# EXAMEN AED 2023-2024

Anotación: Suele poner prácticamente el mismo examen desde antes de 2017.

1. Explicar ventajas y desventajas de un árbol AVL frente a un árbol binario de búsqueda ordinario. Explicar ventajas y desventajas de un árbol B+ frente a un árbol AVL.
2. Te muestra un grafo de las ciudades de España más importantes y los arcos (valorados) representan las carreteras. 5 preguntas que hay que responder y razonar brevemente.
    a) Elegir entre una estructura estática o dinámica.
    b) Qué algoritmo usarías para obtener las ciudades que si las eliminas dejan otras ciudades totalmente desconectadas.
    c) Qué algoritmo usarías para calcular la distancia mínima entre todas las ciudades.
    d) Qué algoritmo usarías para calcular la cantidad máxima de coches que pueden ir de una ciudad a otra (problema de flujo).
    e) Qué algoritmo usarías para saber si puedes eliminar carreteras de tal forma que se pueda ir a todas las ciudades desde cualquier ciudad y el coste sea mínimo.

3. Te da un ejemplo, y te dice 3 casos en los que se necesita almacenar cierta información de cierta forma. Elegir la mejor estructura entre (pila, cola, lista, árbol y grafo).
4. Desarrollar ventajas en inconvenientes en cuanto al tiempo y el espacio de almacenamiento de inserción y búsqueda de:
    a) Una tabla hash ideal (sin colisiones) y una lista enlazada.
    b) Una tabla hash con encadenamiento y por recolocación
5. Te da el código de una función para un algoritmo voraz y tienes que definir:
Variables: S, C, x; y Funciones: Seleccionar, factible, solución, inserir.
6. Explicar el procedimiento para ramificación y poda en caso de minimización. Cuál es la principal diferencia entre ramificación y poda y Backtracking en cuanto a la ramificación.

## Doblea R.O.

## EXAMEN AED 2014

1. Comparar la implementación de la estructura interna de los árboles AVL y los árboles binarios ordinarios, los árboles B y B +. Ventajas de árboles B frente AVL.

AVL y árboles binarios ordinarios

Estos árboles mantienen una estructura donde cada nodo padre puede tener como máximo 2 hijos. Aparte de esto, los árboles binarios ordinarios no tienen condiciones que cumplir que

permitan evitar el desbalance de dicho árbol por lo que se puede encontrar con árboles muy desaprovechados.

Por otra parte, los AVL mantienen la estructura descrita de los árboles binarios ordinarios, más son árboles binarios de búsqueda (árboles donde cada nodo tiene a su izquierda elementos que valen menos que él y a su derecha elementos que valen más) equilibrados donde cada nodo cuenta con un campo llamado factor de equilibrio que permite asegurar el equilibrio del árbol.

A diferencia de los árboles binarios ordinarios, los AVL tienen reglas de reestructuración, que consisten en rotaciones, cuando el árbol se desbalancea, utilizando el factor de equilibrio como base para reestructurar y mantener el equilibrio.

B y B+

Los árboles B, se caracterizan por ser totalmente equilibrados donde todas las hojas se encuentran en el mismo nivel, y los nodos se agrupan en páginas a las que se accede en bloques. Estas estructuras de datos cumplen diferentes características a partir de un orden m que permite mantener su organización, necesitando reorganizar el árbol cuando no se cumplen diferentes condiciones.

Los árboles B+ son optimizaciones de los árboles B que permite conseguir un recorrido secuencial más rápido. Todos los elementos se encuentran en las páginas hoja manteniendo algunos duplicados en páginas de niveles superiores que actúan como índices.

De esta forma, los árboles B+ requieren de usar más espacio pero suponen un aspecto positivo en árboles que requieren frecuentes modificaciones ya que evita la reorganización del árbol siendo una operación costosa.

Ventajas B frente AVL

Los árboles B presentan una ventaja sobre los árboles AVL cuando se gestionan cantidades de datos muy grandes, donde el uso de nodos en AVL sería inviable en memoria principal y el recurso de la memoria externa tardaría mucho tiempo en encontrar los datos. De esta forma, con árboles B se permite reducir significativamente la altura del árbol y en el caso de que los nodos se agrupen en páginas conlleva traer varios nodos de golpe reduciendo el número de accesos necesarios.

---

2. Montículos. Organización de los datos y ejemplo de aplicación.

Los montículos son árboles binarios completos que se caracterizan por dar soporte a operaciones del TAD cola de prioridad.

Pueden ser montículos de mínimos (donde los elementos de menor valor tienen mayor prioridad) o de máximos (los elementos de mayor valor tienen mayor prioridad) en los que

los hijos de cada nodo son elementos de menor prioridad que este. Cumple un orden parcial ya que no es tan estricto como los ABB pero lo es más que una ordenación aleatoria.

Dentro de sus aplicaciones se puede encontrar la planificación de procesos en un sistema multiusuario, gestión de enfermos en servicio de urgencias o planificación de futuras operaciones. Situaciones donde se desea saber quién va siguiente como en aeropuertos, aparcamientos, colas de impresión, etc.

---

3. Recorrido en anchura de un grafo.

A partir de un nodo dado hay que ir visitando los nodos adyacentes que no estuvieran ya visitados, repitiendo estos pasos hasta visitarlos todos.
Siendo un recorrido no recursivo, requiere de una estructura auxiliar donde se usa una cola para recoger los nodos adyacentes al actual que no estén visitados.

---

4. Ventajas y desventajas de usar lista de adyacencia o matriz de adyacencia.

Ventajas lista de adyacencia:
- Ocupa menos memoria que la matriz de adyacencia cuando el número de arcos es mucho menor que el número de nodos, al evitar tener que almacenar todos los 0s de la matriz.

