Programación de Computadores

Repaso de Java

Jonatan Gómez Perdomo, Ph. D.
jgomezpe@unal.edu.co

Arles Rodríguez, Ph.D.
aerodriguezp@unal.edu.co

Camilo Cubides, Ph.D. (c)
eccubidesg@unal.edu.co

Research Group on Artificial Life – Grupo de investigación en vida artificial – (Alife)
Computer and System Department
Engineering School
Universidad Nacional de Colombia

Contenido

Agenda

Enunciado

Al ver los precios y los anuncios del almacén *Cobra Mosmas*, un cliente le pide crear un programa de computador que le permita ingresar el precio individual de tres productos y el precio de la promoción en combo de los tres productos anunciada por el almacen y determine si es preferible comprarlos por separado o en el combo promoción. (Pensemos por 3 minutos en definir claramente el problema)

Entendiendo el Problema

Entradas: Los precios $p1, p2, p3$ de tres productos y del combo pc. Salida: Un texto indicando si se debe comprar el combo o los tres productos por separado. Relaciones: Si el precio del combo pc es menor o igual que la suma de los precios de los tres productos se debe imprimir "Combo" en otro caso se debe imprimir "Por separado". (Pensemos por 2 minutos en la especificación)

Especificando el Problema

$$comprar\,(p1, p2, p3, pc) = \begin{cases} \text{'Combo'} & \text{if } pc \leq p1 + p2 + p3 \\ \text{'Por separado'} & \text{en otro caso} \end{cases} \quad (1)$$

Codificación

```
import java.util.Scanner;

public class RepasoJava {
    public static String comprar(double p1, double p2,
                                 double p3, double pc){
        if(p1 <= p1 + p2 + p3){
            return "en Combo";
        }else{
            return "por separado";
        }
    }
```

Codificación

```
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        double a = Double.parseDouble(sc.nextLine());
        double b = Double.parseDouble(sc.nextLine());
        double c = Double.parseDouble(sc.nextLine());
        double d = Double.parseDouble(sc.nextLine());
        System.out.println("Comprar " + comprar(a, b, c, d));
    }
}
```

Agenda

Enunciado

Un campesino de la región le pide crear un programa de computador que le permita determinar cual de dos opciones (madera o alambre) es la mejor opción (menor costo) para encerrar un terreno rectangular de $n * m$ metros cuadrados, sabiendo el costo de un metro lineal de alambre, el costo de un metro de madera y la cantdad de hilos de alambre o hileras de madera. El campesino solo piensa en usar una de las dos opciones, no las piensa combinar. (Pensemos por 3 minutos en definir claramente el problema)

Entendiendo el Problema

Entradas: Los dos lados del rectángulo n y m. El costo de un metro lineal de alambre a, el costo de un metro lineal de madera p, el número de hilos de alambre si se hace el cercado en alambre h y el número de hileras de madera w si se hace el cercado en madera. Salida: Un texto indicando si se debe cercar en madera o si se debe cercar en alambre. Relaciones: El perímetro del rectángulo es $2n + 2m$. Una cerca en madera usará $(2n + 2m) * w$ metros lineales de madera. Una cerca de alambre usará $(2n + 2m) * h$ metros lineales de alambre. De esta manera el costo de usar alambre será $(2n + 2m) * h * a$ y el de usar madera será $(2n + 2m) * w * p$. Si el costo de usar madera es menor o igual que el de alambre se debe imprimir "Madera" en otro caso se debe imprimir "Alambre". (Pensemos por 2 minutos en la especificación)

Especificando el Problema

$$enMadera\,(n, m, w, p) = (2n + 2m) * w * p \tag{2}$$

$$enAlambre\,(n, m, h, a) = (2n + 2m) * h * a \tag{3}$$

$$usar\,(n, m, h, a, w, p) =$$

$$\begin{cases} \text{'Madera'} & \text{if } enMadera(n, m, w, p) \leq enAlambre(n, m, h, a) \\ \text{'Alambre'} & \text{en otro caso} \end{cases} \tag{4}$$

Codificación

```
import java.util.Scanner;

public class Cerca {
    public static double enMadera(double n, double m, double w, double p){
        return (2 * n + 2 * m) * w * p;
    }

    public static double enAlambre(double n, double m, double h, double a){
        return (2 * n + 2 * m) * h * a;
    }

    public static String usar(double n, double m, double h, double a,
       double w, double p) {
        if (enMadera(n, m, w, p) <= enAlambre(n, m, h, a)) {
            return "Madera";
        } else {
            return "Alambre";
        }
    }
}
```

