Informe de Laboratorio 04

Tema: Arreglos de Objetos, Búsqueda y Ordenamiento de Burbuja

Nota

Estudiante	Escuela	Asignatura
Jhonatan David Arias Quispe jariasq@unsa.edu.pe	Escuela Profesional de Ingeniería de Sistemas	Fundamentos de Programacion 2 Semestre: II Código: 1701213

Laboratorio	Tema	Duración
04	Arreglos de Objetos, Búsqueda y Ordenamiento de Burbuja	04 horas

Semestre académico	Fecha de inicio	Fecha de entrega
2023 - B	Del 20 Setiembre 2023	Al 28 Setiembre 2023

1. Tarea

- Analice, complete y pruebe el Código de la clase DemoBatalla
- Solucionar la Actividad 4 de la Práctica 1 pero usando arreglo de objetos
- Solucionar la Actividad 5 de la Práctica 1 pero usando arreglo de objetos

2. Equipos, materiales y temas utilizados

- Sistema Operativo ArchCraft GNU Linux 64 bits Kernell
- NeoVim
- OpenJDK 64-Bit 20.0.1
- Git 2.42.0
- Cuenta en GitHub con el correo institucional.
- Programación Orientada a Objetos.
- Creacion de programas con CLI

3. URL de Repositorio Github

- URL del Repositorio GitHub para clonar o recuperar.
- `https://github.com/JhonatanDczel/fp2-23b.git`
- URL para el laboratorio 04 en el Repositorio GitHub.
- `https://github.com/JhonatanDczel/fp2-23b/tree/main/fase01/lab04`

4. Documentacion

5. Desarrollo de la actividad

5.1. Clase Reader.java

5.1.1. Commits principales

5.2. Quicksort

- El método «Quicksort», también conocido como ordenación rápida, es uno de los algoritmos de ordenación más eficientes y ampliamente utilizados en la informática.
- Se basa en el paradigma de divíde y vencerás. Consiste en seleccionar un elemento de la lista, llamado "pivote,ż reorganizar los elementos en la lista de manera que los elementos menores que el pivote estén a su izquierda, y los elementos mayores estén a su derecha.

- Cuando se elige un pivote adecuado, «Quicksort» puede alcanzar su rendimiento óptimo, que es O(n log n) en promedio. Sin embargo, en el peor de los casos, se puede degradar a O(n ^{2}).

5.2.1. Commits de la implementación de Quicksort

- **Hash: afde7b901d3ac3c8414a3d6976fde97bcbe9729d**
- En el presente commit se implementó el quicksort para ordenar los CUI, sin embargo cuenta con un error. Cambia la referencia del atributo cui de la clase Student, sin embargo, no cambia la la referencia del objeto mismo, es un error que se solucionará más adelante.

Listing 1: Commit: Se implementó Quicksort para cui

```java
public static void Cui (Reader.Student[] s, int left, int right) {
    int i = left;
    int j = right;
    int aux;
    while (i < j) {
        while (s[i].getCui() <= piv && i < j) i++;
        while (s[j].getCui() > piv) j--;
        if (i < j) {
            aux = s[i].getCui();
            s[i].setCui(s[j].getCui());
            s[j].setCui(aux);
        }
     }
    s[left].setCui(s[j].getCui());
    s[j].setCui(piv);
    if (left < j - 1) Cui(s, left, j -1);
    if (j + 1< right) Cui(s, j + 1, right);

 }
```

- **Hash: c9e70896b8edb82f963375cac9a4cb72d67ed6fb**
- En el presente commit de manera similar se implementó el método email para ordenar según la primera letra a los correos, que están representados mediante Strings, sin embargo, presenta el error de solo ordenar la primera letra. Más adelante se parchará el error.

Listing 2: Commit: Se añadió el método email para ordenar los correos según quicksort

```
1
2    public static void email (Reader.Student[] s, int left, int right) {
3        char piv = s[left].getEmail().charAt(0);
4        int i = left;
5        int j = right;
6        String aux;
7        while (i < j) {
8            while (s[i].getEmail().charAt(0) <= piv && i < j) i++;
9            while (s[j].getEmail().charAt(0) > piv) j--;
10           if (i < j) {
11               aux = s[i].getEmail();
12               s[i].setEmail(s[j].getEmail());
13               s[j].setEmail(aux);
14           }
15       }
16       aux = s[left].getEmail();
17       s[left].setEmail(s[j].getEmail());
18       s[j].setEmail(aux);
19       if (left < j - 1) email(s, left, j -1);
20       if (j + 1< right) email(s, j + 1, right);
21    }
```

- **Hash: cb4418c7361dc0e2da32042698b2400e79beebfd**
- Se creó la fución «smallerThan()» en la cual se usa el método String.compareTO(String), la función de esta misma es restar los caracteres según su orden ascii. Si w1 - w2 es positivo, significa que w1 debería estar después de w2.
- Esta función se utiliza cada vez que se quiere comparar dos String, solucionando el error mencionado anteriormente.

