Informe de Laboratorio 02

Tema: Técnicas y diseño de algoritmos

Nota

Estudiante	Escuela	Asignatura
Reyser Julio Zapata Butrón rzapata@unsa.edu.pe	Escuela Profesional de Ingeniería de Sistemas	Estructura de datos y Algoritmos Semestre: II Código: 1702124

Laboratorio	Tema	Duración
02	Técnicas y diseño de algoritmos	04 horas

Semestre académico	Fecha de inicio	Fecha de entrega
2023 - A	25 septiembre 2023	04 octubre 2023

1. URL de Repositorio Github

- URL para el laboratorio 02 en el Repositorio GitHub.
- `https://github.com/ReyserLyn/eda-lab02.git`

2. Ejercicios designados

2.1. Cuadrado Perfecto

Este programa Java llamado `Rec_square_perfect` se encarga de determinar si un número dado es un cuadrado perfecto de forma recursiva. Un cuadrado perfecto es un número que es el resultado de elevar un número entero a otro número entero. Por ejemplo, 4, 9, 16 son cuadrados perfectos porque son el resultado de elevar 2, 3 y 4 al cuadrado, respectivamente.

Listing 1: Rec_square_perfect.java

```java
public class Rec_square_perfect {
  public static void main(String[] args) {
    int num = (Integer.parseInt(args[0]) >= 0) ? Integer.parseInt(args[0]) : 0;

    boolean result = isSquarePerfectRecursive(num, 0);

    System.out.println(result);
```

```
  }

  public static boolean isSquarePerfectRecursive(int x, int init) {
    int square = init * init;

    if (square == x){
      return true;
    } else if (square > x) {
      return false;
    } else {
      return isSquarePerfectRecursive(x, init + 1);
    }
  }
}
```

1. `public class Rec_square_perfect {`: Aquí comienza la definición de la clase `Rec_square_perfect`.
2. `public static void main(String[] args) {`: Esto es el método `main`, que es el punto de entrada del programa.
3. `int num = (Integer.parseInt(args[0]) >= 0) ? Integer.parseInt(args[0]) : 0;`: Lee el primer argumento de línea de comandos, lo convierte a un entero y lo almacena en la variable `num`. Si el número es negativo, se establece en 0.
4. `boolean result = isSquarePerfectRecursive(num, 0);`: Llama a la función `isSquarePerfectRecursive` con `num` como argumento y almacena el resultado en la variable `result`.
5. `System.out.println(result);`: Imprime el resultado (verdadero o falso) en la consola.
6. `public static boolean isSquarePerfectRecursive(int x, int init) {`: Esto define la función `isSquarePerfectRecursive`, que toma un número `x` y un valor `init` como argumentos.
7. `int square = init * init;`: Calcula el cuadrado de `init` y lo almacena en `square`.
8. `if (square == x){`: Comprueba si `square` es igual a `x`. Si son iguales, devuelve `true`, lo que significa que `x` es un cuadrado perfecto.
9. `} else if (square >x) {`: Comprueba si `square` es mayor que `x`. Si es así, devuelve `false`, lo que significa que `x` no es un cuadrado perfecto.
10. `return isSquarePerfectRecursive(x, init + 1);`: Si `square` es menor que `x`, llama recursivamente a `isSquarePerfectRecursive` con un valor `init` incrementado en 1. Esto permite buscar el cuadrado perfecto de `x` probando diferentes valores de `init`.

2.2. Suma de Conjuntos Extrema

Este programa Java llamado `Suma_subconjuntos_extrema` se encarga de verificar si es posible elegir un subconjunto de algunos de los enteros de un arreglo, de modo que la suma del subconjunto sea igual a un objetivo dado, pero respetando las siguientes restricciones adicionales:

Listing 2: Suma_subconjuntos_extrema.java

```
import java.util.*;

public class Suma_subconjuntos_extrema {

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

```
        Scanner sc = new Scanner(System.in);
        String[] strIn = sc.nextLine().split(" ");

        int size = Integer.parseInt(strIn[0]);
        int objective = Integer.parseInt(strIn[strIn.length - 1]);
        int[] nums = new int[size];

        for (int i = 0; i < size; i++) {
            nums[i] = Integer.parseInt(strIn[i + 1]);
        }

