer[MAX_SIZE];
		this.top = 1;
	}

	/**
	 * Inserts an element in the stack. If the stack is full, it shows a message
	 * and stops
	 *
	 * @param element
	 *            Element to be inserted.
	 */
	public void push(Integer element) {
		if (top >= MAX_SIZE) {
			System.out.println("The stack is full");
		}
		arrayStack[this.top] = element;
		top++;
	}

	/**
	 * Extracts an element from the stack. If the stack is empty, it shows a
	 * message and returns -1.
	 *
	 * @return Extracted element.
	 */
	public Integer pull() {
		if (top <= 0) {
			System.out.println("The stack is empty");
			return 0;
		}
```

```
		top--;
		return arrayStack[top - 1];
	}

	/**
	 * Returns the number of elements in the stack.
	 *
	 * @return Number of elements in the stack.
	 */
	public Integer getSize() {
		return top;
	}

	/**
	 * Extract nElements objects of the stack.
	 *
	 * @param nElements
	 *            Number of elements to be extracted.
	 * @return Array with the extracted elements.
	 */
	public Integer[] pull(Integer nElements) {
		Integer[] res = new Integer[nElements];

		for (int i = 0; i < nElements; i++) {
			res[i] = pull();
		}

		return res;
	}

	/**
	 * Copy an stack into a new stack. The new stack has the same elements in
	 * the same order as the old one.
	 *
	 * @return Duplicated stack.
	 */
	public Stack copyStack() {
		Stack newStack = new Stack();

		for (int i = 0; i < this.getSize(); i++) {
			newStack.push(arrayStack[i]);
		}

		return newStack;
	}

}
```

Al ejecutar el código obtenemos los siguientes resultados.

1.

```
Stack stack = new Stack();
stack.push(1);
stack.push(2);
stack.push(3);
stack.push(4);  Correcto!!
```

2.

```
Stack stack = new Stack();
stack.push(12);
stack.push(2);
stack.push(43);
```

```
stack.push(95);
Integer v[] = stack.pull(stack.getSize());

Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException:
Array index out of range: -1
      at stack.Stack.pull(Stack.java:44)
      at stack.Stack.pull(Stack.java:67)
      at stack.Stack.main(Stack.java:97)
```

Siendo esta ejecución INCORRECTA

3.

```
Stack stack = new Stack();
stack.push(1);
stack.push(2);
stack.push(3);
stack.push(4);
Stack nueva = stack.copyStack();

The stack is full
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException:
Array index out of range: 5
      at stack.Stack.push(Stack.java:30)
      at stack.Stack.copyStack(Stack.java:86)
      at stack.Stack.main(Stack.java:102)
```

Siendo esta ejecución INCORRECTA

4.

```
stack.push(1);
stack.push(2);
stack.push(3);
stack.push(4);
System.out.println("Elemento = " +  stack.pull());

Elemento = 3
```

Siendo esta ejecución INCORRECTA

Aplique el método científico para su depuración rellenando la plantilla correspondiente.

Hipótesis	El código se ejecuta correctamente
Predicción	Recibiremos una salida
Experimento	Runeamos el código
Observación	Produce error, falta un main
Conclusión	Hipótesis rechazada

Hipótesis	El código permite apilar valores en la pila
Predicción	Lo hará sin errores
Experimento	Introducimos los valores 1,2,3 y 4
Observación	El primer valor de la pila es null

Conclusión	Hipótesis rechazada

Hipótesis	El código permite apilar valores en la pila
Predicción	Lo hará sin errores
Experimento	Introducimos los valores 1,2,3 y 4 y cambiamos en el constructor top=1 por top= 0
Observación	Se introduce correctamente
Conclusión	Hipótesis aceptada

Hipótesis	No se pueden introducir más valores que el máximo permitido
Predicción	Saldrá un mensaje de error
Experimento	Introducimos 6 valores
Observación	Produce error ArrayIndexOutOfBounds
Conclusión	Hipótesis rechazada

Hipótesis	No se pueden introducir más valores que el máximo permitido
Predicción	Saldrá un mensaje de error
Experimento	Introducimos 6 valores y añadimos un else a las sentecias posteriores al if
Observación	Se ejecuta correctamente
Conclusión	Hipótesis aceptada

Hipótesis	Es posible sacar un elemento de la pila
Predicción	Saldrá el último elemento
Experimento	A una pila con valores 1,2,3,4 le sacamos el valor 4
Observación	Devuelve el valor 3
Conclusión	Hipótesis rechazada

Hipótesis	Es posible sacar un elemento de la pila
Predicción	Saldrá el último elemento
Experimento	A una pila con valores 1,2,3,4 le sacamos el valor 4 y cambiamos de orden las sentencias top–- y el número a devolver
Observación	Devuelve el valor 4
Conclusión	Hipótesis aceptada

Hipótesis	No es posible sacar un valor de una pila vacía
Predicción	Saldrá un mensaje explicativo y devolverá el número 0
Experimento	Intentamos sacar un valor de una pila vacía
Observación	Lo ejecuta correctamente
Conclusión	Hipótesis aceptada

Hipótesis	Se puede sacar elementos de uno en uno de un array y meterlo en otra
Predicción	Saldrán en el orden contrario en la segunda array
Experimento	Ejecutamos `stack.pull(stack.getSize());`
Observación	Se ejecuta tal cual se predice
Conclusión	Hipótesis aceptada

Hipótesis	Es posible copiar una pila
Predicción	Al copiar una pila tendrá los mismos valores que la otra
Experimento	Copiamos la pila 1,2,3,4 con el método stack.copyStack();
Observación	Lo ejecuta correctamente
Conclusión	Hipótesis aceptada

Código final:

** Las distintas versiones previas al código final se encuentra en git indicado en la portada en la carpeta VVroche/Parte1/Stack. En este documento solo mostraremos el código ya depurado.

```
public class Stack {
	private Integer[] arrayStack;
	private Integer MAX_SIZE = 5;
	private Integer top;

	/**
	 * Create a stack LIFO with finite deep.
	 */
	public Stack() {
		this.arrayStack = new Integer[MAX_SIZE];
		this.top = 0;
	}

	/**
	 * Inserts an element in the stack. If the stack is full, it shows a message
	 * and stops
	 *
	 * @param element
	 *            Element to be inserted.
	 */
	public void push(Integer element) {
		if (top >= MAX_SIZE) {
			System.out.println("The stack is full");
		}
		else {
		arrayStack[this.top] = element;
		top++;
		}
	}

	/**
	 * Extracts an element from the stack. If the stack is empty, it shows a
	 * message and returns -1.
	 *
	 * @return Extracted element.
	 */
	public Integer pull() {
		if (top <= 0) {
			System.out.println("The stack is empty");
			return 0;
		}

		Integer aux = arrayStack[top - 1];
		top--;
		return aux;
```

```
	}

	/**
	 * Returns the number of elements in the stack.
	 *
	 * @return Number of elements in the stack.
	 */
	public Integer getSize() {
		return top;
	}

