Métodos algorítmicos en resolución de problemas I

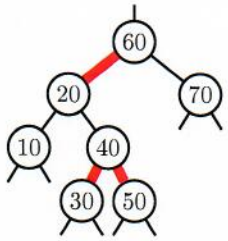
Grado en Ingeniería Informática
Doble Grado en Ingeniería Informática y Matemáticas

EXAMEN FINAL DE FEBRERO — Curso 2020–2021

---

1. Árboles [puntos 1,25]

Ilustrar con suficientes dibujos y explicar el proceso de inserción de la clave 55 en el siguiente árbol LLRB:



2. Montículos [puntos 2,25]

Un montículo paralelo de mínimos tiene la misma estructura de árbol binario casi completo que los montículos de Williams, pero almacena p valores por nodo, siendo $p \geq 1$ un parámetro fijo de la estructura. Su invariante consiste en que todos los valores de un nodo son más pequeños que los valores de sus hijos, y todos los valores de un hermano son más pequeños que todos los valores del otro (no importa cuál sea el hermano "más pequeño").

Los algoritmos son también similares. Una operación de inserción de p elementos procede colocando los nuevos elementos en el primer nodo no utilizado del último nivel y flotando a continuación dicho nodo hasta restaurar el invariante. El borrado de los p mínimos consiste en suprimir el último nodo del árbol —copiándolo previamente en la raíz— seguido de hundir dicho nodo hasta restaurar el invariante.

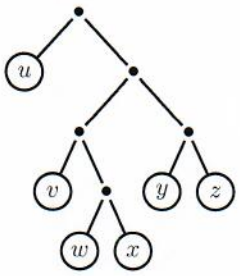
Se pide programar en pseudocódigo las operaciones de flotar, insertar, hundir y eliminar, y determinar su coste en función del cardinal n del montículo y del parámetro p de multiplicidad.

3. Grafos [puntos 2,25]

Se dice que un grafo no dirigido $G = (V, E)$ es k-coloreable si todos los vértices de G pueden ser coloreados usando k colores diferentes de manera que no haya dos vértices adyacentes que tengan el mismo color. Diseñar un algoritmo cuyo coste en tiempo esté en $O(|V| + |E|)$ que coloree un grafo con dos colores o determine que el grafo no es 2-coloreable.

4. Voraz [puntos 1,25]

Dado un fichero en el que aparecen los caracteres u, v, w, x, y y z con frecuencias f_u, f_v, f_w, f_x, f_y y f_z, respectivamente, el algoritmo de Huffman ha calculado el siguiente árbol:



7
3

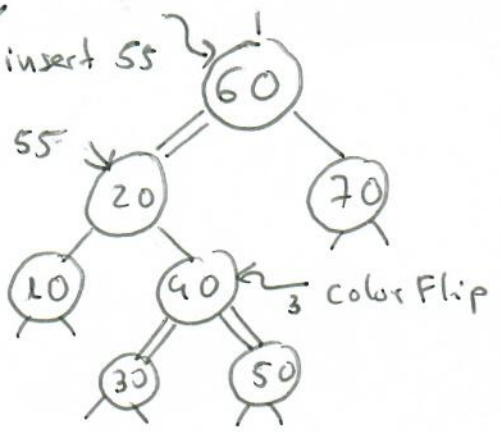
Conjeturar y justificar valores posibles para dichas frecuencias como porcentajes de la longitud del fichero. Indicar también el porcentaje de compresión del fichero que se obtendrá frente a una codificación de longitud fija de 3 bits.

