­­­

**Práctica\_2\_5 v1.** Análisis de algoritmos “voraces”: el problema del cambio

**GUIÓN DE LA PRÁCTICA**

**1.- Objetivos de la práctica**

En esta práctica vamos a intentar resolver el problema del cambio diseñando un algoritmo “voraz” o “greedy”:

Considérese un sistema monetario con n monedas distintas, de valores v1, v2, …., vn y supóngase que hay un número ilimitado de monedas de cada valor. El problema del cambio consiste en descomponer una cierta

cantidad a devolver M en el menor número posible de monedas.

Se trata de implementar un algoritmo voraz para resolver este problema, aplicando el método que utilizamos habitualmente en nuestra vida diaria (primero monedas grandes, usando tantas como se pueda sin pasarse de la cantidad a devolver).

**2.- Actividades a realizar**

**Actividad 1:** Resolver el problema del cambio utilizando un algoritmo voraz: disponemos de monedas de distintos valores (sistema monetario euro): de **1, 20 y 50 céntimos de euro, y de 1 y 2 euros**. Supondremos una cantidad ilimitada de cada una (por tanto, **SIN RESTRICCIONES**). Nuestra función objetivo será devolver el cambio usando el **número mínimo de monedas** (**MINIMIZACIÓN**). Por ejemplo, para devolver 3.89 €: 1 moneda de 2€, 1 moneda de 1€,1 moneda de 50 c€, 1 moneda de 20 c€, 1 moneda de 10 c€, 1 moneda de 5 c€ y 2 monedas de 2 c€.

**funcion** Devolver-cambio (int P): *conjunto de monedas(X)*

**const** C={1,2,5,10,20,50,100 (un euro),200(2 Euros)} // C=monedas disponibles; conj. candidatos

int X[1..N] ; // **X**= conjunto que contendrá la **solución**

actual = 0 //suma acumulada de la cantidad procesada

**para** i = 1 hasta N

X[i] = 0 // **inicialización(X)**

**fpara**

**mientras** actual ≠ P // **no solución(X)**

j = el mayor elemento de C tal que C[j] ≤ (P**-**actual) // **seleccionar(C)**

**si** j=0 entonces // *Si no existe ese elemento =>* **no factible(j)**

**devolver** “No existe solución”;

**fsi**

X[j] = (P**-**actual) div C[j] // **insertar**(C,X)

actual = actual + C[j]\*X[j]

**fmientras**

**devolver** X // objetivo(X)

**ffuncion**

1. Probar el algoritmo para las siguientes secuencias de entradas sin incluir restricciones del número de monedas de cada cantidad: 3.45, 12.48, 8.9, 5.24, 48, 25.5, que expresadas en céntimos serán:

345, 1248, 890,524,48,255

1. Incluir una copia de pantalla de las soluciones para las secuencias de entrada en la Hoja de trabajo
2. ¿Llega esta implementación siempre a la solución óptima? ¿Por qué?

**Actividad 2**: Modificar el código de la Actividad 1 para introducir las siguientes **restricciones** en cuanto al número de monedas:

|  |  |
| --- | --- |
| MONEDA | Nº máximo disponible (en el conjunto de candidatos) |
| 1 | 1 |
| 2 | 1 |
| 5 | 1 |
| 10 | 5 |
| 20 | 6 |
| 50 | 1 |
| 1€ | 2 |
| 2€ | 3 |

1. Probar el algoritmo para las siguientes secuencias de entradas 3.45, 12.48, 8.9, 5.24, 0.48, 25.5, que expresadas en centavos será: 345, 1248, 890,524,48,2550
2. Incluir una copia de pantalla de las soluciones para las secuencias de entrada en la Hoja de trabajo
3. ¿Llega esta implementación siempre a la solución óptima? ¿Por qué?

Ejemplo de ejecución:

Monedas utilizadas para el cambio de 255:

1 moneda(s) de 5

1 moneda(s) de 50

1 moneda(s) de 200

**Actividad 3**:  Modificar el código de la Actividad 1 para cambiar el conjunto de candidatos inicial (en nuestro caso, el sistema monetario), suponiendo ahora que nuestro sistema solo dispone de monedas de **1, 3, 6, 12, 24, 30, 100 y 200**y **SIN RESTRICCIONES** en cuanto al número de monedas disponibles de cada valor.

1. Probar el algoritmo para las siguientes secuencias de entradas 3.45, 12.48, 8.9, 5.24, 48, 25.5, que expresadas en centavos será: 345, 1248, 890,524,48,255
2. Incluir una copia de pantalla de las soluciones para las secuencias de entrada en la Hoja de trabajo
3. ¿Llega esta implementación siempre a la solución óptima? ¿Por qué?

Ejemplo de resultado:

**Monedas utilizadas para el cambio de 345:**

**1 moneda(s) de 3**

**1 moneda(s) de 12**

**11 moneda(s) de 30**

**1 moneda(s) de 100**

**1 moneda(s) de 200**

**3.- Entrega fin sesión**

Los archivos con los programas correspondientes a las actividades se deberán entregar, a través de Moodle, al finalizar la sesión de prácticas, junto con la Hoja de Trabajo del estudiante. Todo el código entregado debe compilar y ejecutar, comente aquel código que está pendiente para la entrega final de la práctica para que no de errores de compilación.

Las líneas de código deben estar comentadas con su funcionalidad, tome como ejemplo los comentarios en el código proporcionado y las fórmulas utilizadas para el cálculo de las operaciones elementales en cada línea de código, así como las fórmulas teóricas utilizadas para el cálculo teórico.

**4.- Después de asistir al laboratorio**

* + Completar el código y el documento de trabajo que no haya dado tiempo a realizar en clase.
  + El código y documento entregado en esta entrega debe realizarse, aunque se haya terminado todo el trabajo en clase. El código debe ser una ampliación de lo realizado en clase no puede ser completamente nuevo.
  + Realizar la Entrega\_Practica\_2\_5 en un fichero comprimido con la hoja de trabajo:
    - Actividades 1-3:
      * Código con la implementación de las tres funciones de calculo de monedas
      * Hoja de Trabajo con lo solicitado en los apartados 3 de las tres actividades.