Segundo Parcial de Programación 3 (2/12/2008)

Instituto de Computación, Facultad de Ingeniería

- Este parcial dura 4:00 horas y contiene 5 carillas. El total de puntos es **60**.
- En los enunciados llamamos C* a la extensión de C al que se agrega el operador de pasaje por referencia &, y las sentencias *new*, *delete* y el uso de *cout* y *cin*.
- NO se puede utilizar ningún tipo de material de consulta. Salvo que se indique lo contrario, puede usarse todo lo visto en el teórico, práctico y laboratorio sin demostrarlo, indicando claramente lo que se está usando.

Se requiere:

i. Numerar todas las hojas e incluir en cada una el nombre y la cédula de identidad.
ii. Utilizar las hojas de un sólo lado y escribir con lápiz.
iii. Iniciar cada ejercicio en hoja nueva.
iv. Poner en la carátula la cantidad de hojas entregadas, y un índice indicando en qué hojas respondieron cada problema.

Solución

Ejercicio 1. (20 puntos)

a) **(10 puntos)**

Solución:

$f(i,g)$ indica la mayor cantidad de denarios que puede obtener el comerciante por la venta de los animales que puede transportar en su barco, donde *i* representa el i–ésimo animal y *g* el peso en kilos que soporta el barco sin hundirse.

Alternativa 1:

Pasos base:

$$f(i,g) = \begin{cases} -\infty \quad si \quad i < 0 \\ 0 \quad si \quad i \geq 0, g = 0 \end{cases}$$

Pasos recursivos:

$$f(i,g) = \begin{cases} f(i-1,g) \quad si \quad g < k_i, \quad 0 \leq g \leq G \quad y \quad 0 \leq i < n \\ \max\{f(i-1,g), f(i-1,g-k_i) + d_i\} \quad si \quad g \geq k_i, \quad 0 \leq g \leq G \quad y \quad 0 \leq i < n \end{cases}$$

donde k_i indica el peso del animal i-ésimo.

El problema se resuelve con $f(n-1,G)$, donde G es el peso máximo en kilos que puede soportar el barco sin hundirse.

Alternativa 2:

La función $f(i,g)$ tiene solución trivial en los casos:

- $i = n$, donde *n* es el total de animales. La función retorna cero porque se evaluaron todos los animales del comerciante.

- $g < k_i$, siendo k_i el peso del animal *i*. La función continua sin incluir el animal i-ésimo, pues su peso excede la capacidad del barco.

En general, se tienen dos opciones:

- incluir el animal *i* en el barco: $f(i+1, g-k_i)+d_i$, donde k_i es el peso en kilos del animal *i*, y d_i la cantidad de denarios que cuesta el animal *i*.
- no incluir el animal *i* en el barco: $f(i+1, g)$

La función recursiva que resuelve el problema es:

$$f(i,g)=\begin{cases}0 \quad si \quad i=n \\ f(i+1,g) \quad si \quad g<k_i \quad y \quad 0\le i<n \\ \max\{f(i+1,g),\, f(i+1,g-k_i)+d_i\} \quad si \quad g\ge k_i \quad y \quad 0\le i<n\end{cases}$$

El problema se resuelve con $f(0,G)$, donde G es el peso máximo en kilos que puede soportar el barco sin hundirse.

b) **(10 puntos)**

Solución:

Desarrollando $T(n)=T(n/2)+n+\log(n)$ con $n=2^k$ se tiene:

$$T(2^k)=T(2^k/2)+2^k+\log(2^k)$$
$$T(2^{k-1})=T(2^{k-1}/2)+2^{k-1}+\log(2^{k-1})$$
$$T(2^{k-2})=T(2^{k-2}/2)+2^{k-2}+\log(2^{k-2})$$
.
.
.
$$T(2^{k-i})=T(2^{k-i}/2)+2^{k-i}+\log(2^{k-i})$$
.
.
.
$$T(2^1)=T(2^1/2)+2^1+\log(2^1)$$
$$T(2^0)=1$$

Sumando se tiene:

$$T(2^k)=1+\sum_{i=1}^{i=k}\left[2^i+\log(2^i)\right]$$
$$T(2^k)=1+\sum_{i=1}^{i=k}2^i+\sum_{i=1}^{i=k}\log(2^i) \quad (*)$$

