Programación III

Solución Examen 2ª Semana de Febrero

Cuestión 1 (2 Puntos)

Explica cómo pueden ordenarse n valores enteros positivos en tiempo lineal, sabiendo que el rango de dichos valores es limitado. Explica las ventajas e inconvenientes de este método

Respuesta.

Suponiendo que el rango de los valores a ordenar es limitado, es decir, sabemos que por ejemplo el rango de números a ordenar va de 0 a 2000, podemos generar un vector de 2000 elementos de tipo entero que almacenará almacenará el número de ocurrencias de cada valor de la lista a ordenar en la posición del vector correspondiente.

1- El vector esta inicializado a 0 (O(n)).
2- Recorremos la lista de valores a ordenar (O(n)).
    a. Por cada valor i extraído de la lista, incrementamos el valor de la posición i del vector.
3- Recorremos el vector generado mostrando los valores generados en el paso anterior (2) y omitiendo aquellas posiciones del vector que contengan un 0.

(Página 79 Brassard)

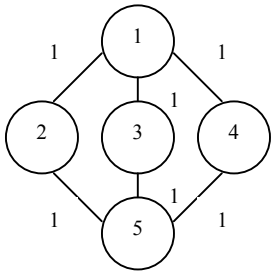
Cuestión 2 (1 Punto)

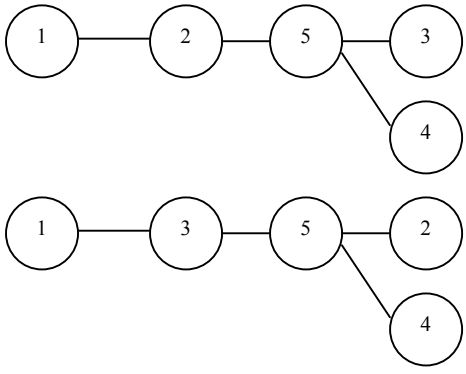
¿Puede un grafo tener dos árboles de recubrimiento mínimo diferentes? En caso afirmativo poner un ejemplo. En caso negativo justificar la respuesta.

Respuesta

Si.

Siendo G=(N,A) un grafo conexo no dirigido en donde N es el conjunto de nodos y A es el conjunto de aristas. Suponiendo que cada arista posee una longitud no negativa. Encontrar un árbol de recubrimiento mínimo consiste en hallar un subconjunto T de las aristas de G tal que utilizando solamente las aristas de T, todos los nodos deben quedar conectados, y además la suma de las longitudes de las aristas de T debe ser tan pequeña como sea posible.





(Página 215 Brassard)

Cuestión 3 (2 Puntos)

Aplicar el algoritmo de planificación con plazo fijo para las actividades ai maximizando el beneficio gi en el plazo di. Detallar todos los pasos con claridad.

$\mathbf{a_i}$	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
$\mathbf{g_i}$	20	10	7	15	25	15	5	30
$\mathbf{d_i}$	4	5	1	1	2	3	1	2

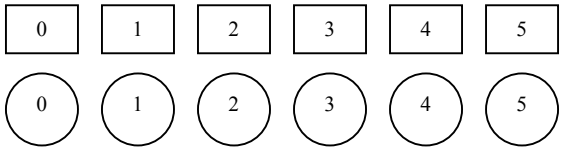
Respuesta

Inicialmente procederemos a ordenar la matriz de costes y plazos de mayor a menor en función de los costes. Una vez realizado este paso, procederemos a aplicar el algoritmo rápido (páginas 237-241 Brassard) de planificación con plazo fijo.

Resultado de la ordenación:

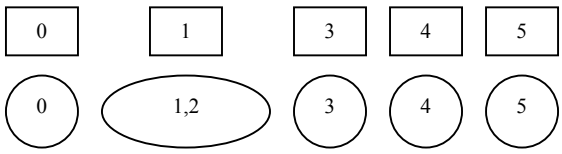
i	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
$\mathbf{a_i}$	8	5	1	4	6	2	3	7
$\mathbf{g_i}$	30	25	20	15	15	10	7	5
$\mathbf{d_i}$	2	2	4	1	3	5	1	1

Calculamos p=min(8,5)=5



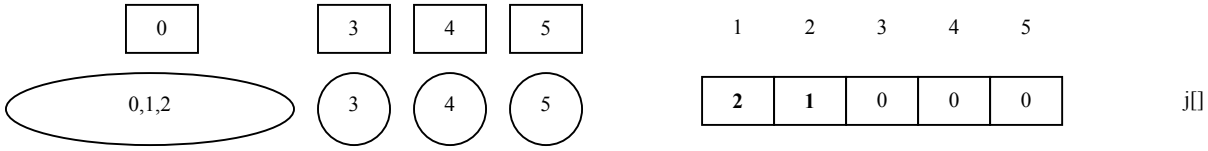
1	2	3	4	5	
0	0	0	0	0	j[]

i=1, d[1]=2. Seleccionamos K(2)