Tiempo: 3 horas
Calificación sobre 7 puntos

1

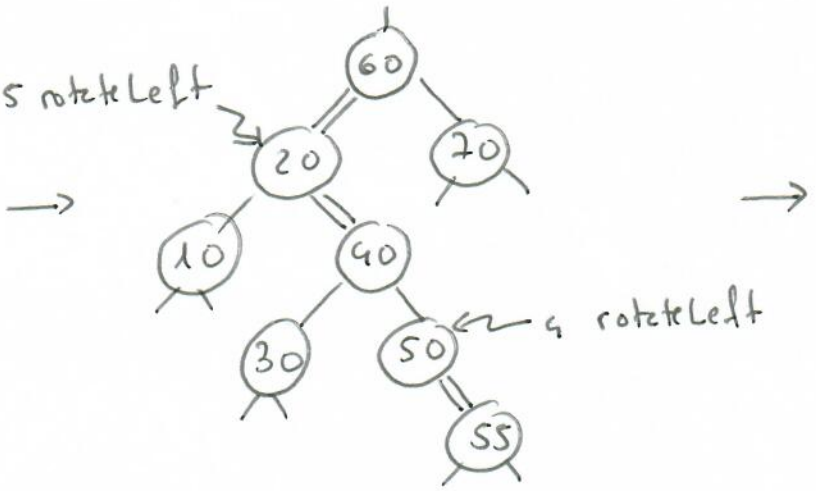
1 insert 55

2 insert 55



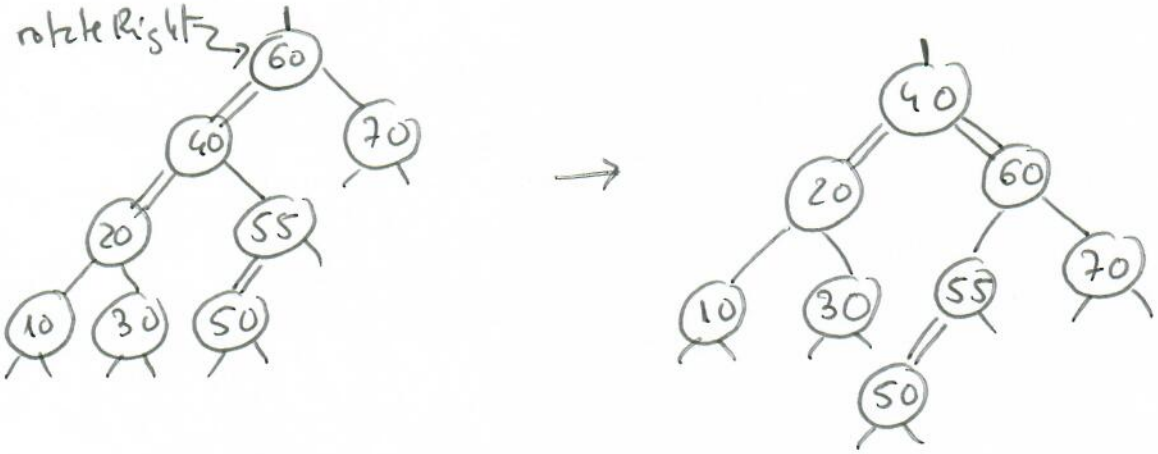
3 colorFlip

5 rotateLeft



4 rotateLeft

6 rotateRight



2/

```
clase MonPar {
    v : vector [1..n] de nodo ; // invariante: monticulo parcial de mínimos
    k : 0..n ; // cardinal
    tipo nodo = vector [1..p] de clave   // invariante: claves ordenadas ≤

    metodo insertar (valores : nodo) {
        si k = n  { error "monticulo lleno" }
        sino  { k := k+1 ; ordenar(valores) ; v[k] := valores; flotar() }
                                  ∈ O(p log p)                ∈ O(p log n)
    }
    metodo borrarMin () {
        si k = 0  { error "monticulo vacio" }
        sino  { v[1] := v[k] ; k := k-1 ; hundir() }
                                          ∈ O(p log n)
    }
    ... // metodos privados flotar y hundir (en siguiente página)
}
```

```
método flotar () {
    aux : vector [1..2p] de clave // para los metodos ordenados
    j := k ;  // j indica el nodo en conflicto con su padre
    mientras j > 1 {
        // asignamos los p valores menores al padre, el resto al hijo
        aux := merge (v[j], v[j/2]) ; // coste en O(p)
        v[j/2] := aux [1..p] ; v[j] := aux [p+1..2p] ;
        // determinamos el mayor hermano y le asignamos los p mayores
        si par(j) ∧ j+1 ≤ k {h := j+1} sino si ¬par(j) {h := j-1} sino {h = 0}
        si h ≠ ∅ { // hay hermano
            si max(v[j]) ≥ max(v[h]) {hg := j ; hp := h}
            sino                     {hg := h ; hp := j}
            aux := merge (v[hp], v[hg]) ;
            v[hp] := aux [1..p] ; v[hg] := aux [p+1..2p] ;
        }
        j := j/2   // nuevo nodo en conflicto es el padre
    }
}
método hundir () {
    aux := vector [1..2p] de clave // para los metodos ordenados
    j := 1 ;  // j indica el nodo en conflicto con sus hijos
    mientras 2j ≤ k { // existe al menos un hijo
        si 2j+1 ≤ k ∧ max (v[2j+1]) ≤ max(v[2j]) {hp := 2j+1 ; hg := 2j}
        sino                                     {hp := 2j ; hg := 2j+1}
        // combinamos los valores de j con el menor hijo
        aux := merge (v[j], v[hp]) ;
        v[j] := aux [1..p] ; v[hp] := aux [p+1..2p] ;
        si 2j+1 ≤ k { // determinar el mayor hermano y combinar
            si max (v[2j+1]) ≤ max(v[2j]) {h1 := 2j+1 ; h2 := 2j}
            sino                          {h1 := 2j ; h2 := 2j+1}
            aux := merge (v[h1], v[h2])
            v[h1] := aux [1..p] ; v[h2] := aux [p+1..2p]
        }
        j := h.p  // nuevo nodo en conflicto es el hijo menor
    }
}  // Coste de flotar() y hundir() en O(p log n)
```

3/ . modificamos el recorrido en anchura BFS

. usamos tres colores: undef (no alcanzado), A y B

```
color[1] := A; color[2..n] := undef; posible := T;
Q := new Queue(); Q.insert(1);
mientras ¬vacia(Q) ∧ posible {
    u := Q.first(); Q.removeFirst(); c := color[u]
    para v ∈ G.ady(u) {
        si color[v] == undef
            color[v] := c̄; // el complementario de A es B y viceversa
            Q.insert(v) // v ha sido alcanzado y hay que visitar sus
                        // adyacentes
        sino si color[v] == c {
            posible := F // (u,v) une dos vértices del mismo color
        } // si no, color[v] = c̄ es correcto y v ya ha
          // sido visitado, no se añade a Q
    }
    retorna posible // lanzar más BFS desde el main si quedan vértices
                    // sin colorear después del primero
```

coste en $O(a+u)$

4/ Tienen que cumplirse las siguientes restricciones:

$$f_w < f_x < f_v < f_w + f_x < f_y < f_z < f_v + f_w + f_x < f_y + f_z < f_u < f_v + f_w + f_x + f_y + f_z$$

por ejemplo: $f_u = 41$, $f_v = 10$, $f_w = 6$, $f_x = 7$, $f_y = 16$, $f_z = 20$

que suman 100

fichero sin comprimir = $(41+10+6+7+16+20) \times 3 = 300$

fichero comprimido = $41 \times 1 + (10+16+20) \times 3 + (6+7) \times 4 = 231$

factor de compresión = $\frac{231}{300} = 77\%$