Aplicando la propiedad $\log(a.b)=\log(a)+\log(b)$ *i* veces se tiene:

$$T(2^k)=1+\sum_{i=1}^{i=k}2^i+\sum_{i=1}^{i=k}\left[\sum_{j=1}^{j=i}\log(2)\right]$$

$$T(2^k)=1+\sum_{i=1}^{i=k}2^i+\sum_{i=1}^{i=k}\left[i.\log(2)\right]$$

$T(2^k) = 1 + \sum_{i=1}^{i=k} 2^i + \log(2) \sum_{i=1}^{i=k} i$ como $\log_2(2) = 1$ se tiene

$$T(2^k) = 1 + \sum_{i=1}^{i=k} 2^i + \sum_{i=1}^{i=k} i$$

$$T(2^k) = 1 + \sum_{i=1}^{i=k} 2^i + 2.\left[\frac{k(k+1)}{2}\right]$$

Aplicando la propiedad $\sum_{i=1}^{n} q^i = \sum_{i=0}^{n} (q^i) - 1 = \frac{q^{n+1}-1}{q-1} - 1$ se tiene:

$$T(2^k) = 1 + \left(\frac{2^{k+1}-1}{2-1}\right) - 1 + \left[\frac{k(k+1)}{2}\right]$$

$$T(2^k) = 2^{k+1} - 1 + \left[\frac{k(k+1)}{2}\right]$$

Como $n = 2^k$ entonces:

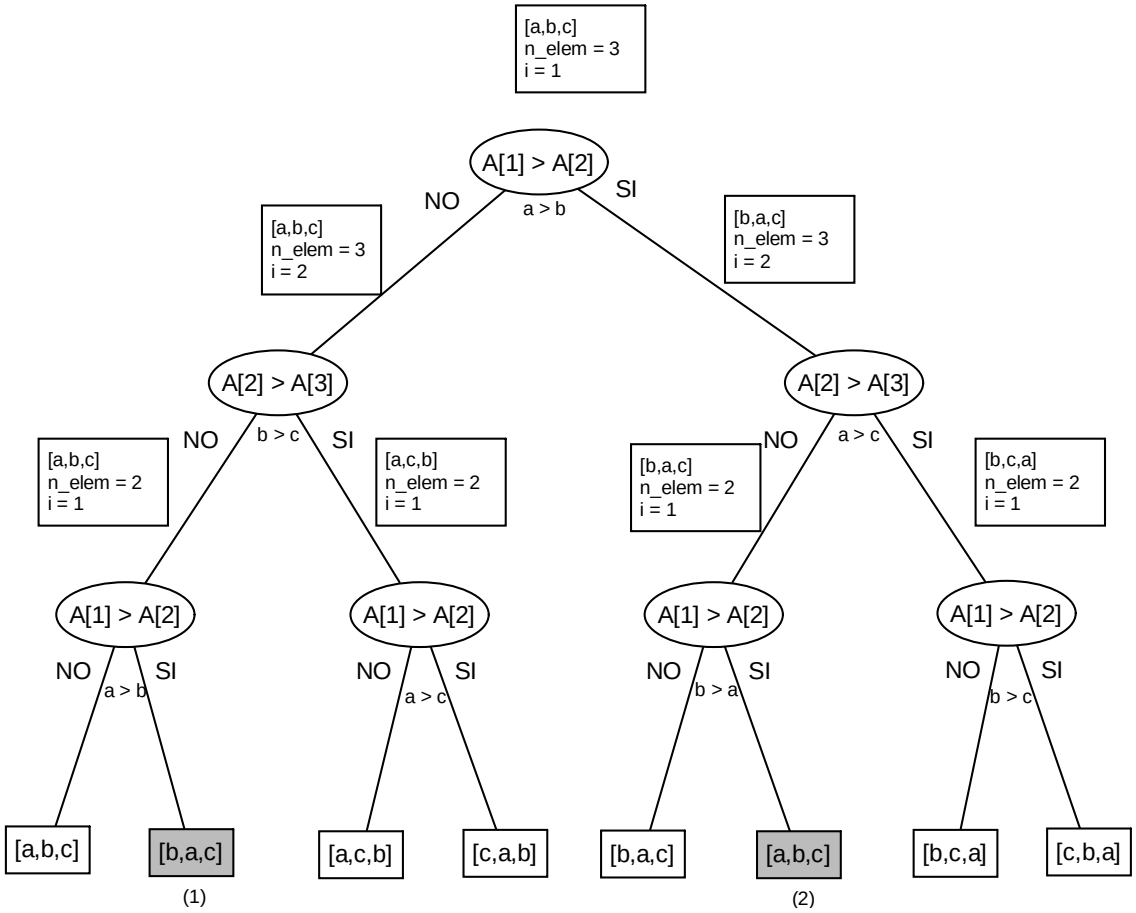
$$T(n) = 2n - 1 + \left[\frac{\log_2 n(\log_2 n + 1)}{2}\right]$$