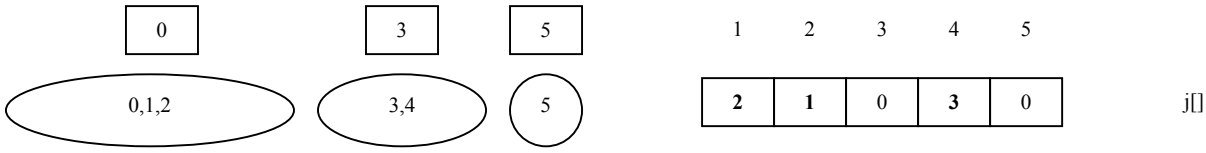


1	2	3	4	5	
0	**1**	0	0	0	j[]

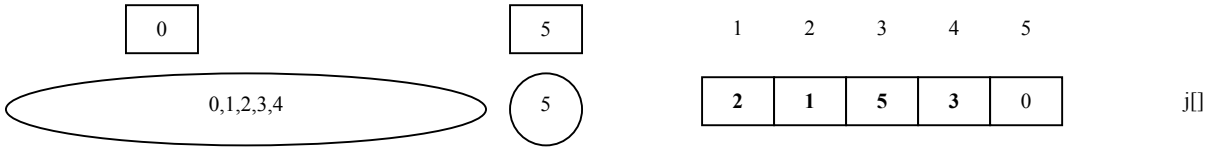
i=2, d[2]=2. Seleccionamos K(1)



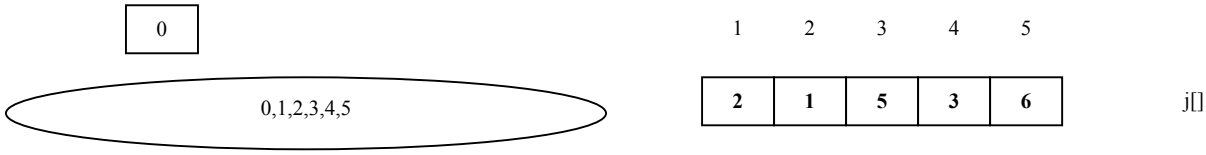
i=3, d[3]=4. Seleccionamos K(4)



i=4, d[4]=1. Tarea rechazada
i=5, d[5]=3. Seleccionamos K(3)



i=6, d[6]=5. Seleccionamos K(5)



Resto de tareas rechazadas. Fin del Algoritmo.

Orden de ejecución de las tareas: a_5, a_8, a_6, a_1, a_2.

(Página 237-241 Brassard)

Problema (5 Puntos)

Se pide diseñar completamente un algoritmo que calcule en tiempo logerítmico el valor de fn de la sucesión definida como $f_n=af_{n-1}+bf_{n-2}$ con $f_0=0$ y $f_1=1$. Sugerencia: Pudiera ser de utilidad razonar el problema utilizando la siguiente igualdad entre matrices:

$$\begin{pmatrix} a & b \\ 1 & 0 \end{pmatrix}\begin{pmatrix} f_{n-1} \\ f_{n-2} \end{pmatrix}=\begin{pmatrix} f_n \\ f_{n-1} \end{pmatrix}$$

Solución

La solución trivial se basa en resolver recursivamente el caso n resolviendo los n términos anteriores, lo cual supondría un coste lineal que no es lo que nos piden en el enunciado del problema.

Planteamiento

Si llamamos F a la matriz $\begin{pmatrix} a & b \\ 1 & 0 \end{pmatrix}$, vamos a intentar encontrar una fórmula que nos permita expresar la solución del problema en función de los casos base que se proporcionan en el enunciado que son $f_0=0$ y $f_1=1$.

$$\left.\begin{array}{l} \begin{pmatrix} f_n \\ f_{n-1} \end{pmatrix} = F \cdot \begin{pmatrix} f_{n-1} \\ f_{n-2} \end{pmatrix} \\ \begin{pmatrix} f_{n-1} \\ f_{n-2} \end{pmatrix} = F \cdot \begin{pmatrix} f_{n-2} \\ f_{n-3} \end{pmatrix} \\ \ldots\ldots\ldots\ldots\ldots\ldots \\ \begin{pmatrix} f_2 \\ f_1 \end{pmatrix} = F \cdot \begin{pmatrix} f_1 \\ f_0 \end{pmatrix} \end{array}\right\} \Rightarrow \begin{pmatrix} f_n \\ f_{n-1} \end{pmatrix} = F^2 \cdot \begin{pmatrix} f_{n-2} \\ f_{n-3} \end{pmatrix} = \ldots\ldots = F^{n-1} \cdot \begin{pmatrix} f_1 \\ f_0 \end{pmatrix}$$

Con esta simplificación, es posible calcular el valor de f_n con sólo calcular F^{n-1} y hacer una multiplicación por el vector de casos base.

