Facultad Politécnica – Ingeniería Informática
Algoritmos y Estructura de Datos III – 2022 – 1er. Periodo – Sección TQ/TR
Tarea 8 - U6

Trabajo a entregar por GRUPO vía EDUCA en cada Actividad VPL. Indique claramente los integrantes de su grupo. El código debe poder probarse sin usar ningún IDE desde la linea de comandos. Cada integrante debe entregar la tarea.

Ejercicio 1 (10p)

Considere el método de exponenciación basada en la técnica de Divide y Vencerás mostrada en la fórmula de abajo:

$$a^n = \begin{cases} (a^{n/2})^2 & si \quad n \quad es\ par\ y\ positivo \\ (a^{(n-1)/2})^2 . a & si \quad n \quad es\ impar\ y\ mayor\ a\ 1 \\ a & si \quad n = 1 \end{cases}$$

a) Implemente en Java y considerar ***a*** y ***n*** como valores enteros mayores a cero (5p)
b) Escriba su ecuación de recurrencia y resuélvala (2p)
c) Compare el tiempo de ejecución con el algoritmo de fuerza bruta generando la tabla de comparación (puede dejar fijo ***a*** e ir aumentando ***n*** gradualmente) (3p)

Ejercicio 2 (10p)

Considere el problema de la multiplicación de dos enteros X e Y de *n* dígitos decimales. Construya un algoritmo utilizando la técnica de divide y vencerás teniendo en cuenta lo siguiente:

Dividir

X en A y B dígitos donde A son los dígitos altos y B los bajos entonces $X = A.10^{n/2} + B$
Y en C y D dígitos donde C son los dígitos altos y D los bajos entonces $Y = C.10^{n/2} + D$

Se sabe que:

$$XY = AC10^n + [(A-B)\ (D-C) + AC + BD]10^{n/2} + BD$$

a) Implemente el algoritmo en Java (Utilice la clase *BigInteger* para ello). Para el caso base utilice el algoritmo clásico cuadrático iterativo de la escuela (que debe implementar) (7p)

b) Compare esta técnica con la versión de multiplicación clásica iterativa. Construya una tabla de comparación (en tiempo) entre ambas implementaciones. Indique a partir de que números se nota la diferencia de tiempo. Determine empíricamente el mejor número n_0 para la implementación DyV (Pruebe entre 1000 y 30000 dígitos) (3p)

Ejercicio 3 (10p)

Utilizando un algoritmo voraz resuelva el problema de minimización de la cantidad de billetes de "vuelto".

a) Implemente un algoritmo en Java para calcular la cantidad de billetes/monedas de cada denominación dado un monto, que se recibe como parámetro, presumiendo que se tiene una cantidad ilimitada de unidades de cada denominación.

   Considere que las denominaciones (enteras) también son un parámetro del problema recibidos en un arreglo.

   ```
   int [] minimizarVuelto ( long monto, int [] denominaciones)
   ```

   En caso de que reciba un monto que no es posible obtener con alguna combinación entonces generar una excepción (5p)

a) ¿Obtiene su solución siempre la cantidad óptima de billetes, es decir mínima? Explique porque si o porque no. (2p)
b) SI NO es óptima proponga un algoritmo eficiente de tiempo polinomial que lo encuentre (3p)

Ejercicio 4 (10p)

Utilizando el código de multiplicación óptima de matrices presentada en el capítulo 15 de libro *Introduction to Algorithm de Cormen et al. 3ed Edition*. 2009.

Tradúzcalo a Java y complete la implementación de tal forma que pueda obtenerse una matriz resultante de multiplicar ***n*** matrices aplicando el algoritmo de optimización mencionado. Para ello debe diseñar un `TDA Matriz`. Además provea otro método independiente que retorne una cadena donde muestre la parentización resultante de la optimización. Una posible forma de aplicación ejemplo sería:

```
Matriz A, B, C, D, E, F;
Matriz R;
Matriz ListaMatrices [ ] = new Matriz [6];
String parentizacion;
ListaMatrices[0] = A;
ListaMatrices[1] = B;
ListaMatrices[2] = C;
..
R = Matriz.multiplicar ( ListaMatrices ) ;  // Quiero multiplicar A.B.C.D.E.F y dejar el
                                                    // resultado en R, en ese orden.
                                                    // Fijarse que debe controlar la
compatibilidad                                      // ListaMatrices puede implementar como
un simple                                              vector o como una clase propia
// Retornar la parentización, algo así como ((A1*(((A2*A3)*A4)*A5))*A6) (Para las 6
matrices)
parentización = Matriz.obtenerParentizacionOptima ( ListaMatrices );
System.out.printf("\n%s\", parentizacion);
```

Ejercicio 5 (10p)

Una empresa dispone de ***k*** máquinas. Se cuenta con ***H*** unidades de combustible. Se asignara un número entero de unidades de combustible a cada máquina. Para cada maquina ***i***, se cuenta con un arreglo $G_{i=}[1..H]$ tal que, si se asignan ***h*** unidades de combustible a la máquina ***i***, la misma producirá un beneficio $B_i = \sum_{i<=h} G_i[h]$. Si se asigna 0 combustible, la máquina produce 0 beneficio.

Se necesita asignar ***H*** unidades de combustible a las máquinas $1..n$, es decir, determinar números enteros positivos $h_i,..,h_k$ con $\sum h_i = H$ tal que $\sum_i B_i$ sea máximo.

El siguiente algoritmo retorna solamente el mejor beneficio total obtenible, no la asignación que lo determina.

```
funcion MEJORBENEFICIO ( H, G1[0,..,H],..,Gk [0,..,H])
inicio
   si n == 1 entonces
      retornar Σ_{i=1}^{H} G_i(i)
   fin-si
   MejorBeneficio <- 0
   Beneficio1 <- 0
   desde h= 0 hasta H hacer
       BeneficioActual <- Beneficio1 + MEJORBENEFICIO (H-h, G2,...,Gk)
       si BeneficioActual > MejorBeneficio entonces
          MejorBeneficio <- BeneficioActual
       fin-si
       si h < H entonces
          Beneficio1 <- Beneficio1 + G1[H+1]
       fin-si
   fin-desde
   retornar MejorBeneficio
fin-funcion
```

Se pide:

a) Dibujar el árbol recursivo de llamadas y respuestas para la siguiente entrada
   `k = 3; H = 3;` G_1 `= [1..3] = [70,10,20];` G_2 `= [1..3] = [30,70,0]`
   G_3 `= [1..3] = [35,40,25]`
b) Indicar la cota superior de la cantidad de llamadas recursivas que realiza el algoritmo en el peor caso, justificando la respuesta.
c) Utilizando la técnica de programación dinámica, sustituir la recursión.
d) Indicar el costo temporal de la solución en c)
e) Mostrar la estructura de datos que produce c) para el ejemplo en a).