Nota: el paso (*) también puede escribirse como

$$T(2^k) = 1 + \sum_{i=1}^{i=k} 2^i + \sum_{i=1}^{i=k} \log(2^i) = \quad T(2^k) = 1 + \sum_{i=1}^{i=k} 2^i + \sum_{i=1}^{i=k} i$$

Ejercicio 2. (20 puntos)

a. **(10 puntos)**



(1) Inalcanzable ya que **a** > **b** NO se cumple para esta rama del árbol.
(2) Inalcanzable ya que **a** > **b** para esta rama del árbol.

En el árbol, los nodos internos representan operaciones del algoritmo (comparaciones en esta caso). Un nodo externo representa el fin de una ejecución por la rama que lleva de la raíz a dicho nodo externo.

b. **(3 puntos)**

- Un algoritmo de ordenación es estable si luego de su ejecución mantiene el orden relativo de los elementos de igual valor con respecto a la entrada original.

- Esta implementación *BubbleSort* es estable, ya que al hacer la comparación entre elementos de igual valor no se realiza el intercambio (porque se pregunta por ">" estricto) y por lo tanto se mantiene su orden relativo con respecto a la entrada original.

c. **(4 puntos)**

Dado que se considera como operación básica la comparación de elementos y ésta se hace siempre, el costo del algoritmo es independiente del caso (todos las posibles entradas son peores casos) por lo que:

$$T_A(N) = T_W(N) = \sum_{i=2}^{N}\left(\sum_{j=1}^{i-1}1\right) = \sum_{i=2}^{N} i - 1 = \sum_{i=1}^{N-1} i = \frac{(N-1)N}{2} = \frac{N^2 - N}{2}$$

d. **(3 puntos)**

Sea $T_i(n)$ la cantidad de comparaciones que realiza un algoritmo A_i, para ordenar una secuencia de n elementos. Se denotará $T_{Wi}(n)$ a la cantidad de comparaciones en el peor caso de A_i.

Sea $T_W = \{T_{W1}(n), T_{W2}(n), \ldots, T_{Wq}(n)\}$

$F_W(n) \leq \min T_W$

ó

Sea $F_W(n)$ la cota inferior para el problema de ordenación por comparaciones de elementos 2 a 2, para el peor caso. Análogamente $F_A(n)$ para el caso medio.

Definición Óptimo: cualquier algoritmo A_i que cumpla $T_W(n) = F_W(n)$ comparaciones se dirá que es óptimo en el peor caso. Análogamente vale para el caso medio. *(Esta definición de optimalidad vale para cualquier problema cambiando la palabra "comparaciones" por "operaciones básicas").*

BubbleSort NO es óptimo en ningún caso ya que se puede observar que:

$$T_B(N) = T_A(N) = T_W(N) = \frac{N^2 - N}{2} \neq N \log N$$

Donde: $N \log N = F_W(N) = F_A(N)$ (teórico)

Ejercicio 3. (20 puntos)

a) (**10 puntos**)

Forma de la solución

$s = (x_0, .. , x_{N*N-1})$, tupla de largo fijo N*N donde cada x_i es un par $<b_i, r_i>$ siendo b_i el índice del bloque en la lista de entrada y r_i la rotación, a ser situado en la columna a = i mod N y la fila b = i div N del tablero solución.
Donde:
$B = (B_0, B_1, B_2, \ldots, B_{M-1})$ la secuencia de bloques de entrada, con largo(B)=$M \geq N*N$
P puntaje dado

Restricciones explícitas

a) $x_i = < b_i, r_i >$ con:
   a. $b_i \in$ N (naturales) $0 \leq b_i \leq$ M-1 y
   b. $r_i \in$ N (naturales) $0 \leq r_i \leq 3$

Restricciones implícitas

Sea x'_i = el bloque b_i rotado r_i veces a la derecha

Sea $n_i^{(j)}$ el número de la parte j (0 izquierda, 1 superior, 2 derecha, 3 inferior) en el bloque x'_i,
con $0 \leq j \leq 3$.

