

# Solución de examen de Introducción a la Computación

Centro de Matemáticas Facultad de Ciencias

21 de diciembre de 2004

## Ejercicio 1 (34 puntos.)

1. Definir inductivamente árbol binario de búsqueda.

*Un árbol binario es*

a) *Un árbol vacío*

*o*

b) *Una raíz unida a dos árboles binarios (que llamaremos subárbol izquierdo y subárbol derecho)*

c) *Un árbol binario de búsqueda es un árbol binario etiquetado tal que para todo nodo, las etiquetas de los nodos de su subárbol izquierdo son menores que la etiqueta de dicho nodo y las etiquetas de los nodos de su subárbol derecho son mayores que dicho nodo.*

2. Definir el tipo de datos árbol binario de enteros en Pascal.

```
NODOA = record
    valor : integer;
    izq, der : ^NODOA
end
ABB = ^NODOA
```

3. Definir el tipo de datos lista de enteros en Pascal.

```
NODOL = record
    valor : integer;
    sig : ^NODOL
end
LISTA = ^NODOL
```

4. Escribir la rutina para insertar un elemento en un árbol binario de búsqueda.

```
procedure insertar (k : integer; VAR A : ABB)  
begin  
if (A = NIL) then  
    new(A);  
    A^.valor = k;  
    A^.izq = NIL;  
    A^.der = NIL;  
else if (k < A^.valor) then  
    insertar (k, A^.izq)  
    else insertar (k, A^.der)  
end
```

5. Escribir una rutina que tome como parámetro una lista de enteros, la recorra e inserte todos los nodos que tengan un valor que sea múltiplo de 7 en un árbol binario de búsqueda. Se puede usar como subprograma la función escrita en la parte anterior. La lista original no se modifica.

```
procedure mul7 (L : LISTA; VAR A : ABB)  
begin  
while not (L = NIL) do  
    begin  
        if (L^.valor mod 7 = 0) then insertar (L^.valor, A);  
        L := L^.sig  
    end  
end
```

*Se puede hacer tambien recursivamente:*

```
procedure mul7 (L : LISTA; VAR A : ABB)  
begin  
if not (L = NIL) then  
    begin  
        if (L^.valor mod 7 = 0) then insertar (L^.valor, A);  
        mul7 (L^.sig, A)  
    end  
end
```

6. Modificar el algoritmo anterior para que además borre de la lista original los elementos que sean múltiplos de 7.

Se presenta la versión recursiva del problema:

```
procedure nuevomul7 (VAR L : LISTA, A : ABB)
var aux : LISTA;
begin
  if not (L = NIL) then
  begin
    if (L .valor mod 7 = 0) then
    begin
      insertar (L .valor, A);
      aux := L;
      L := L .sig ;
      dispose(aux);
    end
    mul7 (L .sig, A)
  end
end
```

7. Probar por Inducción Completa que si se recorre en orden un árbol binario de búsqueda, los elementos se devuelven en orden lexicográfico.

Se va a hacer Inducción Completa en la cantidad de nodos del árbol binario de búsqueda.

**Paso Base:** Para  $n=0$ , entonces por la parte a) de la definición de árbol binario de búsqueda, tenemos el árbol vacío. Entonces el algoritmo termina por el paso base e imprime la secuencia vacía, que está ordenada.

**Hipótesis de Inducción:** Para todo  $0 \leq k \leq n$ , si se recorre en orden un árbol binario de búsqueda de  $k$  nodos, se imprime una secuencia ordenada.

**Tesis de Inducción:** Si se recorre en orden un árbol binario de búsqueda de  $n+1$  ( $n \geq 0$ ) nodos, se imprime una secuencia ordenada.

**Prueba:**

Dado que el árbol tiene al menos un nodo, entonces estamos en la parte b) de la definición de árbol binario de búsqueda. Por tal motivo, de manera recursiva se descompone de una raíz y de dos subárboles, cada uno de ellos con una cantidad menor o igual a  $n$  nodos.

Por Hipótesis de Inducción, para cada uno de los subárboles, al recorrerse en orden, se imprime una secuencia ordenada.

Por la parte c) de la definición de árbol binario de búsqueda, las etiquetas de todos los nodos del subárbol izquierdo son menores que la etiqueta de la raíz, y la de los nodos del subárbol derecho son mayores que la etiqueta de la raíz.