        System.out.println(isPossible(nums, objective, 0, 0));
    }

    public static boolean isPossible(int[] nums, int target, int index, int sum) {
        if (index >= nums.length) {
            return sum == target;
        }

        if (nums[index] == 7) {
            if (index + 1 < nums.length && nums[index + 1] == 1) {
                return isPossible(nums, target, index + 2, sum) || isPossible(nums, target,
                    index + 2, sum + 7);
            } else {
                return isPossible(nums, target, index + 1, sum + 7);
            }
        } else {
            return isPossible(nums, target, index + 1, sum + nums[index]) || isPossible(nums,
                target, index + 1, sum);
        }
    }
}
```

1. `public static void main(String[] args)` {: Esto es el método `main`, que es el punto de entrada del programa.

2. El código dentro de `main` lee la entrada del usuario y llama a la función `isPossible` para determinar si es posible crear un subconjunto que cumpla con las restricciones y sume el objetivo dado.

   La función `isPossible` en el programa Java `Suma_subconjuntos_extrema` tiene la tarea de determinar si es posible elegir un subconjunto de algunos de los enteros de un arreglo, de modo que la suma de los elementos en ese subconjunto sea igual a un objetivo dado, cumpliendo con dos restricciones adicionales:

   a) Todos los múltiplos de 7 en el arreglo deben incluirse en el subconjunto.

   b) Si el valor que sigue inmediatamente a un múltiplo de 7 es 1, no debe elegirse en el subconjunto.

3. La función `isPossible` implementa esta tarea:

   - **Parámetros de entrada**:
     - `nums`: Un arreglo de enteros que representa los números disponibles para formar el subconjunto.
     - `target`: El objetivo de la suma que se debe alcanzar.

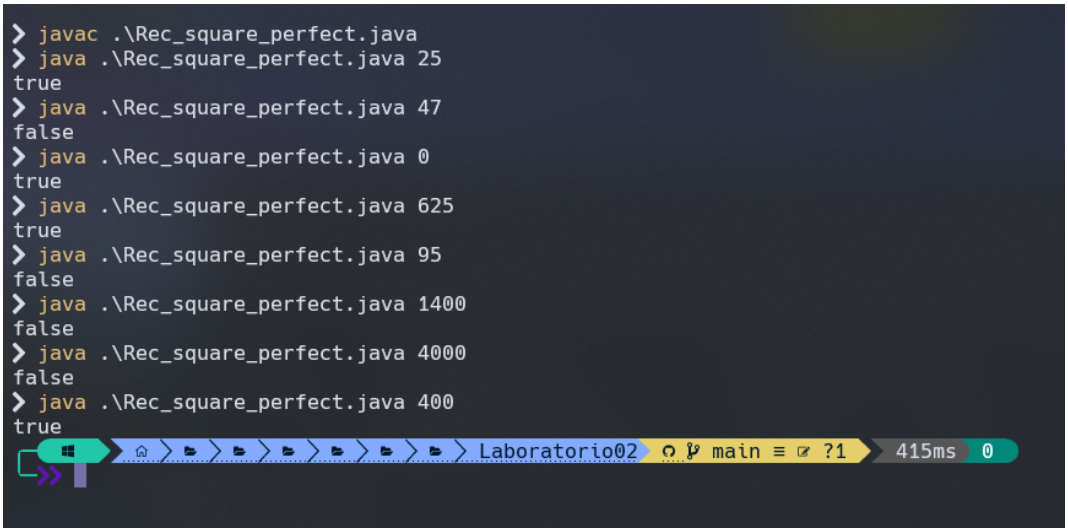
- `index`: Un índice que indica la posición actual en el arreglo `nums`.
- `sum`: La suma parcial de elementos en el subconjunto actual.

▪ **Funcionamiento**:

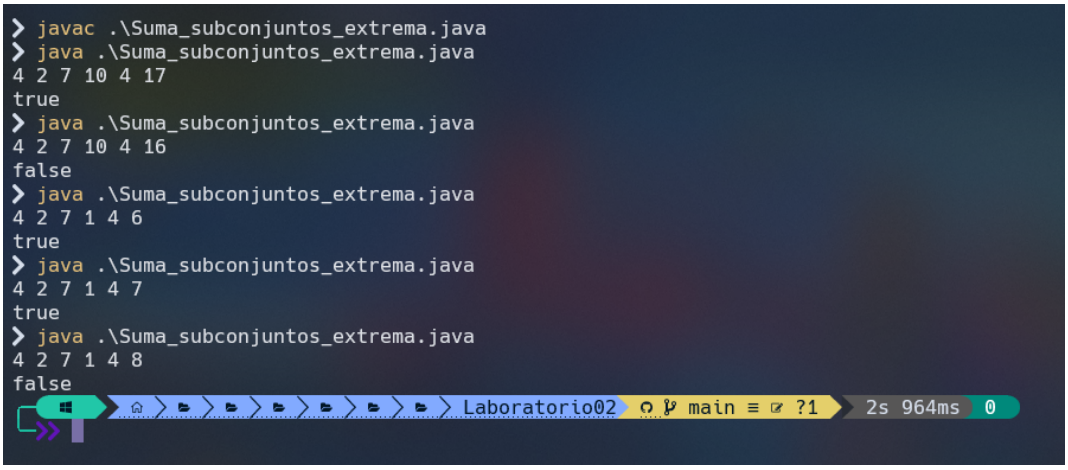
