Programación III

Indicaciones de las soluciones del examen de septiembre 2008 (ORIGINAL)

Cuestión 1 (1 punto). Escriba el grafo asociado al espacio de soluciones del nonograma (práctica de este año), a partir del nodo que se presenta a continuación. Se supone que a partir de dicho nodo se va a comenzar a explorar la fila 3. (No puntúa la exploración por fuerza bruta de coste $2^{n\hat{}2}$.

		1 1		
		1 2	2	2
2		X	X	
1 2	X		X	X
1 1				
2				

Solución:
En función del algoritmo desarrollado por el alumnos en la práctica.

Cuestión 2 (2 puntos). El algoritmo 'mergesort' posee una complejidad T(n) $\in$ θ (n log n) , describa y demuestre un caso en el que 'mergesort' tiene una complejidad T(n) $\in$ θ (n^2).

Solución:

Si utilizamos como posición de corte el penúltimo o (segundo) elemento del vector de forma consecutiva, el coste será equivalente a

$t(n)=t(n-1)+t(1)+g(n),$ donde $g(n)\in\Theta(n)$ de forma que $t(n)\in\Theta(n^2)$

Cuestión 3 (3 puntos). Considere un array A[1..n] ordenado y formado por enteros diferentes, algunos de los cuales pueden ser negativos. Escriba un algoritmo recursivo que calcule en tiempo logarítmico un índice i tal que 1 ≤ i ≤ n y T[i] = i , siempre que este índice exista, devolviendo -1 si no existe. Se supone que las operaciones elementales tienen coste unitario. Demuestre el coste mediante la ecuación de recurrencia. No justifique el esquema usado, aplíquelo.

Solución:

fun *coincide (A:* ***array****[1..n], ini, fin:****natural****) dev entero***

```
 lon = fin – ini + 1
si lon < 1 entonces dev -1
sino
  si lon = 1 entonces
```

```
        si A[ini] = ini entonces
          dev ini
        sino
          dev -1
      sino
        pos = (fin – ini+1) div 2
        si A[pos] = pos entonces dev pos
        sino
          si A[pos] < pos  entonces
            dev coincide(A, pos+1, fin)
          sino
            dev coincide(A, ini, pos-1])
          fsi
        fsi
    fsi
ffun
```

Ecuación de recurrencia:

$$t(n) = \begin{cases} cn^k & \text{si} \quad 1 \le n < b \\ a\,t(n/b) + cn^k & \text{si} \quad n \ge b \end{cases}$$

La solución depende de los valores de *a,b,k*

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

En este problema: a=1, b=2, k=0, luego el caso es $T(n) \in \Theta(n^k \log n)$, por lo que el coste
es $T(n) \in \Theta(\log n)$

Problema (4 puntos). Una empresa de mensajería tiene n repartidores con distintas velocidades según el tipo de envío. Se trata de asignar los próximos n envíos, uno a cada repartidor, minimizando el tiempo total de todos los envíos. Para ello se conoce de antemano la tabla de tiempos T[1..n,1..n] en la que el valor t[i,j] corresponde al tiempo que emplea el repartidor i en realizar el envío j. Se pide:

- Elección del esquema más apropiado, el esquema general y explicación de su aplicación al problema (0,5 puntos).
- Descripción de las estructuras de datos necesarias (0.5 puntos).
- Algoritmo completo a partir del refinamiento del esquema general (2,5 puntos).
- Estudio del coste del algoritmo desarrollado (0.5 puntos).

Solución:

Determinar qué esquema algorítmico es el más apropiado para resolver el problema.

Se trata de un problema de optimización con restricciones. Por tanto, podría ser un esquema voraz o un esquema de ramificación y poda. Sin embargo descartamos el esquema voraz porque no es posible encontrar una función de selección y de factibilidad tales que una vez aceptado un candidato se garantice que se va alcanzar la solución óptima.
Se trata, por tanto, de un algoritmo de ramificación y poda.

Escribir el esquema general.

```
Función RamificaciónPoda (nodo_raíz) dev nodo
    Montículo:=montículoVacío();
    cota:=acotar(nodo_raíz);
    poner((cota,nodo_raíz), Montículo);
    mientras no vacío(Montículo) hacer
        (cota, nodo):=quitarPrimero(Montículo);
        si solución(nodo) entonces
            devolver nodo;
        si no
            para cada hijo en compleciones(nodo) hacer
                cota:=acotar(hijo);
                poner((cota,hijo),Montículo);
            fpara;
        fsi;
    fmientras
devolver Ø
```

Indicar que estructuras de datos son necesarias.

nodo=tupla
    asignaciones: vector[1..N];
    último_asignado: cardinal;
    filas_no_asignadas: lista de cardinal;
    coste: cardinal;

Montículo de mínimos (cota mejor la de menor coste)

Desarrollar el algoritmo completo.

Las funciones generales del esquema general que hay que instanciar son:

1. a. solución(nodo): si se han realizado N asignaciones (último_asignado==N)
2. b. acotar(nodo,costes): nodo.coste + “mínimo coste de las columnas no asignadas”
3. c. compleciones(nodo,costes): posibilidades para la siguiente asignación (valores posibles para asignaciones[último_asignado+1])
4.

```
Función asignación(costes[1..N,1..N]) dev solución[1..N]
    Montículo:=montículoVacío();
    nodo.último_asignado=0;
    nodo.coste=0;
    cota:=acotar(nodo,costes);
    poner((cota,nodo),Montículo);
    mientras no vacío(Montículo) hacer
        (cota,nodo):=quitarPrimero(Montículo);
        si nodo.último_asignado==N entonces
            devolver nodo.asignaciones;
        si no para cada hijo en compleciones(nodo,costes) hacer
            cota:=acotar(hijo,costes);
            poner((cota,hijo),Montículo);
        fsi;
    fmientras
    devolver ∅;

Función acotar(nodo,costes[1..N,1..N]) dev cota
    cota:=nodo.coste;
    para columna desde nodo.último_asignado+1 hasta N hacer
        mínimo=∞;
        para cada fila en nodo.filas_no_asignadas hacer
            si costes[columna,fila]<mínimo entonces
                mínimo:=costes[columna,fila];
            fsi
        fpara
        cota:=cota+mínimo;
    fpara
    devolver cota;

Función compleciones(nodo,costes[1..N,1..N]) dev lista_nodos
    lista:=crearLista();
    para cada fila en nodo.filas_no_asignadas hacer
        hijo:=crearNodo();
        hijo.último_asignado:=nodo.último_asignado+1;
        hijo.asignaciones=nodo.asignaciones;
        hijo.asignaciones[hijo.último_asignado]:=fila;
        hijo.coste:=nodo.coste+costes[hijo.último_asignado,fila];
        hijo.filas_no_asignadas:=nodo.filas_no_asignadas;
        eliminar(fila,hijo.filas_no_asignadas);
        añadir(hijo,lista);
    fpara;
devolver lista;
```

Coste:

En este caso únicamente podemos hallar una cota superior del coste del algoritmo por descripción del espacio de búsqueda. En el caso peor se generan (k-1) hijos por cada nodo del nivel k, habiendo n. Por tanto, el espacio a recorrer siempre será menor que n!.