Utilizando a y b como se definieron en la forma de la solución, se deberán cumplir las siguientes restricciones:

a) Si dos bloques son adyacentes en la matriz entonces los números enfrentados deben ser iguales.
   Esto es:
   -Si 0 < a => $n_{i-1}^{(2)} = n_i^{(0)}$ [bloques al lado] (notar que a < N por división entera)
   -Si 0 < b => $n_{i-N}^{(3)} = n_i^{(1)}$ [bloques arriba] (notar que b < N por división entera)

b) No se pueden repetir los bloques:

   $\forall x_i, x_j$ con $x_i = < b_i, r_i >$ y $x_j = < b_j, r_j >$, $i \neq j \Rightarrow b_i \neq b_j$

   Notar que esta restricción es independiente de la rotación

c) El puntaje de la solución es menor a *P*

Sea *P'* el puntaje de la solución

$$P' = \sum_{i=0}^{N*N-1} h_i + v_i < P$$

Donde:

h:

-si 0 < a : $h_i = n_i^{(0)}$

-si no: $h_i = 0$

v:

-si 0 < b : $v_i = n_i^{(1)}$

-si no $v_i = 0$

Función objetivo

Encontrar una tupla s tal que su puntaje *P'* sea menor que *P* y lo más cercano posible al mismo.

La función objetivo es f = *max P'* tal que *P* > *P'*, donde *P'* es el puntaje de la tupla solución *s* = (x_0, .. ,x_{N*N-1}) (tal como fue definido en las restricciones implícitas) y *P* es el valor dado

Formalmente, sea S = {s = (x_0, .. ,x_{N*N-1}) / s es factible} , lo que se busca es $\max_{s \in S}$ *P'* / *P* > *P'*

Predicados de poda

No hay.

b) (**10 puntos**)

```
void BuscarMejorSolucion(int P, bool &hay_solucion, Tablero*
&tablero_solucion,Tablero* &tablero_iter, ListaBloque* lista )
{
    if ( TableroCumpleAdyacencia(tablero_iter)   // restriccion implicita a
        && TableroPuntaje(tablero_iter) < P ) // restriccion implicita c
    {
        if (TableroLleno(tablero_iter))
        {
            if (!hay_solucion)
            {
                hay_solucion = true;
                tablero_solucion = TableroCopia(tablero_iter);
            }
            else if (TableroPuntaje(tablero_iter) >
TableroPuntaje(tablero_solucion) ) // funcion objetivo
            {
                TableroDestruir (tablero_solucion);
                tablero_solucion = TableroCopia(tablero_iter);
            }
        }
        else
        {
            ListaBloque* iter = lista;
            while ( !ListaBloqueEsVacia (iter) ) // restriccion explicita a
            {
                Bloque* bloque = ListaBloqueHead (iter);
                if ( !BloqueEstaMarcado(bloque) ) // restriccion implicita b
                {
                    BloqueMarcar(bloque);
                    for (int i=0; i<4; i++) // restriccion explicita b
                    {
                        TableroInsertarBloque(tablero_iter, bloque); // el
tablero incrementa ultima posicion insertada
                        BuscarMejorSolucion( P, hay_solucion,
tablero_solucion,tablero_iter, lista);
                        TableroQuitarBloque(tablero_iter);
                        BloqueRotarIzq(bloque);
                    }
                    BloqueDesmarcar(bloque);
                }
                iter = ListaBloqueTail (iter);
            }
        }
    }
}

Tablero* Juego (int N, int P, ListaBloque* lista)
{
    bool haySolucion = false;

    Tablero* tablero_iter = TableroCrear (N); //forma de la solucion
    Tablero* tablero_solucion;

    BuscarMejorSolucion(P, haySolucion, tablero_solucion, tablero_iter,
lista);
    return tablero_solucion;
}
```