Listing 3: Commit: Se mejoró los métodos que ordenan Strings

```
1    public static boolean smallerThan(String w1, String w2) {
2        w1.toUpperCase();
3        w2.toUpperCase();
4        if (w1.compareTo(w2) <= 0) return true;
5        return false;
6
7    }
8
9  // Uso de smallerThan en otras funciones:
10
11   public static void lastNameM (Reader.Student[] s, int left, int right) {
12       String piv = s[left].getLastNameM();
13       int i = left;
14       int j = right;
15       String aux;
16       while (i < j) {
17           while (smallerThan(s[i].getLastNameM(), piv) && i < j) i++;
18           while (!smallerThan(s[j].getLastNameM(), piv)) j--;
19           if (i < j) {
20               aux = s[i].getLastNameM();
```

```
            s[i].setLastNameM(s[j].getLastNameM());
            s[j].setLastNameM(aux);
        }
    }
    aux = s[left].getLastNameM();
    s[left].setLastNameM(s[j].getLastNameM());
    s[j].setLastNameM(aux);
    if (left < j - 1) lastNameM (s, left, j -1);
    if (j + 1< right) lastNameM(s, j + 1, right);
}
```

- **Hash: 0c89058320e8f2dc1981a4666c3deb86df450702**
- Se solucioó el primer error mencionado, en esta oportunidad, el código intercambia la referencia de los objetos en el array en vez de sus atributos.

Listing 4: Commit: Se solucionó error de cambio de valores de atributos en vez de las referencias de los objetos

```
public static void email (Reader.Student[] s, int left, int right) {
    String piv = s[left].getEmail();
    int i = left;
    int j = right;
    Reader.Student aux;
    while (i < j) {
        while (smallerThan(s[i].getEmail(), piv) && i < j) i++;
        while (!smallerThan(s[j].getEmail(), piv)) j--;
        if (i < j) {
            aux = s[i];
            s[i] = s[j];
            s[j] = aux;
        }
    }
    aux = s[left];
    s[left] = s[j];
    s[j] = (aux);
    if (left < j - 1) email(s, left, j -1);
    if (j + 1< right) email(s, j + 1, right);
 }
```

- Hash:
-

Listing 5: TITULOOOOOO

5.3. Busqueda Binaria

- La búsqueda binaria es un algoritmo de búsqueda eficiente utilizado para encontrar un elemento específico en una lista ordenada.
- Tanto la versión recursiva como la iterativa de la búsqueda binaria se basan en el mismo principio: dividir y vencerás.
- En la búsqueda binaria recursiva, se divide repetidamente la lista en dos mitades y se compara el elemento buscado con el elemento en el medio. Este proceso se repite de manera recursiva hasta que se encuentre el elemento deseado

- La búsqueda binaria iterativa se implementa mediante un bucle en lugar de una función recursiva.

5.3.1. Forma iterativa

- **Hash: 55e501872e486ef65d414a41aeb9f673c6b8318e**
- Se creo «IterativeBinarySearch.java» donde se implementó el método cui() para realizar la búsqueda binaria de manera iterativa.

Listing 6: Commit: Se creó el método cui para la búsqueda binaria

```
public static int cui (Reader.Student [] s, int x) {
  int l, r;
  l = 0;
  r = s.length - 1;
  while (l <= r) {
    int m = l + ( r - l ) / 2;
    if (s[m].getCui() == x) return m;
    if (s[m].getCui() < x) l = m + 1;
    else r = m - 1;
  }
  return -1;
}
```

- **Hash: b90e4fbe4c4373de351edf016e255e9534088251**
- De manera similar, se implementó el método « email() » usando como comparador la función «smallerThan()».
- Usando esta última función sabemos si la palabra buscada debería estar al delante o atrás.
- Usando este método que funciona con Strings, también se implementó el resto de funciones según los atributos de la clase Student.