Codificación

```
public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    System.out.println("Largo terreno?");
    double n = Double.parseDouble(sc.nextLine());
    System.out.println("Ancho del terreno");
    double m = Double.parseDouble(sc.nextLine());
    System.out.println("Costo metro alambre?");
    double a = Double.parseDouble(sc.nextLine());
    System.out.println("Hilos de alambre?");
    double h = Double.parseDouble(sc.nextLine());
    System.out.println("Costo metro madera?");
    double p = Double.parseDouble(sc.nextLine());
    System.out.println("Hileras de madera?");
    double w = Double.parseDouble(sc.nextLine());
    System.out.println("Usar" + usar(n, m, a, h, p, w));
}
}
```

Agenda

Enunciado

Unos padres de familia desesperados por determinar el dinero que deben pedir prestado para pagar los útiles escolares de su hijo, le han pedido crear un programa de computador que a partir de una lista de los precios de cada útil escolar y de la cantidad de cada útil escolar en la lista, determine el precio total de la lista. (Pensemos por 5 minutos en la solución)

Entendiendo el Problema

Entradas: Una lista con los precios de los útiles (en la posición i de la lista esta el precio del útil i), una lista con las cantidades de dichos útiles (en la posición i esta la cantidad requerida del útil i). Salida: El costo total de la lista. Relaciones: El costo total es la suma sobre todos los útiles del precio del útil por la cantidad del mismo. (Pensemos por 2 minutos en la especificación)

Especificando el Problema

$$costo\,(precio, cantidad) = \sum_{i=0}^{\text{precio}-1} precio_i * cantidad_i \qquad (5)$$

Codificación

```
import java.util.Scanner;

public class Utiles
    public static double costo(double[] precio,
    int[] cantidad){
        double costo = 0;
        for (int i = 0; i < precio.length; i++) {
            costo = costo + precio[i]*cantidad[i];
        }
        return costo;
    }
```

Codificación

```
public static void main(String[] args) {
    double[] precio = new double[100]; //se define un tamaño máximo
    int[] cantidad = new int[100];
    Scanner sc = new Scanner(System.in);
    int i = 0;
    while(true)
        System.out.println("Ingresar util? S/N:");
        if(!sc.nextLine().toUpperCase().equals("S")){
            break;
        }
        System.out.println("Precio util?");
        precio[i] = Double.parseDouble(sc.nextLine());
        System.out.println("Cantidad?");
        cantidad[i] = Integer.parseInt(sc.nextLine());
        i++;
    }
    System.out.println("La lista cuesta: " + costo(precio, cantidad));
}
```

Agenda

Enunciado

En la última edición de la revista científica "ADN al día" se indica que las pruebas de relación entre individuos a partir de código genético se define de la siguiente manera: Si las dos cadenas se diferencian en menos de p letras, existe una relación de padre-hijo, si se diferencian en menos de $f > p$ letras, existe una relación de formar parte de la misma familia. De otra manera no existe relación. El laboratorio *Tein Cul Pan*, le pide desarrollar un programa que a partir de dos cadenas de ADN del mismo tamaño, determine si existe una relación pader-hijo, de la misma familia o ninguna, siguiendo las reglas definidas por la revista científica "ADN al día". (Pensemos por 5 minutos en la solución)

Entendiendo el Problema

Entradas: Dos cadenas de carácteres (representando cadenas de ADN) a y b de la misma longitud. Los límites p, f para considerar una relación 'Padre-Hijo', 'Familiar' o 'Ninguna' Salida: Un texto indicando si las cadenas tienen una relacion 'Padre-Hijo', 'Familiar' o 'Ninguna'. Relaciones: Si las dos cadenas a, b se diferencian en menos de p letras, existe una relación de 'Padre-Hijo, si se diferencian en menos de $f > p$ letras, existe una relación de 'Familia'. En otro caso tienen 'Ninguna' relación. (Pensemos por 2 minutos en la especificación)

Especificando el Problema

$$dif\,(a,b) = \sum_{i=0}^{\mathsf{a}-1} 1 \mathsf{if} a_i \neq b_i \tag{6}$$

$$relacion\,(a,b,p,f) = \begin{cases} \text{'Padre-Hijo'} & \text{if } dif(a,b) \leq p \\ \text{'Familiar'} & \text{if} p < dif(a,b) \leq f \\ \text{'Ninguna'} & \text{en otro caso} \end{cases} \tag{7}$$