	/**
	 * Extract nElements objects of the stack.
	 *
	 * @param nElements
	 *            Number of elements to be extracted.
	 * @return Array with the extracted elements.
	 */
	public Integer[] pull(Integer nElements) {
		Integer[] res = new Integer[nElements];

		for (int i = 0; i < nElements; i++) {
			res[i] = pull();
		}

		return res;
	}

	/**
	 * Copy an stack into a new stack. The new stack has the same elements in
	 * the same order as the old one.
	 *
	 * @return Duplicated stack.
	 */
	public Stack copyStack() {
		Stack newStack = new Stack();

		for (int i = 0; i < this.getSize(); i++) {
			newStack.push(arrayStack[i]);
		}

		return newStack;
	}

	public static void main (String[] args){
		Stack stack = new Stack();
		stack.push(1);
		stack.push(2);
		stack.push(3);
		stack.push(4);
		System.out.println("Elemento = " +  stack.pull());
    }

}
```

3. Práctica 2

Dado el siguiente código fuente en lenguaje Java:

- Aplicación de Agenda
- Algoritmo de Dijsktra

Se pide emplear técnicas de análisis estático para analizar dicho código. Concretamente se solicita analizar ambos códigos fuentes mediante las técnicas de Reading by Abstraction y CheckList. Únicamente deberán detectarse los errores que hay en dichos códigos, no es necesario corregirlos.

Lea atentamente los comentarios dentro del código proporcionado y compruebe que el código realiza lo que indica. Recuerde que las revisiones no se realizan EJECUTANDO EL CODIGO, se pide una revisión estática mediante técnicas de lectura.

Las CheckList a emplear pueden encontrarse anexas a esta prática.

3.1. Checklist

3.1.1. Checklist Dijkstra

Java Code Inspection Checklist
1. Variable and Constant Declaration Defects (VC)

1. Are descriptive variable and constant names used in accord with naming conventions?
Afirmativo, todos los nombres de constants y variables cumplen con los estándares acordados.
Ejemplo: líneas de la 11 a la 19, todas correctamente nombradas.

2. Are there variables with confusingly similar names?
No, todas las variables tienen un nombre fácilmente identificable y sin opción a equívocos.
Ejemplo: en la línea 108 se declara una variable NewDistance cerca de la variable distance (línea 112), que ambas se refieren a distancias, pero está claro cuál es la nueva y cual no.

3. Is every variable properly initialized?
Si, por ejemplo, de la línea 52 a la 54 donde hay 3 variables bien inicializadas.

4. Could any non-local variables be made local?
No, todas las variables no locales aparecen en más de un método.
Por ejemplo: la variable dijkstraExec aparece en la línea 134 y en la 154, líneas que pertenecen a dos métodos diferentes.

5. Are there literal constants that should be named constants?
No, no hay ningún caso de constante literal que se repita tanto como para que sea necesario crear una constante. Ejemplo: línea 112, se utiliza la constante literal 0.0 pero no aparece en ningún otro momento.

6. Are there macros that should be constants?
No, no existe ningún macro.

7. Are there variables that should be constants?

No, todas las variables son variables bien creadas, ninguna tiene que ser constante. Por ejemplo: nVertices (línea 16) obtiene su valor en los parámetros del constructor (línea 41)

2. Function Definition Defects (FD)

8. Are descriptive function names used in accord with naming conventions?
Si, todas tienen un nombre acorde a los estándares.
Por ejemplo: en la línea 73 la función nextCur se adapta a los estándares.

9. Is every function parameter value checked before being used?
No. Por ejemplo: en la línea 107 en parámetro ini de la función es utilizado en la línea 111 sin haberlo chekeado antes.

10. For every function: Does it return the correct value at every function return point?
Si, por ejemplo, en la línea 179 es una función que devuelve un booleano como bien hace en la línea 180.

3. Class Definition Defects (CD)

11. Does each class have an appropriate constructor and destructor?
Sí, aunque java no hace uso de destructores, sí podemos encontrar un constructor para la clase Dijkstra que comienza en la línea 41.

12. For each member of every class: Could access to the member be further restricted?
No, ya que todas las declaraciones de variables de la clase (líneas 13 a 19) hacen el acceso restringido.

13. Do any derived classes have common members that should be in the base class?
En este código no hay clases derivadas.

14. Can the class inheritance hierarchy be simplified?
No, más simple no podría ser porque solo hay una clase.

4. Computation/Numeric Defects (CN)

15. Is overflow or underflow possible during a computation?
No contemplamos que se pueda producir overflow ni underflow.

16. For each expressions with more than one operator: Are the assumptions about order of evalu-ation and precedence correct?
Sí, en la línea 118 se puede ver un claro ejemplo.

17. Are parentheses used to avoid ambiguity?
No se utilizan paréntesis en ninguna expresión para evitar la ambigüedad, pero también es cierto que no son necesarios, como en la línea 78.

5. Comparison/Relational Defects (CR)

18. Are the comparison operators correct?

Sí, todos los comparadores están usados correctamente. Un ejemplo del uso de estos se puede ver en la línea 118.

19. Is each boolean expression correct?
Sí, por ejemplo en la línea 78 se utiliza el boolean visited[i] inicializado a false con anterioridad de forma correcta.

20. Are there improper and unnoticed side-effects of a comparison?
No

6. Control Flow Defects (CF)

21. For each loop: Is the best choice of looping constructs used?
Si, por ejemplo, en la línea 57 el bucle for usado está correctamente elegido ya que inicializa un array pasando así por todos los valores.

22. Will all loops terminate?
Si. Por ejemplo: en la línea 113 funciona correctamente porque va cambiando el valor de false a true en la línea 130, hasta que visited[end]= true y sale del bucle.

23. When there are multiple exits from a loop, is each exit necessary and handled properly?
No existe ninguna salida multiple en ningún bucle.

24. Does each switch statement have a default case?
No existe ningún switch.

25. Are missing switch case break statements correct and marked with a comment?
No existe ningún switch.

26. Is the nesting of loops and branches too deep, and is it correct?

No, el nivel máximo de anidamiento en es la línea 113 con un while que anida con el for de la línea 115, que además es correcto ya que no sería simplificable.

27. Can any nested if statements be converted into a switch statement?
No, en ningún caso sería adecuado. Por ejemplo: en la línea 154 solo hay un if por lo que no sería necesario crear un switch.

28. Are null bodied control structures correct and marked with braces or comments?
No existe ninguna estrcutura de control con un cuerpo null en el código.

29. Does every function terminate?
Si, por ejemplo, la función de la línea 73, todos sus bucles acaban y además devuelve el valor del tipo correcto.

30. Are goto statements avoided?
No existe ninguna sentencia goto.

7. Input-Output Defects (IO)

31. Have all files been opened before use?

No se utiliza ningún archivo para guardar o leer datos.

32. Are the attributes of the open statement consistent with the use of the file?
No se utiliza ningún archivo para guardar o leer datos.

33. Have all files been closed after use?
No se utiliza ningún archivo para guardar o leer datos.

34. Is buffered data flushed?
No está implícito en el código.

35. Are there spelling or grammatical errors in any text printed or displayed?
No se imprime ningún texto por pantalla.

36. Are error conditions checked?
No, por ejemplo, en la línea 78 no verifica el valor de las variables usadas.

8. Module Interface Defects (MI)

37. Are the number, order, types, and values of parameters in every function call in agreement with the called function's declaration?
Si, los parámetros en cada llamada de cada función están conforme a los parámetros que aparecen en la declaración de la función llamada. Por ejemplo: línea 131 aparece una llamada a la función nextCur con los parámetros adecuados, en este caso ninguno.

38. Do the values in units agree (e.g., inches versus yards)?
Si. Por ejemplo, la unidades de distancia son siempre double, es decir, si fuesen metros todos serian metros y si fuesen kilómetros, todos serían kilómetros.

9. Comment Defects (CM)

39. Does every function, class, and file have an appropriate header comment?
Si. Por ejemplo, de la línea 21 a la 40 es un comentario describiendo el constructor.

40. Does every variable or constant declaration have a comment?
No, por ejemplo, la variable next de la línea 74 no está contada.

41. Is the underlying behavior of each function and class expressed in plain language?