- La función comienza comprobando si `index` ha alcanzado o superado la longitud del arreglo `nums`. Si esto es cierto, significa que se han explorado todos los elementos del arreglo y la función verifica si `sum` es igual a `target`. Si es igual, devuelve `true`, lo que indica que se ha encontrado un subconjunto que cumple con las restricciones y la suma objetivo. En caso contrario, devuelve `false`.
- Si `index` no ha alcanzado la longitud del arreglo, la función continúa explorando las opciones recursivamente. El comportamiento depende del valor en `nums[index]`:
  - Si `nums[index]` es igual a 7, la función verifica si el siguiente elemento (`nums[index + 1]`) es igual a 1. Si es así, tiene dos opciones: 1. Excluir tanto el 7 como el 1 y avanzar recursivamente a la siguiente posición (`index + 2`) sin agregar nada a la suma actual. 2. Incluir el 7 y avanzar recursivamente a la siguiente posición (`index + 2`) sumando 7 a la suma actual.
  - Si `nums[index]` no es igual a 7, la función tiene dos opciones: 1. Incluir el elemento en la suma actual y avanzar recursivamente a la siguiente posición (`index + 1`). 2. Excluir el elemento y avanzar recursivamente a la siguiente posición (`index + 1`) sin agregar nada a la suma actual.
- La función se llama recursivamente en cada una de estas opciones, explorando todas las combinaciones posibles para determinar si es posible encontrar un subconjunto que cumpla con las restricciones y la suma objetivo. Si en algún punto se cumple la condición de `index >= nums.length`, se verifica si la suma parcial es igual al objetivo.

2.3. Ejecución

2.3.1. Cuadrado Perfecto

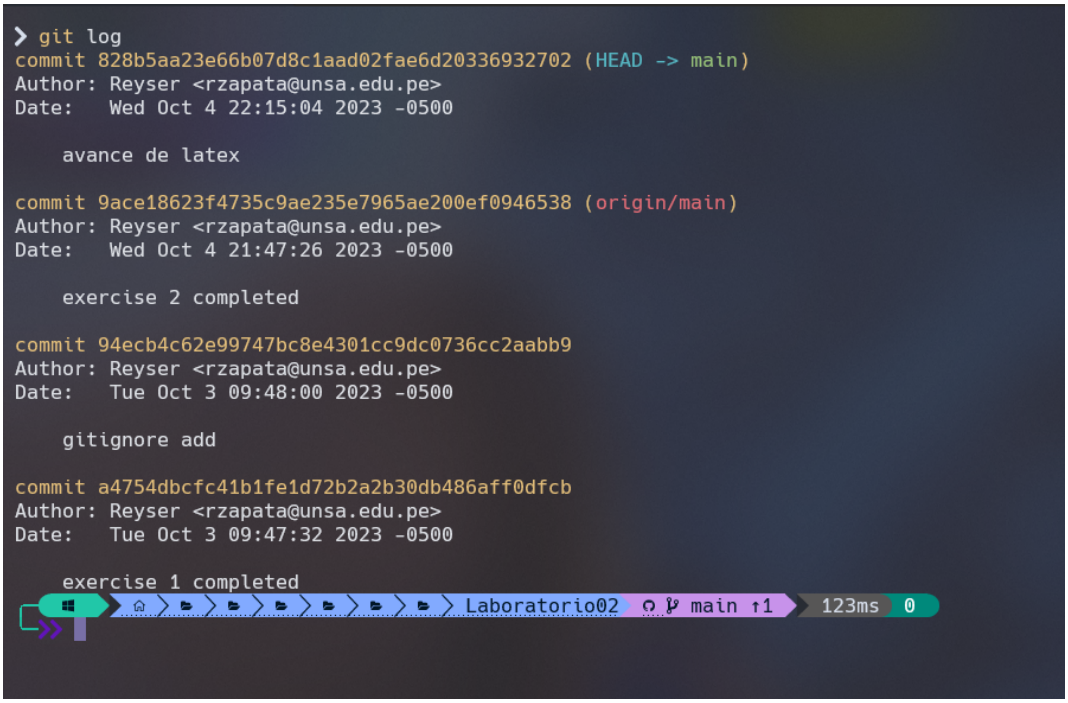


2.3.2. Suma de conjuntos extrema



2.4. Commits del trabajo

- Acá estan algunos de los commits mas importantes en este trabajo:



2.5. Instrucciones para la ejecucion

- Para poder compilar y ejecutar ambos ejercicios desde consola, se pueden seguir los siguientes comandos compatibles para todos los SO:

Listing 3: Compilación de los ejercicios

```
javac Rec_square_perfect.java Suma_subconjuntos_extrema.java
```

Listing 4: Ejecución de los ejercicios

```
java Rec_square_perfect [num]
java Suma_subconjuntos_extrema
```

3. Referencias

- https://dialnet.unirioja.es/servlet/articulo?codigo=4573315
- https://www.fdi.ucm.es/profesor/gmendez/docs/edi0910/_02-Recursion.pdf
- https://docs.oracle.com/javase/tutorial/