Dado que recorrer en orden un árbol binario implica recorrer primero el subárbol izquierdo, luego la raíz y luego el subárbol derecho, vamos a tener que se imprimen ordenadas las secuencias de

los nodos con etiquetas menores y mayores que la raíz. Además, la raíz se imprime luego de recorrer el subárbol izquierdo y antes de recorrer el subárbol derecho. Por tal motivo, se imprime una secuencia ordenada.

## Ejercicio 2 (24 puntos.)

Una niña juega a la rayuela sobre cuadrados ubicados en línea recta. Del casillero  $i$ , puede ir de un salto al casillero  $i + 1$  o al casillero  $i + 2$ . Empieza a saltar del casillero 0. Se desea contar la cantidad de maneras diferentes de llegar al casillero  $n$ . Por ejemplo, para ir al casillero 1 tiene 1 posibilidad (saltar del casillero 0 al 1); para ir al casillero 2 tiene 2 posibilidades (saltar del casillero 0 al 1 y del 1 al 2, o saltar del casillero 0 al 2); para ir al casillero 3 tiene 3 posibilidades (saltar del casillero 0 al 1 al 2 y al 3, o saltar del casillero 0 al 1 y al 3, o saltar del casillero 0 al 2 y al 3), etc. Se pide:

1. Indicar la cantidad de posibilidades que hay para ir a los casilleros 4 y 5. Justificar su respuesta.

*La cantidad de maneras de ir al casillero 4 es cinco. El motivo es que para ir al casillero 4, hay que llegar primero al casillero 3 y luego dar un salto de un casillero o ir primero al casillero 2 y dar luego un salto de 2 casilleros. Entonces la cantidad de maneras de ir al casillero 4 es la cantidad de maneras de ir al casillero 2 (dos) más la cantidad de maneras de ir al casillero 3 (tres), o sea cinco.*

*De manera análoga se puede ver que la cantidad de maneras de ir al casillero 5 es ocho, o sea la cantidad de maneras de ir al casillero 3 (tres) más la cantidad de maneras de ir al casillero 4 (cinco).*

2. Definir recursivamente la secuencia de Rayuela, que calcule la cantidad de manera diferentes de ir del casillero 0 al casillero  $n$ .

$$\begin{aligned} \text{Rayuela}(0) &= 0 \\ \text{Rayuela}(1) &= 1 \\ \text{Rayuela}(n) &= \text{Rayuela}(n - 1) + \text{Rayuela}(n - 2), \quad n > 1 \end{aligned}$$

3. Escribir un programa recursivo para calcular el valor  $n$ -simo de la secuencia Rayuela.

```
function Rayuela (n : integer) : integer;
begin
  if n = 0
  then Rayuela := 0
  else if n = 1
    then Rayuela := 1
    else Rayuela := Rayuela(n - 1) + Rayuela(n - 2)
end
```

4. Indicar dos problemas importantes que tiene implementar Rayuela recursivamente.

*El problema más importante que tiene es que como no guarda resultados intermedios, entonces tiene que computar la misma función con los mismos parámetros muchas veces. Esto hace que el algoritmo, tal cual está implementado tenga tiempo de ejecución exponencial. Otro problema es que como consecuencia no se hace un uso adecuado de la memoria "heap" (la memoria auxiliar usada para implementar la pila de llamadas a funciones), teniendo la pila una cantidad exponencial de llamadas, en vez de una cantidad polinomial.*

5. Escribir una rutina iterativa para calcular el valor  $n$ -simo de la secuencia Rayuela.

```
function Rayuela ( $n$  : integer) : integer;
var ant1, ant2, valor,  $i$  : integer;
begin
  if  $n = 0$ 
  then Rayuela := 0
  else if  $n = 1$  then Rayuela := 1
  else begin
    ant2 := 0;
    ant1 := 1;
    for  $i := 2$  to  $n$  do
      begin
        valor := ant1 + ant2;
        ant2 := ant1;
        ant1 := valor;
      end
      Rayuela := valor;
    end
  end
```

### Ejercicio 3 (40 puntos.)

Un aventurero encuentra un tesoro escondido y decide llevarse la mayor cantidad de dinero en su mochila. El tesoro consiste en  $n$  lingotes de diferentes pesos y valores. El lingote  $i$  tiene peso  $p_i$  y valor  $v_i$ . Los valores de los lingotes no tienen nada que ver con su peso (un lingote liviano puede valer mucho y un lingote pesado puede valer poco). Sin embargo tiene una restricción importante: su mochila puede cargar sólo hasta un peso máximo de  $P$  kilos, antes de romperse.