Si, todos los comentarios explican de forma clara el comportamiento subyacente de las funciones y la clase.
Por ejemplo: de la línea 3 a la 9 explica la utilización de la clase implementada.

42. Is the header comment for each function and class consistent with the behavior of the functionor class?
Si, todas las descripciones de funciones corresponden con lo que hace el código. Por ejemplo: de la línea 47 a la 48 dice que lo hace la función de inicializar los datos y la función hace lo que dice el comentario.

43. Do the comments and code agree?
Sí, no existe ninguna inconsistencia entre los comentarios y el código que le sigue. Por ejemplo: línea 51 dice que crea las estructuras de datos y en las líneas 52,53 y 54 lo hace.

44. Do the comments help in understanding the code?
Si, se entiende perfectamente. Por ejemplo: línea 111, existe un comentario explicativo en la misma línea sobre la utilidad de esa línea de código.

45. Are there enough comments in the code
Si, está explicado prácticamente línea por línea, solo las líneas más básicas no tienen explicación debido a su obviedad, por lo que existen comentarios suficientes.

46. Are there too many comments in the code?
No, porque los comentarios no son suficientes como para ensuciar el código y ayudan a una comprensión total del código desarrollado.

10. Packaging Defects (LP)

47. For each file: Does it contain only one class?
Si, solo hay un fichero java con una única clase llamada Dijkstra.

48. For each function: Is it no more than about 60 lines long?
No, hay ninguna función con más de 60 líneas, la más extensa es computerShortestPath que va de la línea 107 a la 136 (29 líneas).

49. For each class: Is no more than 2000 lines long (Sun Coding Standard) ? Praktikum Software Engineering 99: Code Inspection Checklist

Solo hay una clase y tiene 182 líneas, por lo que no supera las 2000.

11. Modularity Defects (MO)

50. Is there a low level of coupling between packages (classes)?

No hay acoplamiento, ya que solo hay una clase.

51. Is there a high level of cohesion within each package?

Si, ya que solo hay una clase donde todos los métodos tienen una temática común. Por ejemplo, dentro de la función computeShortestPath se llama a la función initializeDataStructures.

52. Is there duplicate code that could be replaced by a call to a function that provides the behaviorof the duplicate code?

No, no existe código duplicado.

53. Are framework classes used where and when appropriate?

No, no hay uso de frameworks.**Dijkstra**

3.1.2. Checklist Agenda

Java Code Inspection Checklist

1. Variable and Constant Declaration Defects (VC)

1. Are descriptive variable and constant names used in accord with naming conventions?
Clase Agenda: si, por ejemplo, la variable numEntries (línea 24)
Agenda Interface: no hay variables.
Clase Entry: si, por ejemplo, la variable surname (línea 28)
Clase Parser: si, por ejemplo, la variable PRO (línea 22)

2. Are there variables with confusingly similar names?
Clase Agenda: no, cada una tiene un nombre claramente autodescriptivo, por ejemplo, en la línea 13 la variable info.
Agenda Interface: NO existen variables
Clase Entry: no, cada una tiene un nombre claramente autodescriptivo, por ejemplo, en la línea 16 la variable name.
Clase Parser: no, cada una tiene un nombre claramente autodescriptivo, por ejemplo, en la línea 60 la variable telefono.

3. Is every variable properly initialized?
Clase Agenda: si, por ejemplo, numEntries= 0 en la línea 28
Agenda Interface: no hay variables
Clase Entry: si, por ejemplo, country= " " en la línea 31
Clase Parser: si, por ejemplo, line = null en la línea 31

4. Could any non-local variables be made local?
Clase Agenda: no, tanto first como nunEntries aparecen en varios métodos
Agenda Interface: no hay variables
Clase Entry: no, ya que solo hace sets y gets con todas las variables
Clase Parser: no, todas las variables no locales son correctas

5. Are there literal constants that should be named constants?
Clase Agenda: no, no hay ninguna constante literal
Agenda Interface: no hay constantes literales
Clase Entry: no, no hay constantes literales
Clase Parser: no, un ejemplo correcto serían los -1 de la línea 76

6. Are there macros that should be constants?
Clase Agenda: no, no existen macros
Agenda Interface: no, no existen macros
Clase Entry: no, no existen macros
Clase Parser: no, no existen macros

7. Are there variables that should be constants?
Clase Agenda: no, no hay ninguna que no deba modificar su valor, por ejemplo: línea 82 la variable numEntries
Agenda Interface: no hay variables
Clase Entry: no, no hay ninguna que no deba modificar su valor, por ejemplo: línea 59 la variable name
Clase Parser: no, no hay ninguna que no deba modificar su valor, por ejemplo: línea 81 la variable provincia

2. Function Definition Defects (FD)

8. Are descriptive function names used in accord with naming conventions?
Clase Agenda: si, por ejemplo, línea 39 el método addEntry
Agenda Interface: si, por ejemplo, línea 11 el método isEmpty
Clase Entry: si, por ejemplo, línea 131 el método getZip
Clase Parser: si, por ejemplo, línea 53 el método a createEntry

9. Is every function parameter value checked before being used?
Clase Agenda: no, no checkea ninguno
Agenda Interface: no, no checkea ninguno
Clase Entry: no, no checkea ninguno
Clase Parser: no, no checkea ninguno

10. For every function: Does it return the correct value at every function return point?
Clase Agenda: si, por ejemplo, la función boolean loadAgenda devuelve un booleano en la línea 153
Agenda Interface: no hay ningún return
Clase Entry: si, por ejemplo, la función voidTelephone no devuelve nada en la línea 87
Clase Parser: si, por ejemplo, la función getLine devuelve un string en la línea 49

3. Class Definition Defects (CD)

11. Does each class have an appropriate constructor and destructor?
Clase Agenda: sí, aunque no hace uso de destructores se puede ver el constructor de la clase Agenda en la línea 26 y el constructor de la clase AgendaNode en la 16.
AgendaInterface: no tiene constructor ni destructor, al ser una interfaz que ha de ser implementada desde la clase Agenda.
Clase Entry: en la línea 25 podemos encontrar el constructor de la clase Entry.
Clase Parser: sí, en la línea 30 encontramos el constructor de la clase Parser.

12. For each member of every class: Could access to the member be further restricted?
Clase Agenda: no, ya que las declaraciones de las variables de la clase son private (líneas 23 y 24).
AgendaInterface: al tratarse de la interfaz no posee miembros propios.
Clase Entry: no, todas las variables están declaradas como private en la líneas 16 a 23.
Clase Parser: no, en las líneas 27 y 28 se pueden ver las que las declaraciones de las variables son private.

13. Do any derived classes have common members that should be in the base class?
No hay clases derivadas para ninguna de las clases.

14. Can the class inheritance hierarchy be simplified?
No ya que no hay herencia de clases.

4. Computation/Numeric Defects (CN)

15. Is overflow or underflow possible during a computation?
No encontramos la posibilidad de que se produzca overflor o underflow.

16. For each expressions with more than one operator: Are the assumptions about order of evalu-ation and precedence correct?

Clase Agenda: sí, en la línea 45 podemos ver un ejemplo de ello.
AgendaInterface: esta clase no tiene expresiones que evaluar.
Clase Entry: sí, como por ejemplo en la línea 45.
Clase Parser: sí, en la línea 76 encontramos un claro ejemplo.

17. Are parentheses used to avoid ambiguity?
Clase Agenda: no, aunque no son necesarios, como por ejemplo en la línea 74.
AgendaInterface: esta clase no tiene expresiones en las que usar paréntesis.
Clase Entry: no, en la única expresión del código (línea 45) no se usan (no necesarios).
Clase Parser: sí, en la línea 76 se puede observar su uso de manera correcta.

5. Comparison/Relational Defects (CR)

18. Are the comparison operators correct?
Clase Agenda: sí, como por ejemplo en la línea 46.
AgendaInterface: al ser una interfaz no posee operadores que evaluar.
Clase Entry: no se usan operadores de comparación.
Clase Parser: sí, se puede ver el ejemplo en la línea 76.

19. Is each boolean expression correct?
Clase Agenda: sí, por ejemplo en la línea 46 se utiliza el boolean found que se inicializa antes a false.
AgendaInterface: la interfaz no tiene booleans.
Clase Entry: no tiene expresiones booleanas.
Clase Parser: sí, como en la línea 88 que utiliza el boolean auxb que ha sido inicializado con anterioridad.

20. Are there improper and unnoticed side-effects of a comparison?
No, en las comparaciones encontradas en las clases no hemos podido encontrar ninguna en la que se puedan producir efectos secundarios.

