UNIVERSIDAD DEL QUINDÍO – INGENIERIA DE SISTEMAS

ALGORITMOS III Septiembre 7/2000

Prof. **Leonardo Hernández**

Ejercicios sobre algoritmos voraces

11. Ejercicios sobre el problema de "dar la vuelta"

11.1) En el problema de "dar la vuelta" con suministro limitado de monedas, de un ejemplo para cada una de las siguientes situaciones:
a) Existe una solución óptima, pero el algoritmo voraz encuentra una solución no óptima.
b) Existe una solución óptima y el algoritmo voraz no encuentra ninguna solución.
c) No existe solución aunque el valor en total de monedas dispon ibles sea mayor al total a devolver.
d) Existen tres soluciones óptimas diferentes.

11.2) En el problema de "dar la vuelta" con suministro inagotable de monedas, de un ejemplo para demostrar que el algoritmo voraz no siempre encuentra una solución óptima en cada uno se los siguientes casos:

a) Con el sistema monetario inglés antiguo. Este sistema contiene monedas de media corona(30 peniques), un florín(24 peñiques), un chelín (12 peniques) y monedas de 6, 3, y 1 penique.
b) Con el sistema monetario portugués. Este sistema contiene monedas de 1, 2.50, 5, 10, 20, 25 y 50 escudos. Los precios siempre son un número entero de escudos.
c) Toda moneda vale al menos el doble de la moneda inmediatamente inferior, hay monedas de $1.

11.3) En el problema de "dar la vuelta" con suministro inagotable de monedas, de un ejemplo donde la solución óptima conste de 5 monedas y la solución del algoritmo voraz conste de 6 monedas.

11.4) En el problema de "dar la vuelta" con suministro inagotable de monedas, de un ejemplo donde la solución óptima conste de 3 monedas diferentes y la solución del algoritmo voraz conste de 4 monedas diferentes.

11.5) En el problema de "dar la vuelta" con suministro inagotable de monedas, de un ejemplo, en el cual el algoritmo voraz, encuentra una solución óptima y, además, hay otras dos soluciones óptimas diferentes, por lo menos.

11.6) El siguiente es el algoritmo voraz de "dar la vuelta" con suministro inagotable de monedas, un poco más detallado que el explicado en clase. Infortunadamente tiene algunos errores y Ud. debe corregirlos. Como ejemplo ya se efectuó una corrección. Pista: es posible arreglar el algoritmo efectuando únicamente otras tres correcciones.

c representa las denominaciones de moneda disponibles, y se asume que está previamente ordenado en orden descendente. Ejemplo: c = [1000, 500, 100, 50, 20] indica que se dispone de un suministro inagotable de $1000, $500 , $100, $50 y $20.

i es el subíndice para recorrer el arreglo **c**

n es el número de denominaciones disponibles. En el ejemplo anterior n = 5.

v es la cantidad a devolver. Ejemplo: v = 2140 indica que se debe devolver $2140.

s representa la solución, es un arreglo unidimensional donde **s[i]** representa el número de monedas de denominación **c[i]** que se requieren en la devuelta. Ejemplo: s = [2, 0, 1, 0, 2] significa, siguiendo el ejemplo anterior, que para devolver $2140 se requieren 2 monedas de $1000, 0 monedas de $500, 1 moneda de $100, 0 monedas de $50 y 2 monedas de $20.

```
Función devolver( vector c[1..n], v): vector[1..n]
      vector s[1..v]  <---------- s[1..n]
      para i <- 1 hasta n hacer
            s[i] <- 0
      sum <- 0
      i <- 1
      mientras sum < v hacer
            mientras c[i] + sum > v hacer
                  i <- i + 1
            si i = n entonces
                  devolver << no encuentro la solución >>
            s[i] <- s[i] + c[i]
            sum <- sum + c[i]
      devolver s
```

12. Ejercicios sobre algoritmos voraces en general

12.1) Sean ***n*** programas $P_1, P_2, \ldots, P_n$ que hay que almacenar en un disco. El programa P_i requiere s_i kilobytes de espacio y la capacidad del disco es ***D*** Kilobytes, donde:

$$D < s_1 + s_2 + \ldots + s_n$$

Se desea utilizar la mayor cantidad de espacio posible en el disco. Se considera utilizar un algoritmo voraz que seleccione los programas por ord en no creciente de s_i. De un contraejemplo que demuestre que el algoritmo voraz no siempre funciona.

12.2) En el problema de planificación con plazo fijo, considere el siguiente conjunto de tareas con sus respectivos plazos:

Tarea	i	1	2	3	4
Plazo	d_i	3	1	1	2

La lista de todos los conjuntos factibles es(complete):

{1}, {2}, {3}, {4}, {1,2}, {1,3}, ...

12.3) En el problema de planificación con plazo fijo, considere el siguiente conjunto de tareas con sus respectivos plazos:

Tarea	i	1	2	3	4
Plazo	d_i	1	3	2	1

La lista de todos los conjuntos factibles es(complete):

{1}, {2}, {3}, {4}, {1,2}, {1,3}, ...

13) Ejercicios sobre el problema de la mochila.

En cada uno de los casos siguientes, complete el siguiente cuadro con las tres formas indicadas de llenar la mochila:

Función de selección	fracción de cada objeto						Valor total de la mochila
	1	2	3	4	5	6	
Máximo valor							$
Mínimo peso							$
Máximo valor por unidad de peso							$

13.1) Capacidad de la mochila: 110

Objeto:	1	2	3	4	5	6
Peso:	40	80	150	50	25	20
Valor:	80	16	150	300	400	280

13.2) Capacidad de la mochila: 110

Objeto:	1	2	3	4	5	6
Peso:	20	40	80	150	50	25
Valor:	140	40	8	75	150	200

13.3) Capacidad de la mochila: 130

Objeto:	1	2	3	4
Peso w:	20	30	50	60
Valor v:	20	45	25	120

13.4) Capacidad de la mochila: 60

Objeto:	1	2	3	4
Peso w:	10	20	40	50
Valor v:	30	20	80	25

13.5) Capacidad de la mochila: 140 kilos

Objeto:	1	2	3	4
Peso w:	20k	40k	60k	80k
Valor v:	$40	$20	$90	$80

13.6) Capacidad de la mochila: 120 kilos

Objeto:	1	2	3	4
Peso w:	20k	40k	60k	80k
Valor v:	$50	$80	$60	$96

Respuestas

(11.6) (a) renglón 7: mientras i<=n y c[i]+sum>v hacer (b) renglón 10: si i>n entonces (c) renglón 12: s[i]=s[i]+1

(13.4)

1	0	1	.2	115
1	1	.75	0	110
1	.5	1	0	120

(13.5)

0	0	1	1	170
1	1	1	¼	170
1	0	1	¾	190

(13.6)

0	1	0	1	176
1	1	1	0	190
1	1	0	¾	202