La forma más eficiente de calcular exponenciaciones es mediante la técnica de divide y vencerás, aplicada sobre:

$$F^n = \left(F^{n div 2}\right)^2 F^{n \bmod 2} = \begin{cases} \left(F^{\frac{n}{2}}\right)^2 & \text{si n es par} \\ F\left(F^{\frac{n-1}{2}}\right)^2 & \text{si n es impar} \end{cases}$$

Como puede observarse, reducimos el problema n a uno de n/2 que nos permite obtener una eficiencia de O(log n) frente a O(n) como ocurría si aplicábamos el algoritmo inicial.

Descripción del Esquema Algorítimico

Para resolver el problema de elevar una matriz F a un determinado exponente utilizaremos la técnica de divide y vencerás aplicada a la fórmula de potencia descrita en el apartado anterior.

El esquema algorítmico de divide y vencerás se basa en la idea de dividir un problema en varios subproblemas del mismo tipo cuya solución será combinada posteriormente para obtener la solución al problema inicial.

Como en toda función que resuelva un problema recursivamente, existirán casos umbral para los cuales la solución del problema pueda ser obtenida directamente mediante el uso de una sencilla función.

En nuestro caso concreto:

1- Tamaño umbral y solución simple: En nuestro caso, para n=1 y n=0 existen soluciones triviales y, por tanto, nuestro tamaño umbral lo podemos poner en n=1.
2- Descomposición: Fn siempre lo vamos a descomponer en un único subproblema Fn/2. En este caso, el algoritmo de divide y vencerás realiza una reducción en lugar de una descomposición.
3- Combinación: La función de combinación en nuestro caso será la que hemos desarrollado al final del apartado anterior.

Estructuras de Datos.

En nuestro caso utilizaremos enteros y matrices de 2x2.

Algoritmo Completo.

Suponiendo que conocemos a, b, f0 y f1.

```
fun f(n:entero) dev entero
F ← ⎛a  b⎞
    ⎝1  0⎠
caso    n=0: dev f0
        n=1: dev f1
        verdadero: hacer
                        S ← exp_mat(F,n-1)
                        s ← S⎛f1⎞
                             ⎝f0⎠
                        dev s[1]
fcaso
ffun


fun exp_mat(M:vector[2,2]; n:entero) dev vector[2,2]

si n <= 1 entonces
        dev solucion_simple(M,n)
si no hacer
        p ← n div 2
        r ← n mod 2
        T ← exp_mat(M,p)
        dev combinacion(T,r,M)
fsi
ffun

fun combinación (T:vector[2,2]; r:entero; M:vector[2,2]) dev vector[2,2]
        dev TT*M^r
ffun
```

Donde $M^1=M$ y $M^0=\begin{pmatrix}1 & 0\\ 0 & 1\end{pmatrix}$

```
fun solucion_simple (M:vector[2,2], n:entero) dev vector[2,2]

si n=1 dev M
si no dev
```
$$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$
```
fsi
ffun
```

Estudio del Coste

Función de recurrencia:

$$T(n)=\begin{cases} c\cdot n^k \text{ si } 1 \le n < b \\ a\cdot T(n/b)+c\cdot n^k \text{ si } n \ge b \end{cases}$$
$$a,c\in R^+, k\in R^+\cup\{0\}, n,b\in N, b>1$$

Resolución de la función de recurrencia: k=0, b=2; a=1

$$T(n)=\begin{cases} \Theta(n^k) \text{ si } a < b^k \\ \Theta(n^k\cdot\log n) \text{ si } a = b^k \\ \Theta(n^{\log_b a}) \text{ si } a > b^k \end{cases}$$

De acuerdo a los valores extraídos del caso particular de nuestra función, podemos concluir que la complejidad del problema solucionado será de O(log n) como nos pedían en el enunciado.

(Páginas 55-59 Esquemas Algorítmicos: Enfoque Metodológico y Problemas Resueltos. J. Gonzalo Arroyo y M. Rodríguez Artacho)