6. Control Flow Defects (CF)

21. For each loop: Is the best choice of looping constructs used?.
Clase Agenda: si, por ejemplo, el while de la línea 133
Agenda Interface: no hay bucles
Clase Entry: no hay bucles
Clase Parser: no hay bucles

22. Will all loops terminate?
Clase Agenda: no, en el bucle while de la línea 133 no termina, ya que no pasa al siguiente de cur.
Agenda Interface: no hay bucles
Clase Entry: no hay bucles
Clase Parser: no hay bucles

23. When there are multiple exits from a loop, is each exit necessary and handled properly?
Clase Agenda: si, por ejemplo, el bucle while de la línea 74, tiene dos salida y controla ambas de forma correcta
Agenda Interface: no hay bucles
Clase Entry: no hay bucles
Clase Parser: no hay bucles

24. Does each switch statement have a default case?
Clase Agenda: no hay ninguna sentencia switch
Agenda Interface: no hay ninguna sentencia switch
Clase Entry: no hay ninguna sentencia switch
Clase Parser: no hay ninguna sentencia switch

25. Are missing switch case break statements correct and marked with a comment?
Clase Agenda: no hay ninguna sentencia switch
Agenda Interface: no hay ninguna sentencia switch
Clase Entry: no hay ninguna sentencia switch
Clase Parser: no hay ninguna sentencia switch

26. Is the nesting of loops and branches too deep, and is it correct?
Clase Agenda: no hay ningún anidamiento
Agenda Interface: no hay bucles
Clase Entry: no hay bucles
Clase Parser: no hay bucles

27. Can any nested if statements be converted into a switch statement?
Clase Agenda: no hay ningún anidamiento
Agenda Interface: no hay bucles
Clase Entry: no hay bucles
Clase Parser: no hay bucles

28. Are null bodied control structures correct and marked with braces or comments?
Clase Agenda: no hay ninguna estrcutura de control
Agenda Interface: no hay ninguna estrcutura de control nula
Clase Entry: no hay ninguna estrcutura de control
Clase Parser: no hay ninguna estrcutura de control

29. Does every function terminate?
Clase Agenda: si, por ejemplo, la función nEntries de la línea 105
Agenda Interface: si, ya que solo son las declaraciones que luego se implementan en clase agenda
Clase Entry: si, por ejemplo, setAdress de la línea 67
Clase Parser: si, por ejemplo, createEntry de la línea 53

30. Are goto statements avoided?
Clase Agenda: no existe ninguna sentencia goto
Agenda Interface: no existe ninguna sentencia goto
Clase Entry: no existe ninguna sentencia goto
Clase Parser: no existe ninguna sentencia goto

7. Input-Output Defects (IO)

31. Have all files been opened before use?
Clase Agenda: si, por ejemplo, en la línea 129
Agenda Interface: no hay ficheros
Clase Entry: no hay ficheros
Clase Parser: no hay ficheros

32. Are the attributes of the open statement consistent with the use of the file?

Clase Agenda: si, por ejemplo, el uso del fichero en la función saveAgenda de la línea 123
Agenda Interface: no hay ficheros
Clase Entry: no hay ficheros
Clase Parser: no hay ficheros

33. Have all files been closed after use?
Clase Agenda: no, en la línea 129 se abre un ficheor que nunca llega a cerrar
Agenda Interface: no hay ficheros
Clase Entry: no hay ficheros
Clase Parser: no hay ficheros

34. Is buffered data flushed?
Clase Agenda: no limpia en buffer en ningún momento
Agenda Interface: no hay ficheros
Clase Entry: no hay ficheros
Clase Parser: no hay ficheros

35. Are there spelling or grammatical errors in any text printed or displayed?
Clase Agenda: no hay nada escrito para imprimir solo le llegan strings ya formadas
Agenda Interface: no hay nada para imprimir
Clase Entry: no hay errores como en la asignación de la línea 112
Clase Parser: no hay errores como en la asignación de la línea 113

36. Are error conditions checked?
Clase Agenda: si, en la línea 123 se invoca una excepción en caso de error
Agenda Interface: no hay opción a que haya un error
Clase Entry: no hay ningún control de error
Clase Parser: no hay ningún control de error

8. Module Interface Defects (MI)

37. Are the number, order, types, and values of parameters in every function call in agreementwith the called function's declaration?
Clase Agenda: si, por ejemplo, el getName() de la línea 74
Agenda Interface: no hay llamadas a métodos
Clase Entry: si, por ejemplo, la función equals de la línea 45
Clase Parser: si, por ejemplo, la función createEntry de la línea 42

38. Do the values in units agree (e.g., inches versus yards)?
Clase Agenda: no existen valores que puedan tener distintas unidades
Agenda Interface: no hay ningún valor
Clase Entry: no existen valores que puedan tener distintas unidades
Clase Parser: no existen valores que puedan tener distintas unidades

9. Comment Defects (CM)
39. Does every function, class, and file have an appropriate header comment?
Clase Agenda: No, esta clase no tiene ningún tipo de comentario además de que algunas de sus funciones como nEntries() o isEmpty(), también carecen de comentario explicativo. Por otro lado, todos los comentarios que aparecen, son apropiados a las funcionalidades.
Clase Entry: Tanto la clase como todas sus funciones tienen sus respectivos comentarios explicativos.

Clase Parser: La clase Parser tiene comentada su funcionalidad, pero sus funciones, ningún tiene un comentario explicativo.
AgendaInterface: El archivo contenedor de la interfaz tiene un comentario explicativo, sus funciones no están comentadas, pero se explica su funcionalidad en la clase Agenda, pero esto no pasa con las funciones nEntries() y isEmpty().

40. Does every variable or constant declaration have a comment?
Clase Agenda: Ninguna variable tiene comentario
Clase Entry: Ninguna variable tiene comentario
Clase Parser: Ninguna variable tiene comentario
AgendaInterface: No existen ni constantes ni variables

41. Is the underlying behavior of each function and class expressed in plain language?
Si, por ejemplo, todos los nombres de las funciones son descriptivos y adecuados a la función que representa. Lo mismo ocurre con las variables.

42. Is the header comment for each function and class consistent with the behavior of the function or class?
Todas las funciones y clases comentadas tienen un comentario que corresponde adecuadamente a la definición y a su funcionalidad.

43. Do the comments and code agree?
Sí, no encontramos ninguna inconsistencia entre los comentarios y el código, como por ejemplo en las líneas 38 a 42 de la clase Entry se explica el método hasData() y a continuación se desarrolla el código de manera correcta.

44. Do the comments help in understanding the code?
La ayuda de los comentarios es muy útil para entender el código de cada clase y sus métodos.

45. Are there enough comments in the code
Sí, hay suficientes comentarios como para poder entender bien el código.

46. Are there too many comments in the code?
No, en todas las clases podemos encontrar comentarios, pero los consideramos necesarios para el correcto entendimiento del código.

10. Packaging Defects (LP)

47. For each file: Does it contain only one class?
No, en el archivo Agenda.java, hay contenidas dos clases la Clase AgendaNode (línea 12) y en la clase Agenda (linea 22)

48. For each function: Is it no more than about 60 lines long?
No hay ninguna función con más de 60 líneas. La función más larga es de 23 líneas, que corresponde a loadAgenda(), de la clase Agenda

49. For each class: Is no more than 2000 lines long (Sun Coding Standard) ?
No hay ninguna clase que tenga más de 2000 lineas. La clase más larga de nuestro programa Agenda tiene 129 líneas (clase Entry).

11. Modularity Defects (MO)

50. Is there a low level of coupling between packages (classes)?
Existe un bajo acoplamiento entre clases. En este caso solo hay una dependencia de datos, entre la clase AgendaNode y la clase Agenda, que implementa un objeto de la clase anterior. La clase Agenda no funcionaría sin el objeto AgendaNode, pero AgendaNode funcionaría tanto si hay cambios en Agenda como si no se viera declarada en esa clase. Lineas de la 12 a la 23 de la Clase Agenda.

51. Is there a high level of cohesion within each package?
Existe una alta cohesión, pues cada módulo (clase) tiene una funcionalidad clara y definida y su alcance abarca la implementación de la clase.

52. Is there duplicate code that could be replaced by a call to a function that provides the behavior of the duplicate code?
No, no existe en ninguna clase, ninguna línea o función de código duplicado o que haga funciones similares.

53. Are framework classes used where and when appropriate?
No existen clases framework en este caso.

3.2. Reading by Abstration

3.2.1. Reading by Abstraction Dijkstra

Línea 11 private Double[][] adjMatrix;	//Adjacency matrix	Declara la matriz de proximidad