Listing 7: Commit: Implementación para busqueda según email

```
public static int email (Reader.Student [] s, String x) {
  int l = 0, r = s.length - 1;
  while (l <= r) {
    int m = l + (r - l) / 2;
    if (s[m].getEmail().equals(x)) return m;
    if (smallerThan(s[m].getEmail(), x)) l = m + 1;
    else r = m - 1;
  }
  return -1;
}

public static boolean smallerThan(String w1, String w2) {
    w1.toUpperCase();
    w2.toUpperCase();
    if (w1.compareTo(w2) <= 0) return true;
    return false;

  }

}
```

5.3.2. Forma recursiva

- **Hash: 2318dc5647d46b889c72f61a09a9fd7b8f8f2f5a**
- Posteriormente se creo el archivo «RecursiveBinarySearch.java» donde se implementaron las siguientes clases: «cui()» y «email()».
- Estas clases usan el algoritmo de la búsqueda binaria recursiva. Se reutiliza la función «smallerThan()» para comparar Strings.
- De manera análoga se implementa al resto de atributos de la clase Student que están basados en Strings, los cuales no se muestran en el presente informe para evitar una extensión innecesaria del mismo.

Listing 8: Commit: Se creó la búsqueda binaria con los métodos de cui y email

```java
package algorithms;

import reader.Reader;

public class RecursiveBinarySearch {
  public static int cui (Reader.Student [] s, int x, int l, int r) {
    if (r >= l) {
      int m = l + (r - l) / 2;
      if (s[m].getCui() == x) return m;
      if (s[m].getCui() > x) return cui(s, x, l, m -1);
      return cui(s, x, m + 1, r);
    }
    return -1;
  }
  public static int email (Reader.Student [] s, String x, int l, int r) {
    if (r >= l) {
      int m = l + (r - l) / 2;
      if (s[m].getEmail().equals(x)) return m;
      if ( smallerThan(s[m].getEmail(), x) ) return email(s, x, m + 1, r);
      return email(s, x, l, m - 1);
    }
    return -1;
  }
  public static boolean smallerThan(String w1, String w2) {
      w1.toUpperCase();
      w2.toUpperCase();
      if (w1.compareTo(w2) <= 0) return true;
      return false;

    }

}
```

5.4. Parte grafica

5.4.1. Commits principales

6. Ejecucion del programa

- A continuacion veremos la ejecucion del programa en la terminal:

7. Rúbricas

7.1. Rúbrica para el contenido del Informe y demostración

- El alumno debe marcar o dejar en blanco en celdas de la columna **Checklist** si cumplio con el ítem correspondiente.
- El alumno debe autocalificarse en la columna **Estudiante** de acuerdo a la siguiente tabla:

Tabla 1: Niveles de desempeño

	Nivel			
Puntos	Insatisfactorio 25 %	En Proceso 50 %	Satisfactorio 75 %	Sobresaliente 100 %
2.0	0.5	1.0	1.5	2.0
4.0	1.0	2.0	3.0	4.0

Tabla 2: Rúbrica para contenido del Informe y demostración

Contenido y demostración		Puntos	Checklist	Estudiante	Profesor
1. GitHub	Hay enlace URL activo del directorio para el laboratorio hacia su repositorio GitHub con código fuente terminado y fácil de revisar.	2	X	2	
2. Commits	Hay capturas de pantalla de los commits más importantes con sus explicaciones detalladas. (El profesor puede preguntar para refrendar calificación).	4	X	4	
3. Código fuente	Hay porciones de código fuente importantes con numeración y explicaciones detalladas de sus funciones.	2	X	2	
4. Ejecución	Se incluyen ejecuciones/pruebas del código fuente explicadas gradualmente.	2	X	2	
5. Pregunta	Se responde con completitud a la pregunta formulada en la tarea. (El profesor puede preguntar para refrendar calificación).	2	X	1	
6. Fechas	Las fechas de modificación del código fuente estan dentro de los plazos de fecha de entrega establecidos.	2	X	2	
7. Ortografía	El documento no muestra errores ortográficos.	2	X	1	
8. Madurez	El Informe muestra de manera general una evolución de la madurez del código fuente, explicaciones puntuales pero precisas y un acabado impecable. (El profesor puede preguntar para refrendar calificación).	4	X	3	
Total		20		17	