Universidad de Valladolid

Departamento de Informática

Estructuras de Datos, 2º I.T.I. de Gestión

Examen ordinario. 3 de febrero de 2001

Apellidos	
Nombre	D.N.I.

1.- El tiempo disponible para la realización del examen es de 2,5 horas.

2.- Sólo hay una respuesta válida para cada pregunta del test.

3.- En el test, una respuesta válida suma 1/2 de punto, una respuesta erronea resta 1/8 de punto y dejar una cuestión sin responder ni

TEST (7,5 PUNTOS)

1. Para un mismo problema, existen tres algoritmos que lo resuelven, de complejidades $O(n^3)$, $O(n^2)$ y $O(n)$, respectivamente. Si el tamaño de los datos de entrada es $n = 100$, ¿Cuál de ellos se ejecutará más rápidamente?
   - ☐ El algoritmo de orden $O(n^3)$
   - ☐ El algoritmo de orden $O(n^2)$
   - ☐ El algoritmo de orden $O(n)$
   - ☐ Depende de la constante de proporcionalidad de cada uno.

2. Se dispone de un vector, *V*, que almacena números enteros en orden estrictamente creciente, y se desea averiguar si existe algún elemento que cumpla *V[i]=i*. ¿Cuál sería la estrategia más adecuada para resolver el problema?
   - ☐ Programación dinámica.
   - ☐ Algoritmo voraz.
   - ☐ Divide y vencerás.
   - ☐ Ninguna de las anteriores es aplicable al problema.

3. La función de Ackerman se define recursivamente de la siguiente forma:

$$A(n,m) = \begin{cases} m+1 & \text{si } n = 0 \\ A(n-1,1) & \text{si } m = 0 \\ A(n-1, A(n, m-1)) & \text{caso contrario} \end{cases}$$

   Donde los parámetros y el resultado son enteros no negativos. Si se implementa directamente esta fórmula, se obtiene un algoritmo de tiempo no polinómico. ¿Sería adecuado aplicar la técnica de la tabla de resultados parciales (definida en la estrategia de programación dinámica) para calcular los valores de la función para un rango dado de los parámetros *n* y *m*?
   - ☐ Si, y el orden pasaría a ser polinómico.
   - ☐ Si, pero el orden no cambiaría.
   - ☐ No, porque no se cumple el principio de suboptimalidad.
   - ☐ No, porque se desconoce cuál podría ser el tamaño necesario para la tabla.

4. Se desea implementar el algoritmo de ordenación rápida (quicksort) para aplicarlo a vectores que estan casi ordenados. A la hora de elegir el elemento pivote para la partición de los subvectores, ¿Cuál sería la elección más adecuada para este caso concreto?
   - ☐ El primer elemento del subvector.
   - ☐ El último elemento del subvector.
   - ☐ El elemento que se encuentra en la posición media del subvector.
   - ☐ La elección del elemento pivote no influye en el rendimiento del algoritmo.

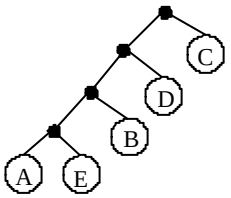
5. Se dispone de un vector que almacena *n* enteros cuyos valores pertenecen al rango 1 .. *m*. ¿Cuál sería el orden del tiempo empleado en realizar una ordenación por recuento sobre el vector?
   - ☐ $O(n)$
   - ☐ $O(m)$
   - ☐ $O(n+m)$
   - ☐ $O(n \cdot \lg n)$

6. ¿Cual de las siguientes implementaciones de un TAD bicola garantiza que el tiempo de las operaciones de inserción y borrado al principio y al final es de orden $O(1)$?
   - ☐ Vector lineal.
   - ☐ Vector circular.
   - ☐ Lista lineal doblemente enlazada.
   - ☐ Lista circular simplemente enlazada.

7. ¿Cual de las siguientes afirmaciones es cierta?
   - ☐ Un montículo es un arbol binario de búsqueda.
   - ☐ Un montículo es un arbol binario equilibrado.
   - ☐ Un recorrido inorden sobre los elementos de un montículo los recorre en orden creciente.
   - ☐ El número de elementos de un montículo es $2^h - 1$, donde *h* es la altura del arbol.

8. ¿Cual sería el orden de la operación de búsqueda sobre un arbol binario de búsqueda no equilibrado que contiene *n* elementos?
   - ☐ $O(\lg n)$ en todos los casos.
   - ☐ $O(n)$ en el peor caso, $O(\lg n)$ en los casos promedio y mejor
   - ☐ $O(n)$ en los casos peor y promedio, $O(\lg n)$ en el mejor.
   - ☐ $O(n)$ en todos los casos.

9. El arbol de Huffman de la figura se ha construido a partir de una de las siguientes listas de pares (símbolo : frecuencia de aparición). ¿Cuál es la lista compatible con el arbol?
   - ☐ (A:1), (B:2), (C:3), (D:4), (E:5)
   - ☐ (A:5), (B:4), (C:3), (D:2), (E:1)
   - ☐ (A:1), (B:10), (C:100), (D:20), (E:1)
   - ☐ Todas las anteriores.