Línea 19 private boolean dijkstraExec;	//Flag: if the algorithm was executed	Declara una variable booleana que indica si el algoritmo ha sido ejecutado
Línea 75 Double cost = Double.POSITIVE_INFINITY;		Declara la variable coste con un valor infinito positivo para que, más tarde al comparar con el primer valor de distancia, distancia sea siempre más pequeño
Línea 111 Integer cur = ini;	//Starts in the initial vertex	Declara la variable cur (current) para controlar el vertice que estamos tratando en ese momento y lo inicializa con el primero
Línea 112 distances[ini]=0.0;	//The cost of the path from ini to itself is 0	La coste de la distancia de un punto a sí mismo es 0

Línea 57 - 61
for(int i=0; i<nVertices; i++) {
distances[i] = Double.POSITIVE_INFINITY;
visited[i] = false;
prev[i] = -1;
}
	// Initialization of data structures
	//The estimated distance is infinity
	//No node has been visited
	//No shortest path
		Este bucle for se encarga de inicializar todas las estructuras de datos con tantos elementos en el array como numero de vertices hay.
		Las distancias las inicializa a infinito, si están visitados o no a false ya que todavía no ha pasado por ninguno y el anterior a -1, ya que no hay ningún anterior.

Línea 77 - 82
for(int i=0; i<nVertices; i++) {
if(!visited[i] && distances[i]<cost) {
next = i;
cost = distances[i];
}
}
		Es un for que pasa por todos los vertices.
		Deentrio encontramos un if (línea 78) que, para ejecutarse, tiene que cumplir que el vértice no haya sido visitado y que la el coste de la distancia sea menor que el actual.
		Si esto sucede el vertice i será asignado a la variable next y el coste de la distancia del vértice i a la variable cost.

Línea 115 - 128
for(int neigh=0; neigh<nVertices; neigh++) {
if(adjMatrix[cur][neigh]>0 && !visited[neigh]) {
newDistance = distances[cur] + adjMatrix[cur][neigh];
if(distances[neigh] > newDistance) {
distances[neigh] = newDistance;
prev[neigh] = cur;
}
}
}
	//If the vertex is connected with the current node and
	//has not been visited
	//Computes the distance of going to neigh via cur.
	//If the new distance is better, update it.
		Es un for que pasa por todos los vértices.
		Si el vértice a comparar está conectado con le vertice actual, suma a la distancia la distancia de esta nueva unión y comprueba si esta distancia es mayor que el coste de la distancia del nodo, en este caso, a la distancia del nodo actual le asigna el valor de la nueva distancia y señala que el nodo previa del actual es el nodo que estamos comparando.

Línea 113 -132
while(!visited[end]) {
for(int neigh=0; neigh<nVertices; neigh++) {
if(adjMatrix[cur][neigh]>0 && !visited[neigh]) {
newDistance = distances[cur] + adjMatrix[cur][neigh];
if(distances[neigh] > newDistance) {
distances[neigh] = newDistance;
prev[neigh] = cur;
}
}
}
 	
visited[cur] = true;
cur = nextCur();
}
		Este bucle indica que mientras que no se lllegue al último nodo, se realiza el bucle descrito en el anterior punto. Una vez se realiza el vector actual pasa a estado de visitado y se asigna el siguiente vector

Línea 154 - 157
if(!dijkstraExec) {
error = true;
return null;
}
	* If Dijkstra's algorithm has not been previously executed,
	* the function returns null and sets that an error occurred.
		Si el algoritmo no ha sido ejecutado anteriormente devuelve null y pone el valor error como verdadero

Línea 165-168
while(cur != ini) {
path.add(prev[cur]);
cur = prev[cur];
}
		Es un bucle que, mientras que el vertice actual es disitnto del inicial, añade ese vertice a un ArrayList y el siguiente pasa a ser el anterior a este

Línea 41 - 44
public Dijkstra(Double[][] adjMatrix, Integer nVertices){
this.adjMatrix = adjMatrix;
this.nVertices = nVertices;
}
	/**
	* Constructor of the Dijkstra class.
	* This function saves the adjacency matrix of a
	* directed simple graph with weighted edges.
	*
	* The nodes are labeled by their position in the
	* adjacency matrix. That is, the vertex whose
	* outgoing edges are saved in the ith row of adjMatrix
	* is the vertex i.
	*
	* @param adjMatrix Square matrix that represents
	* the adjacency matrix of the graph. The entry (i,j)
	* of adjMatrix represents the weight of the edge that
	* goes from vertex i to vertex j. The weights must be
	* positive. A negative entry in the position (i,j)
	* means that there is no edge between the vertices i and j.
	*
	* @param nVertices Non-negative number of vertices of the graph,
	* that is, the order of the adjacency matrix.
	*/
		El constructor de la clase, asigna a la variable de la clase adjMatrix la matriz que se indica en la llamada al método, del mismo modo que asigna el valor a nVertices del Integer enviado en la llamada del método.

Línea 50 - 63
private void initializeDataStructures() {
 	
distances = new Double[nVertices];
visited = new boolean[nVertices];
prev = new Integer[nVertices];
 	
for(int i=0; i<nVertices; i++) {
distances[i] = Double.POSITIVE_INFINITY;
visited[i] = false;
prev[i] = -1;
}
 	
return;
}
	/**
	* Private function that initializes all the data
	* structures as required by the Dijkstra's algorithm.
	*/
		Esta función inicializa todas las estrucutras de datos necesarias para la ejecución del algoritmo.

Línea 72 - 84
private Integer nextCur() {
Integer next = -1;
Double cost = Double.POSITIVE_INFINITY;
 	
for(int i=0; i<nVertices; i++) {
if(!visited[i] && distances[i]<cost) {
next = i;
cost = distances[i];
}
}
 	
return next;
}
	/**
	* Returns the next vertex to be analyzed by the
	* Dijkstra algorithm. As required, that should
	* be the unvisited vertex with smallest expected cost.
	*
	* @return Next vertex to be explored.
	*/
		Esta función devuelve el siguiente vértice a analizar.
		Este vértice será un vértice no visitado cuyo coste sea el menor

Líneas 106 - 136
public Double computeShortestPath(Integer ini, Integer end) {
Double newDistance;
initializeDataStructures();
Integer cur = ini;
distances[ini]=0.0;
while(!visited[end]) {
for(int neigh=0; neigh<nVertices; neigh++) {
if(adjMatrix[cur][neigh]>0 && !visited[neigh]) {
newDistance = distances[cur] + adjMatrix[cur][neigh];
if(distances[neigh] > newDistance) {
distances[neigh] = newDistance;
prev[neigh] = cur;
}
}
}
visited[cur] = true;
cur = nextCur();
}
dijkstraExec = true;
return distances[end];
	/**
	* Main call of Dijkstra's shortest path algorithm.
	* Given a directed simple graph with non-negative
	* weighted edges, an initial vertex ini and a
	* final vertex end, this function computes the
	* shortest path between ini and end.
	*
	* The minimum distance is returned by the function.
	* The list of nodes that form the shortest path
	* can be recovered by calling the function getPath
	* after the call to this function.
	*
	* If there is no path between ini and end, the function
	* returns Double.POSITIVE_INFINITY and no shortest path
	* is computed.
	*
	* @param ini Initial vertex
	* @param end Final vertex
	* @return Cost of the shortest path between ini and end.
	*/
		Dado el vertice inicial y final de un grafo esta función, realiza el algoritmo de Dijkstra y devuelve el coste del camino más corot entre ini y end.

Línea 152 - 170
public ArrayList<Integer> getPath(Integer ini, Integer end) {
if(!dijkstraExec) {
error = true;
return null;
}
ArrayList<Integer> path = new ArrayList<Integer>();
 	
Integer cur = end;
path.add(cur);
while(cur != ini) {
path.add(prev[cur]);
cur = prev[cur];
}
 	
return path;
}
	/**
	* Get the shortest path computed by Dijkstra's algorithm.
	* Returns an ArrayList with the ordered list of nodes that
	* form the path, from the initial vertex to the final vertex.
	*
	* If there are two possible paths with minimum cost, this
	* function returns the one with less number of edges.
	*
	* If Dijkstra's algorithm has not been previously executed,
	* the function returns null and sets that an error occurred.
	*
	* @param ini Initial vertex of Dijkstra's algorithm
	* @param end Final vertex of Dijkstra's algorithm
	* @return ArrayList with the shortest path.
	*/
		Dado el camino más corto entre dos punto pr el algoritmo de Dijkstra, este méotdo devuelve un array ordenado desde el vértice final al inicial, es decir los vértices de forma decreciente.
		Si hay dos algoritmos con caminos de coste mínimo devolvera el que tenga menos vértices.

public class Dijkstra	/** * Class implementing Dijkstra's shortest path algorithm * for directed simple (i.e. without loops) graphs * with non-negative weighted edges. * * For more info: https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm */	La clase implementa le algoritmo de Dijkstrta para encontrar el camino mas corto para grafos simples y sin valores negativos

** Estas son capturas extraída del documento LearningByAnsDijkstra.xlsx encontrado en el git dentro de la carpeta VVroche/Análisis estático

3.2.2. Reading by Abstration Agenda