Codificación

```
import java.util.Scanner;

public class ADN {
    public static int diferencia(String a, String b) {
        int cuenta = 0;
        for (int i = 0; i < a.length(); i++) {
            if (a.charAt(i) != b.charAt(i)) {
                cuenta++;
            }
        }
        return cuenta;
    }
```

Codificación

```
public static String relacion(String a, String b, int p, int f) {
    int d = diferencia(a, b);
    if (d <= p) {
        return "Padre-Hijo";
    } else if (d <= f) {
        return "Familia";
    } else {
        return "Ninguna";
    }
}
```

Codificación

```
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Cadena ADN individuo 1?");
        String ind1 = sc.nextLine();
        System.out.println("Cadena ADN individuo 2?");
        String ind2 = sc.nextLine();
        System.out.println("Diferencia máxima para ser Padre-Hijo?");
        int p = Integer.parseInt(sc.nextLine());
        System.out.println("Diferencia máxima para ser Familia?");
        int f = Integer.parseInt(sc.nextLine());
        System.out.println("Relación " + relacion(ind1, ind2, p, f));
    }
}
```

Agenda

Enunciado

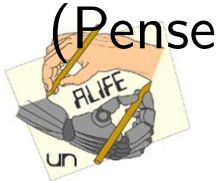
Se tienen que procesar algunos comandos para realizar el procesamiento de notas de una Universidad. Se tiene una lista de estudiantes

- Comando 1: Agregar estudiante y nota

  `1&nombre_estudiante&nota`
- Comando 2: Calcular promedio de los estudiantes en un momento dado
- Comando 3: Ordenar estudiantes agregados por nombre
- Comando 4: Consultar la nota de un estudiante

  `4&nombre_estudiante`
- Comando 5: Visualizar lista de estudiantes
- Comando 6: Salir

(Pensemos por 5 minutos en la solución)



Enunciado

Para poder resolver el problema se puede dividir dicho problema en problemas más pequeños que son:

- Definir la lista de estudiantes.
- agregar un estudiante dada la información
- calcular el promedio de notas de los estudiantes en un momento dado
- ordenar estudiantes agregados por nombre
- consultar la nota de un estudiante
- visualizar lista
- procesar los comandos
- mostrar menu

(Pensemos por 5 minutos en la solución)

Definición de Clase Estudiante

Para poder resolver el problema se pueden crear las clases Estudiante y ListaEstudiantes

- Un estudiante tiene una cadena de caracteres llamada nombre
- Un estudiante tiene una nota de tipo double
- ListaEstudiantes tiene un arreglo de Estudiantes
- Se define también la cantidad de estudiantes de la lista n

Definición de Clase Estudiante

Se define un archivo llamado Estudiante.java con la siguiente estructura:

```
public class Estudiante {
    private String nombre;
    private double nota;

    public Estudiante(String nombre, double nota) {
        this.nombre = nombre;
        this.nota = nota;
    }

    public String getNombre() {
        return nombre;
    }
```

Definición de Clase Estudiante

```
    public double getNota() {
        return nota;
    }

    @Override
    public String toString() {
        return nombre + "\t" + nota;
    }
}
```

Definición de Clase ListaEstudiantes

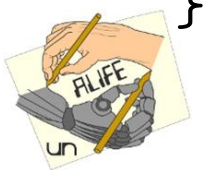
Se define un archivo llamado ListaEstudiantes.java con la siguiente estructura:

```
import java.util.Arrays;
import java.util.Scanner;

public class ListaEstudiantes {

    private Estudiante[] estudiantes;
    private int n;

    public ListaEstudiantes() {
        //soporta hasta 10000 estudiantes
        estudiantes = new Estudiante[10000];
    }
```



Agregar un estudiante

```
public void agregarEstudiante(Estudiante e) {
    estudiantes[n] = e;
    n++;
}
```

Obtener promedio de notas

```
public void promedio(){
    double prom = 0;
    for(int i=0; i < n; i++){
        prom += estudiantes[i].getNota();
    }
    System.out.println("El promedio de los estudiantes es:");
    System.out.println(prom/n);
}
```

Ordenar estudiantes

Aquí aprovechamos la librería Arrays y su método sort. Adicionalmente definimos que los estudiantes se compararán por nombre.

```
public void ordenar(){
    Arrays.sort(estudiantes, 0, n,
    (e1,e2)->e1.getNombre().compareTo(e2.getNombre()));
}
```

Consultar nota de un estudiante

```
public void consultar(String nombre){
    boolean encontrado = false;
    for (int i = 0; i < n; i++) {
        if(estudiantes[i].getNombre().equals(nombre)){
            System.out.println(estudiantes[i].getNota());
            encontrado = true;
        }
    }
    if(!encontrado){
        System.out.println("Estudiante no encontrado.");
    }
}
```