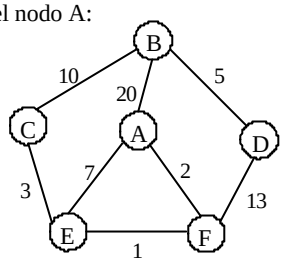
10. Indicar cuál sería la lista de nodos hoja (los que contienen símbolos) obtenida al aplicar un recorrido postorden al arbol de la cuestión anterior:
    - ☐ A, E, B, D, C.
    - ☐ C, D, B, E, A.
    - ☐ E, D, C, B, A.
    - ☐ Ninguna de las anteriores.

11. Tras la inserción de un nodo en un arbol AVL, ¿Cuantas rotaciones son necesarias, como máximo, para reequilibrar el arbol?

- ☐ Ninguna, siempre es posible insertar el nodo de forma que no se altere el equilibrio del arbol.
- ☐ Una, tras la rotación se reestablece la altura original del subarbol.
- ☐ Tantas como ascendientes tenga el nodo insertado.
- ☐ Tantas como nodos tenga el arbol.

12. Dado el grafo no dirigido de la figura, indicar cuál sería el orden en que se seleccionarían (pasan a pertenecer al arbol) los nodos al aplicar el algoritmo de Prim comenzando por el nodo A:

- ☐ A, F, E, C, B, D
- ☐ A, B, C, E, D, F
- ☐ A, F, E, C, D, B
- ☐ Ninguno de los anteriores



13. Dado el grafo de la cuestión anterior, indicar cuál sería el orden en que se seleccionarían (pasan a estar explorados) los nodos al aplicar el algoritmo de Dijkstra sobre el nodo A:

- ☐ A, F, E, C, B, D
- ☐ A, B, C, E, D, F
- ☐ A, F, E, C, D, B
- ☐ Ninguno de los anteriores.

14. Sobre una tabla de dispersión cerrada de tamaño 6, usando exploración lineal, se insertan las claves 1, 13, 7, 2, 5, 4. Al final los elementos de la tabla están en el orden:

- ☐ 1, 7, 13, 2, 4, 5
- ☐ 4, 1, 13, 7, 2, 5
- ☐ 1, 13, 7, 4, 2, 5
- ☐ Ninguno de los anteriores.

15. Respecto de la afirmación siguiente: "El tamaño de una tabla de dispersión cerrada con desplazamiento cociente debe ser un número primo".

- ☐ Es falsa, el tamaño de la tabla no influye.
- ☐ Es falsa, lo que se requiere es que el tamaño sea potencia de dos.
- ☐ Es cierta, de esa manera se garantiza que la exploración recorre todas las posiciones una única vez.
- ☐ Es cierta porque la función de dispersión sólo devuelve números primos.

PROBLEMA (2,5 PUNTOS)

En una aplicación se desea definir una estructura (que llamaremos **grupo**) para representar a una colección de enteros sobre la cuál se van realizar únicamente las siguientes operaciones:

- **vacio** : Crea un grupo vacío.
- **insertar**(n) : Añade el entero n al grupo, independientemente de si existen o no otros enteros con el mismo valor ya almacenados en él.
- **borrar**(n_1, n_2) : Elimina del grupo todos los enteros cuyo valor este en el rango [n_1.. n_2] (es decir, los menores o iguales a n_1 y los mayores o iguales a n_2). Como requisito para usar esta operación debe cumplirse que $n_1 \leq n_2$.
- **cuantos_hay**(n_1,n_2) : Devuelve un valor entero indicando cuantos elementos del grupo (incluyendo los elementos repetidos) pertenecen a ese rango. Tambien se debe cumplir que $n_1 \leq n_2$.
- **todos_menores_que**(n) : Devuelve un valor lógico, cierto si todos los elementos del grupo tienen un valor estrictamente menor que n y falso en caso contrario.

a) Definir un TAD que represente esta estructura (incluyendo el apartado de ecuaciones).

b) Indicar brevemente y de modo textual (no es necesario escribir pseudocódigo) cómo se realizarían las operaciones *insertar*, *borrar*, *cuantos_hay* y *todos_menores_que* para cada una de las implementaciones siguientes:

- Una lista circular simplemente enlazada.
- Un vector lineal ordenado.
- Un montículo.
- Un arbol AVL.

Nota: Se puede hacer referencia a operaciones *propias* de estas implementaciones (reestructurar el montículo, efectuar una rotación en el caso del arbol AVL, etc.) sin necesidad de explicarlas.

c) Escribir una tabla indicando el orden de complejidad de cada operación en cada una de las implementaciones anteriores, suponiendo que el grupo contiene n elementos y que en las operaciones **borrar** y **cuantos_hay** la mitad de los elementos se encuentran en el intervalo dado.