public boolean addEntry(Entry p) { AgendaNode cur;
boolean found = false;
 	
if (first != null) {
cur = first;
while (cur != null && !found) {
if (cur.info.getName() == p.getName()) {
found = true;
}
 	
cur = cur.sig;
}
}
 	
if (!found) {
first = new AgendaNode(p, first);
return true;
} else
return false;
}
	Esta función añadirá un nuevo contacto a la agenda. Almacenará los contactos en una lista por Nombre y apellidos, si la agenda esta vacía, el contacto será el primero de la lista. La función debe devolver TRUE si el contacto se ha añadido correctamente y falso si los datos están duplicados	El código utiliza un boolean found=false como centinela, sin embargo solo contrasta los datos del nombre y no de los apellidos. Además no devuelve falso si los datos están duplicados

public boolean removeEntry(String name) {
if (first == null) {
return false;
}
 	
AgendaNode prev = first;
while (prev.sig != null && prev.sig.info.getName() != name)
{
prev = prev.sig;
}
 	
if (prev.sig == null) {
return false;
} else {
prev.sig = prev.sig.sig;
numEntries--;
return true;
}
}
	Esta función debe borrar un contacto de la agenda. Recibe el nombre de un contacto que debe ser eliminado. Si la agenda esta vacía la función no hace nada, si la agenda tiene un solo contacto la agenda al eliminarlo deberá quedar vacía. La función devolverá TRUE si se ha eliminado correctamente y FALSE si no.	El código devuelve falso si la agenda esta vacía. Si encuentra el nombre que quiere eliminar reduce en uno el numero de contactos de la agenda, devolverá false si no lo ha encontrado

public Entry removeFirst() {
Entry p = null;
 	
if (first != null) {
p = first.info;
first = first.sig;
numEntries--;
}
 	
return p;
}
 	
public int nEntries() {
return numEntries;
}
 	
public boolean isEmpty() {
if (first == null) {
return true;
} else {
return false;
}
}
	Esta función eliminará el primer contacto de la agenda. Si la agenda esta vacía la función no hara nada en la lista, si solo existe un contacto, se eliminará y la lista quedará vacía. Devolverá a Entry object si se ha realizado con éxito y nulo si ocurre lo contrario	El código comprueba si la agenda esta vacía, si no exite ningún contacto devolverá un valor nulo, si no lo está eliminará el primer contacto y devolverá el entry object.

public boolean saveAgenda() throws IOException {
AgendaNode cur = first;
String line;
boolean success = false;
Parser p = new Parser();
 	
FileWriter fichero = new FileWriter("agendofile.txt");
BufferedWriter bufferescritura = new
BufferedWriter(fichero);
PrintWriter output = new PrintWriter(bufferescritura);
 	
while (cur != null) {
if (!success) {
success = true;
}
p.insertEntry(cur.info);
line = p.getLine();
output.println(line);
}
 	
return success;
}
	La función guardará la agenda en un .txt denominado agendafile.txt. Si la agenda esta vacía no creara ningún archivo. Deberá preguntar para guardar los cambios. Devolverá TRUE si se ha guardado correctamente y FALSE en cualquier otro caso	El código crea un txt distinto llamado agendo.txt. Además no comprueba si se desean guardar los cambios y no devuelve FALSE en ningún caso

public boolean loadAgenda() throws IOException {
FileReader filein = new FileReader("agendafile.txt");
BufferedReader bufferin = new BufferedReader(filein);
 	
Parser p = new Parser();
String cad;
if ((cad = bufferin.readLine()) == null) {
bufferin.close();
return false;
}
 	
do {
System.out.println(cad);
p.insertLine(cad);
Entry auxEntry = p.getEntry();
if (auxEntry.hasData()) {
addEntry(auxEntry);
}
} while ((cad = bufferin.readLine()) != null);
filein.close();
 	
return true;
}
}
	La función cargará la agenda del archivo agendafile.txt. Añadirá todas las entradas en el archivo. Si la agenda no está vacía todos los
	los contactos preexistentes se eliminan y la agenda final solo contiene las entradas del archivo. Si el archivo agendafile.txt está vacío o no existe, la agenda permanece sin cambios.
	La función devuelve verdadero si el archivo existe y falso en caso contrario
		El código comprueba si la agenda cargada esta vacía, de ser así devolverá false y la cerrará. Además añadirá los contactos de no estar vacía, eliminando los preexistentes.

** Estas son capturas extraída del documento LearningByAnsAgenda.xlsx encontrado en el git dentro de la carpeta VVroche/Análisis estático

3.2.3. Reading by Abstration Entry

public class Entry	/** * Entry of the agenda. Contains information * of the name, surname, address, city, * county, zip, telephone and year of birth * of the contact. * * These fields are inicialized as empty (birthyear as 0) * and can be set using the corresponding set methods. * * An Entry is considered to have data if the name * and surname of the contact are not empty. * */	La clase Entry contiene información relativa a un contacto de la agenda como el nombre, el apellido, país…

Líneas 43-53
public boolean hasData ()
{
if (!name.equals("") && !surname.equals(""))
{
return true;
}
else
{
return false;
}
}
	/**
	* Checks if the Entry has data, that is, if
	* the name and surname of the entry are not empty.
	*
	* @return True if the Entry has data, false otherwise.
	*/
		Comprueba si un objeto de a case Entry tiene datos, es decir, si tiene nombre y apellidos. Devuelve true si es que si y fase si no.

** Estas son capturas extraída del documento LearningByAnsEntry.xlsx encontrado en el git dentro de la carpeta VVroche/Análisis estático

3.2.4. Reading by Abstration Parser

Líneas 35-38
public void insertEntry (Entry p) {
en = p;
line = this.createLine ();
}
		Asinga un objeto Entry, 'p' pasado como parámetro, a la variabe 'en' y asigna a la variable 'line' el resutado de ejecutar el método createLine()

Líneas 40-43
public void insertLine (String l) {
line = l;
en = this.createEntry ();
}
		Asinga el String 'l' pasado como parámetro a la variabe 'line' y asigna a la variable 'en' el resutado de ejecutar el método createEntry()

Líneas 53-99
private Entry createEntry () {
String nombre = new String ();
String apellido = new String ();
String direccion = new String ();
String poblacion = new String ();
String provincia = new String ();
String codigo = new String ();
String telefono = new String ();
int anno = 0;
int posNom,posApe,posDir,posPob,
posProv,posCod,posTfno,posAnno,longitud;
boolean auxb = true;
Entry auxp = new Entry ();
 	
longitud = line.length ();
posNom = line.indexOf (NOM);
posApe = line.indexOf (APE);
posDir = line.indexOf (DIR);
posPob = line.indexOf (POB);
posProv = line.indexOf (PRO);
posCod = line.indexOf (COD);
posTfno = line.indexOf (TFN);
posAnno = line.indexOf (ANNO);
 	
if ((posNom != -1) && (posApe != -1) && (posDir != -1) &&
(posPob != -1) && (posCod != -1) && (posTfno != -1) &&
		Esta función crea y devuelve un objeto Entry con los mismo datos que un objeto String que es una variabe de la clase Parser.

Líneas 101-118
private String createLine() {
String nombre = en. getName ();
String apellido = en.getSurname ();
String direccion = en.getAddress ();
String poblacion = en.getCity ();
String provincia = en.getCounty ();
String codigo = en.getZip ();
String telefono = en.getTelephone ();
int anno = en.getBirthYear ();
String aux2 = "";
 	
if (en.hasData()) {
aux2 = NOM + " " + nombre + ". " + APE + " " + apellido + ". " + DIR + " " + direccion + ". ";
aux2 = aux2 + POB + " " + poblacion + ". " + PRO + " " + provincia + ". " + COD + " " + codigo + ". " ;
aux2 = aux2 + TFN + " " + telefono + ". " + ANNO + " " + anno;
}
return aux2;
}
		Función que devuelve un String concatenando toda la información(nombre,apellido,dirección,población,provincia,código,teléfono y anno) que se encuentra en el objeto 'en' de la clase Entry en el caso en que disponga de dicha información, sino devuelve un String vacío.

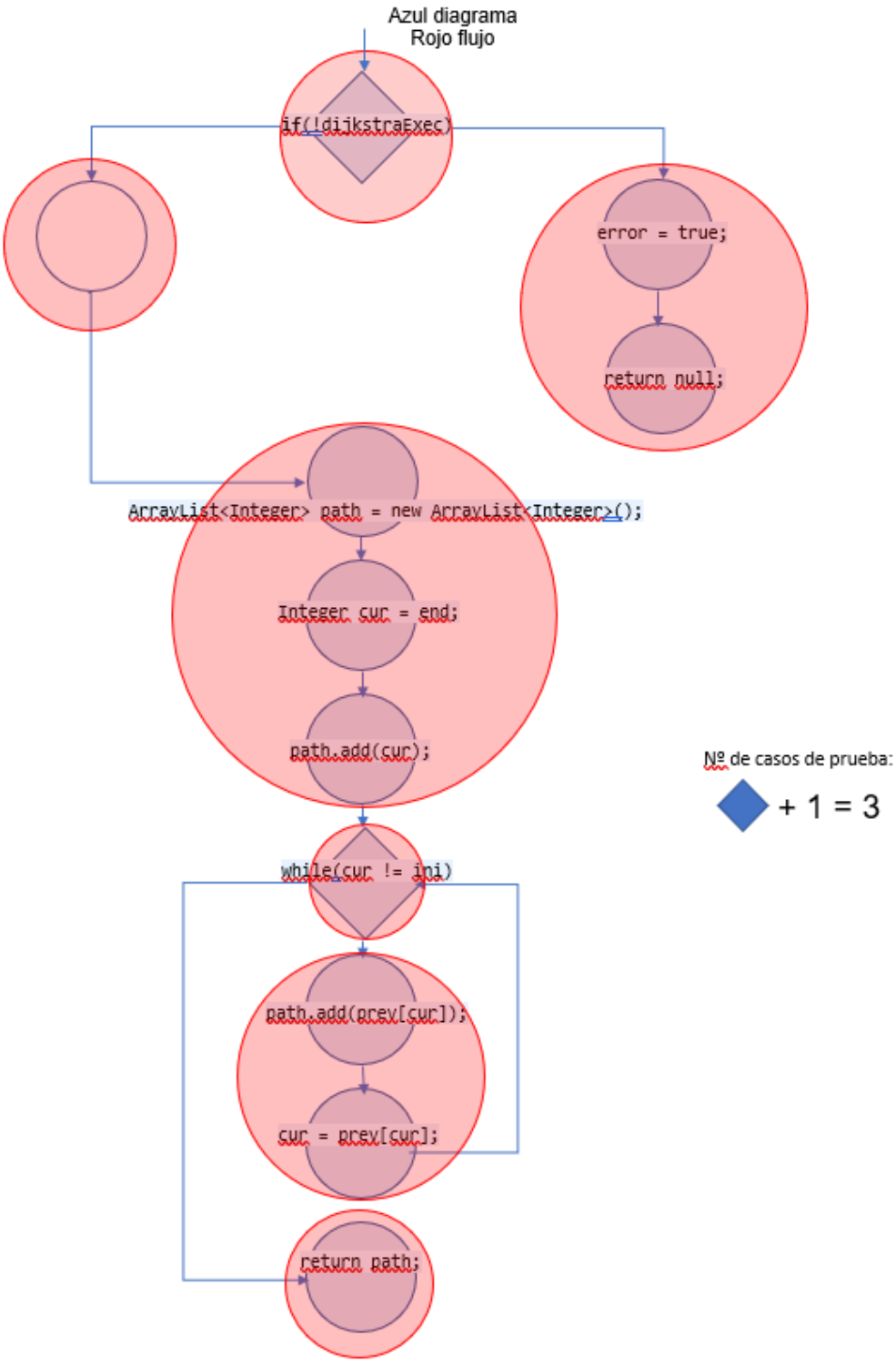
public class Parser	/** * Auxiliar class for persistence in text files * of the class Agenda. * * A Parser translates between Entry objects and * lines of the text file (String objects). * * If a Entry is given via the method insertEntry, * the Parser translates it into a line that can * be recovered with the getLine method. * * Analogously, given a String via the method * insertLine, the Parser creates a Entry object * with the same data that can be recovered * with the getEntry method. */	Explica perfectamente el funcionamiento de la clase Parser.Realiza a traducción entre objetos de la clase Entry y Strings (utiizando el método getLine) y viceversa (utilizando el método getEntry).

** Estas son capturas extraída del documento LearningByAnsParser.xlsx encontrado en el git dentro de la carpeta VVroche/Análisis estático

4. Práctica 3

4.1. Caja blanca método GetPath()

Diagrama y grafo de flujo



Código del método con las variaciones necesarias:

