

Estructuras de Datos

Curso 2012–2013. Convocatoria de Febrero

Grado en Ing.Informática y Doble Grado en Ing.Informática y Matemáticas

1. (1 punto) Usando la notación O , determinar la eficiencia de la siguiente función (n es una potencia de 2):

```
int eficexamen(bool existe)
{
    int sum2=0; int k,j,n;
    if (existe)
        for(k=1; k<=n; k*=2)
            for(j=1; j<=k; j++)
                sum2++;
    else
        for(k=1; k<=n; k*=2)
            for(j=1; j<=n; j++)
                sum2++;
    return sum2;
}
```

2. (1 punto) Rellenar la siguiente tabla con la eficiencia de las operaciones de las filas en las estructuras de datos de las columnas.

| | Vector ordenado | Lista ordenada | Pila | ABB | AVL | Tabla Hash |
|------------------------|-----------------|----------------|------|-----|-----|------------|
| Buscar | | | | | | |
| Insertar | | | | | | |
| Borrar | | | | | | |
| Imprimir ordenadamente | | | | | | |

3. (2 puntos) Una empresa mantiene datos sobre personas (DNI, apellidos, nombre, domicilio y teléfono, todos de tipo string) y necesita poder hacer búsquedas sobre ellos tanto por DNI como por apellidos. Define una representación eficiente (discutiendo las alternativas) para manejar los datos de esta empresa e implementa las operaciones de inserción y de borrado de una persona en dicha estructura. Las cabeceras serían:

- `void datos::insertar(const persona &p)`
- `void datos::borrar(const persona &p)`

4. (2 puntos) Dadas dos listas de intervalos cerrados $[ini, fin]$, $L1$ y $L2$, donde para cada lista los intervalos se encuentra ordenados por orden de tiempo de inicio satisfaciendo que si un intervalo $it1$ está antes que otro $it2$ en la lista entonces $it1.fin < it2.ini$.

Diseña un método que permita seleccionar un (sub)intervalo de $L1$ y lo pase a $L2$. En este proceso podría ser necesario el dividir un intervalo de $L1$ así como fusionar intervalos en $L2$ cuando ocurran solapamientos.

```
typedef pair<int,int> intervalo;
```

`bool Extraer(list<intervalo> & L1, intervalo x, list<intervalo> & L2);`
Devuelve true si ha sido posible realizar la extracción (x debe pertenecer a un único intervalo de L1).

Por ejemplo, si $L1 = [1,7], [10,14], [18,20], [25,26]$; $L2 = [0,1], [14,16], [20,23]$ y $x = [12,14]$ entonces las listas quedarán como $L1 = [1,7], [10,11], [18,20], [25,26]$ y $L2 = [0,1], [12,16], [20,23]$. De igual forma, si $L1 = [1,7], [10,22], [25,26]$; $L2 = [0,1], [14,16], [20,23]$ y $x = [12,20]$ entonces las listas quedarán como $L1 = [1,7], [10,11], [21,22], [25,26]$ y $L2 = [0,1], [12,23]$.

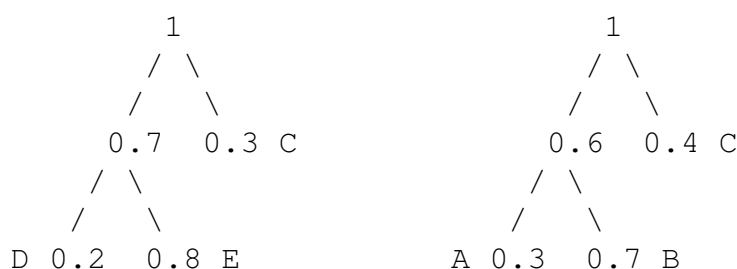
5. (2 puntos) Un árbol de sucesos es un árbol binario donde cada nodo tiene asociada una etiqueta con un valor real en el intervalo $[0,1]$. Cada nodo que no es hoja cumple la propiedad de que la suma de los valores de las etiquetas de sus hijos es 1. Un suceso es una hoja y la probabilidad de que éste ocurra viene determinada por el producto de los valores de las etiquetas de los nodos que se encuentran en el camino que parte de la raíz y acaba en dicha hoja. Se dice que un suceso es probable si la probabilidad de que ocurra es mayor que 0.5. Usando el TDA árbol binario:

(a) Diseñar una función que compruebe si un árbol binario A es un árbol de sucesos. Su prototipo será `bool check_rep (const bintree<float> &A)`

(b) Diseñar una función que indique si existe algún suceso probable en el árbol de probabilidades A. Su prototipo será:

`bool probable (const bintree<float> & A)`

Ejemplo: En el árbol de probabilidades de la derecha no existe un suceso probable porque los tres sucesos tienen probabilidad inferior a 0.5 (A: $1.0 \cdot 0.6 \cdot 0.3 = 0.18$; B: $1.0 \cdot 0.6 \cdot 0.7 = 0.42$; C: $1.0 \cdot 0.4 = 0.4$). En cambio, en el árbol de la izquierda hay un suceso probable: E, porque $1.0 \cdot 0.7 \cdot 0.8 = 0.56$



6. (2 puntos) Supongamos que en una Tabla Hash abierta hacemos uso en cada cubeta de árboles AVL en lugar de listas.

- ¿Cual es la eficiencia (en el caso peor) de la función buscar?
- Mostrar la tabla resultante de insertar los elementos {16, 72, 826, 1016, 12, 42, 623, 22, 32}.

| x | 16 | 72 | 826 | 1016 | 12 | 42 | 623 | 22 | 32 |
|------|----|----|-----|------|----|----|-----|----|----|
| h(x) | 6 | 2 | 6 | 6 | 2 | 2 | 3 | 2 | 2 |

- Construir un APO a partir del anterior conjunto de claves y a continuación elimina el elemento más pequeño.

Notas

- Tiempo para realizar el examen: 3 horas.