1. Nuestro amigo decide cargar hasta el tope la mochila, maximizando el valor total del tesoro llevado. Tiene que encontrar una cantidad de lingotes tal que pese exactamente  $P$  kilos, pero que maximice el valor del tesoro llevado. En el caso de no haber ninguna combinación de lingotes que pese exactamente  $P$  kilos, vuelve con la mochila vacía.

Se pide escribir un algoritmo de retroceso para encontrar la manera óptima de cargar la mochila.

En caso de haber más de una solución, se queda con la primera encontrada.

```
procedure tesoro (n, k, P, pact, vact, cant : integer; peso, valor : array[1..n] of integer;  
                VAR total, vmax : integer; solucion, solmax : array[1..n] of integer)  
var i : integer;  
begin  
  if (pact = P) and (vact > vmax) then      (*)  
    begin  
      vmax := vact;  
      for i := 1 to cant do solmax[i] := solucion[i];  
      total := cant;  
    end  
  for i := k to n do  
  begin  
    if (pact + peso[i] <= P) then      (**)  
      begin  
        solucion[cant + 1] := i;  
        tesoro (n, i + 1, P, pact + peso[i], vact + valor[i], cant + 1, peso, valor,  
                total, vmax, solucion, solmax);  
      end  
    end  
  end
```

Se llama inicialmente de la siguiente manera:

```
tesoro(n, 1, P, 0, 0, 0, peso, valor, 0, 0, solucion, solmax)
```

*Notar que no se marca ni se desmarca, pues vamos recorriendo los lingotes en orden creciente de numeración. Por tal motivo, no hay riesgos de contabilizar dos veces el mismo lingote en una solución dada. Esto se asegura con el parámetro k. En la llamada interna en el "for" se llama con el valor  $k = i + 1$ , y en el for, se hace a partir de k, con lo cual se asegura que no se recorran lingotes con numeración menor.*

2. Luego de pensarlo bien, consideró que era un desperdicio dejar el tesoro de lado en el caso de no haber ninguna combinación que pese exactamente  $P$  kilos. Por tal motivo, decidió llevarse una carga de máxima valor pero cuyo peso total sea menor o igual a  $P$ .

Se pide modificar el algoritmo anterior para resolver este problema. Indicar sólo las partes que se modifican del algoritmo.

*La mejor manera es eliminar la condición ( $pact=P$ ) de la instrucción (\*). El motivo es que no se va a pasar del peso máximo por la pregunta en la instrucción (\*\*).*

*Una manera alternativa, es cambiar esta condición por ( $pact \leq P$ ).*

3. Cuando al final de mucho tiempo logró encontrar los lingotes que maximizaban su ganancia con las restricciones de peso de la mochila, vio con gran desazón que la mochila tenía un volumen determinado y que no todos los lingotes le entraban. Por tal motivo se considera ahora que el lingote  $i$  además de su peso  $p_i$  y su valor  $v_i$  tiene un volumen  $V_i$ . La mochila tiene una capacidad máxima de  $V$  centímetros cúbicos. Modificar la rutina anterior para maximizar el valor de lingotes que finalmente se lleva con las restricciones de peso y volumen que tiene.

*Lo únicos cambios consisten en agregar datos relacionados con el volumen, de manera similar en la cual se procesan los datos relacionados con el peso. Se describe la rutina completa.*

```

procedure tesoro ( $n, k, P, V, pact, vact, Vact, cant : integer$ ;
                   $peso, valor, Volumen : array[1..n]$  of integer;
                  VAR  $total, vmax : integer$ ;  $solucion, solmax : array[1..n]$  of integer)
var  $i : integer$ ;
begin
  if ( $pact \leq P$ ) and ( $Vact \leq V$ ) and ( $vact > vmax$ ) then
    begin
       $vmax := vact$ ;
      for  $i := 1$  to  $cant$  do  $solmax[i] := solucion[i]$ ;
       $total := cant$ ;
    end
    for  $i := k$  to  $n$  do
      begin
        if ( $pact + peso[i] \leq P$ ) and ( $Vact + Volumen[i] \leq V$ ) then
          begin
             $solucion[cant + 1] := i$ ;
             $tesoro(n, i + 1, P, V, pact + peso[i], vact + valor[i], Vact + Volumen[i], cant + 1,$ 
                   $peso, valor, Volumen, total, vmax, solucion, solmax)$ ;
          end
        end
      end
    end

```

*Se llama inicialmente de la siguiente manera:*

```

 $tesoro(n, 1, P, V, 0, 0, 0, 0, peso, valor, Volumen, 0, 0, solucion, solmax)$ 

```