```
155     public ArrayList<Integer> getPath(Integer ini, Integer end) {
156         line = "";
157         line += "155 "; if(!dijkstraExec) {
158             error = true; line += "156 ";
159             line += "157 "; return null;
160         }
161 
162         ArrayList<Integer> path = new ArrayList<Integer>(); line += "160 ";
163 
164         //Runs over the path from end to ini by
165         //jumping through the previous nodes.
166         Integer cur = end; line += "164 ";
167         path.add(cur); line += "165 ";
168         line += "166 "; while(cur != ini) {
169             path.add(prev[cur]); line += "167 ";
170             cur = prev[cur]; line += "168 ";
171         }
172 
173         line += "171 "; return path;
174     }
```

Código de los test:

```
 9  class DijkstraTest {
10 
11      static Dijkstra dijkstra,dijkstra3;
12 
13      @BeforeAll
14      static void setUpBeforeClass() throws Exception {
15          /*Path 1 y 2*/
16                  Double [][] mpath = new Double[3][3];
17                  for(int i=0; i<3;i++) {
18                      for(int j=0; j<3;j++)
19                          mpath[i][j]= Double.POSITIVE_INFINITY;
20                  }
21                  mpath[0][1]=2.0;mpath[1][2]=1.0;mpath[0][2]=5.0;
22                  Integer numV = new Integer(3);
23                  dijkstra = new Dijkstra(mpath,numV);
24                  dijkstra.computeShortestPath(0, 2);
25          //Path 3
26                  Double [][] mpath3 = new Double[1][1];
27                  mpath3[0][0]= Double.POSITIVE_INFINITY;
28                  Integer numV3 = new Integer(1);
29                  dijkstra3 = new Dijkstra(mpath3,numV3);
30      }
31 
```