Visualizar lista

```
public void visualizar(){
    System.out.println("****Lista de Estudiantes****");
    for (int i=0; i < n; i++) {
        System.out.println(estudiantes[i]);
    }
}
```

Mostrar menú

```
public void mostrarMenu(){
    System.out.println("Seleccione una opción:");
    System.out.println("Comando 1: Agregar estudiante y nota:
        1&nombre_estudiante&nota");
    System.out.println("Comando 2: Calcular promedio de los estudiantes
        en un momento dado.");
    System.out.println("Comando 3: Ordenar estudiantes agregados por nombre");
    System.out.println("Comando 4: Consultar la nota de un estudiante
    4&nombre_estudiante");
    System.out.println("Comando 5: Visualizar");
    System.out.println("Comando 6: Salir");
}
```

Procesar Comandos

```
public void procesarComandos(){
    String[] comando;
    Scanner sc = new Scanner(System.in);
    do{
        mostrarMenu();
        comando = sc.nextLine().split("&");
        switch(comando[0]){
            case "1":
                agregarEstudiante(new Estudiante(comando[1],
                        Double.parseDouble(comando[2])));
                break;
            case "2":
                promedio();
                break;
            case "3":
                ordenar();
                break;
```

Procesar Comandos II

```
                case "4":
                    consultar(comando[1]);
                    break;
                case "5":
                    visualizar();
                break;
            }
        }while(!comando[0].equals("6"));
    }
```

El programa principal

```
public static void main(String[] args)
    ListaEstudiantes e = new ListaEstudiantes();
    e.procesarComandos();
```

Problemas varios I

1. Hacer un algoritmo que deje al final de un arreglo de números enteros todos los ceros que aparezcan en dicho arreglo.

Example

arreglo original: `[1, 6, 0, 7, -3, 8, 0, -2, 11]`

arreglo salida: `[1, 6, 7, -3, 8, -2, 11, 0, 0]`

Example

arreglo original:

`[0, 11, 36, 10, 0, 17, -23, 81, 0, 0, 12, 11, 0]`

arreglo salida:

`[11, 36, 10, 17, -23, 81, 12, 11, 0, 0, 0, 0, 0]`

Problemas varios I

1. Suponga que un arreglo de enteros esta lleno de unos y ceros y que el arreglo representa un número binario al revés. Hacer un algoritmo que calcule los números en base decimal que representa dicho arreglo de unos y ceros.

Example

arreglo: `[1, 0, 0, 1, 0, 1, 1, 1, 1]`, que representa el número 111101001.

número: 389.

Example

arreglo: `[0, 1, 0, 1, 0, 1, 1]`, que representa el número 1101010.

número: 106.

Problemas varios I

1. Hacer un algoritmo que dado un número entero no negativo, cree un arreglo de unos y ceros que representa el número en binario al revés.

Example

número: 389.

arreglo: `[1, 0, 0, 1, 0, 1, 1, 1, 1]`, que representa el número 111101001.

Example

número: 106.

arreglo: `[0, 1, 0, 1, 0, 1, 1]`, que representa el número 1101010.

2. Hacer un algoritmo que calcule el máximo común divisor para un arreglo de enteros positivos.

Example

Problemas varios I

1. Hacer un algoritmo que deje al final de un arreglo de números enteros todos los ceros que aparezcan en dicho arreglo.

Example

arreglo original: `[1, 6, 0, 7, -3, 8, 0, -2, 11]`

arreglo salida: `[1, 6, 7, -3, 8, -2, 11, 0, 0]`

Example

arreglo original:

`[0, 11, 36, 10, 0, 17, -23, 81, 0, 0, 12, 11, 0]`

arreglo salida:

`[11, 36, 10, 17, -23, 81, 12, 11, 0, 0, 0, 0, 0]`

2. Suponga que un arreglo de enteros esta lleno de unos y ceros y que el arreglo representa un número binario al revés. Hacer un algoritmo que calcule los números en base decimal que representa dicho arreglo de

Problemas varios I

1. Hacer un algoritmo que calcule el máximo común divisor para un arreglo de enteros positivos.

Example

arreglo: `[12, 20, 14, 124, 72, 2458]`.

m.c.d del arreglo: 2.

2. Hacer un algoritmo que calcule el mínimo común múltiplo para un arreglo de enteros positivos.

Example

arreglo: `[12, 20, 30, 15]`.

M.C.M del arreglo: 60.