```
44      @Test
45      void testGetPath1() {
46          /*PATH1:
47           *Es un grafo ya ejecutado por lo que en el primero rombo va a la izquierda
48           *pero que no entra en el bucle while*/
49          dijkstra.getPath(0, 0);
50          assertEquals("155 160 164 165 166 171 ", dijkstra.line);
51      }
52
53      @Test
54      void testGetPath2() {
55          /*PATH2:
56           *Es un grafo ya ejecutado por lo que en el primero rombo va a la izquierda
57           *pero que entra en el bucle while*/
58          dijkstra.getPath(0, 2);
59          assertEquals("155 160 164 165 166 167 168 167 168 171 ", dijkstra.line);
60      }
61
62      @Test
63      void testGetPath3() {
64          /*PATH3:
65           *Es un grafo no ejecutado por lo que en el primer rombo va a la derecha
66           *y ahí finaliza*/
67          dijkstra3.getPath(new Integer(0), new Integer(0));
68          assertEquals("155 156 157 ", dijkstra3.line);
69      }
70
71  }
```

En la parte de BeforeAll creamos dos matrices y todo lo necesario para los casos de prueba. Las pruebas 1 y 2 compartirán los datos y la prueba 3 tendrá los suyos propios. Esto es debido a que necesitamos hacerlo así para ser fieles al orden de los caminos, siendo el 1 el que deriva en todos los rombos a la izquierda, el 2 el primero a la izquierda y el segundo a la derecha y el tercero el primero a la derecha, cubriendo así todos los posibles caminos.

** Estas son capturas extraída de los documentos encontrados en el git dentro en la carpeta VVroche/Caja Blanca