Desventajas lista de adyacencia:
- La representación es más compleja, al contar con varios punteros.
- Es ineficiente para encontrar los arcos que llegan a un nodo, pues hay que recorrer todas las listas viendo en cada una si hay un arco al elemento buscado.

Ventajas matriz de adyacencia:
- Eficiencia a la hora de obtener los costes asociados a un arco en el caso de la matriz de pesos.
- La comprobación de adyacencia entre 2 nodos cualesquiera es inmediata e independiente del número de nodos.

Desventajas matriz de adyacencia:
- No se permite la eliminación de nodos del grafo ya que no se pueden suprimir filas ni columnas en la matriz.
- En grafos dispersos, con pocos arcos, se producen matrices con muchos 0, lo que es un ejemplo de espacio desaprovechado.

---

5. Grafo dirigido.

    a. ¿Cómo saber si hay un camino entre CADA par de vértices?

Verificando el par de vértices en la matriz de caminos proporcionada por el algoritmo de Warshall.
Este algoritmo se basa en la relación entre las matrices Pk-1 y Pk que nos permite, partiendo de P0 (matriz de adyacencia), encontrar Pn=P (matriz de caminos) por sucesivas iteraciones.

¿VERIFICANDO SI EL GRAFO ES FUERTEMENTE CONEXO?

b. ¿Cómo saber el camino mínimo entre cada par de vértices?

(Floyd)
Se podría saber utilizando el recurso del algoritmo de Floyd o con el de Dijkstra.
El de Floyd se basa en el de Warshall y consiste en tener una matriz D donde cada elemento Dij sea el coste mínimo de los caminos que van del vértice i al j.

c. ¿Cómo saber puntos críticos del grafo?

(puntos de articulación)
Realizando un recorrido en profundidad recursivo del grafo se puede representar con un árbol de recubrimiento.

---

6. Aplicar algoritmo de Prim/Kruskal

Son algoritmos que buscan el árbol de expansión de coste mínimo.

Algoritmo de Prim: algoritmo voraz donde en cada paso se añade el arco más corto al árbol a partir de vértices adyacentes.

Algoritmo de Kruskal: con un grafo T idéntico al grafo G original pero sin arcos, se van examinando los arcos de A en orden creciente de coste y si conecta a componentes conexas distintas se añade a T.

---

7. Relacionar pasos del algoritmo de Kruskal con las funciones de un algoritmo voraz genérico Solución(S), insertar(S,x), factible(S,x)...

C: conjunto de candidatos serían todos los arcos del grafo.
S: candidatos ya elegidos serían uno a uno los arcos que se añaden al grafo T.
R: arcos que fueron rechazados, por ejemplo si es un coste mínimo pero de vértices ya conectados.

Solución(S): verifica que todo el grafo sea solo una componente conexa
Seleccionar(C): selecciona el arco con coste mínimo de los candidatos.
Factible(S,x): verifica si el candidato seleccionado sería el indicado en la iteración, si cumple la condición de que una componentes conexas distintas.
Insertar(S,x): añade un elemento x al conjunto S de candidatos seleccionados para la solución.
Objetivo(S): dada la solución S, devuelve el coste mínimo asociado al árbol.

---

8. Tablas hash. ¿Qué es una colisión? Explicar la recolocación cuadrática.

Las tablas Hash son estructuras de datos que permiten realizar búsquedas de los elementos mediante el uso de claves y proporciona un tiempo de búsqueda constante aunque los elementos no estén ordenados.

A partir de funciones matemáticas denominadas funciones hash se le asigna a cada elemento una posición de manera constante sin importar el tamaño de la tabla. Sin embargo, es posible que dos elementos distintos luego de pasar por la función hash sean asignados la misma clave ya ocupada, a esto se le denomina colisión

La recolocación cuadrática resulta luego de que se realice una colisión, se busca en la posición resultante de sumar el número de colisión elevado al cuadrado de la posición original. De esta forma se evitan formar bloques de posiciones llenas consecutivas.

---

9. Ramificación y poda. Estrategia de minimización de costes.

Siendo ramificación y poda un método que permite determinar la solución de un problema de asignación eliminando del árbol de posibilidades las ramas que no son solución y que vayan a dar una solución descartable añadiendo al algoritmo un coste de asignación a cada nodo del árbol un beneficio estimado, para la optimización de minimización de costes se realiza lo siguiente:

1. Establecer los valores iniciales:

LNV: lista de nodos vivos y se inserta solo la raíz
C: cota superior de la raíz
S: el conjunto solución comienza vacío
La condición de poda será que la cota inferior de la raíz sea mayor que C

2. Método a seguir:

Seleccionar de la lista de nodos vivos el nodo más prometedor con el menor coste y en caso de empate habría que establecer si se usa una estrategia LIFO o FIFO.

El nodo de LNV: en caso de que la cota inferior del nodo sea superior que la variable de poda: se poda el nodo, y en caso contrario:

- Para cada hijo posible de X, en caso de que sea solución y esa solución sea mejor que la que ya se tenía que establecer como nueva y se actualiza la variable de poda como el mínimo entre la nueva solución y la variable de poda actual.
- En caso que no sea solución y la cota inferior sea menor que la variable de poda, es un nodo prometedor por lo que se añade a la lista de nodos vivos. Y se actualiza la variable de poda como el mínimo de la cota superior del nodo y la variable de poda.

Se repite el bucle hasta que la LNV esté vacía.

---
---
---

# EXAMEN AED 2017

1. (1.5) Describe la estructura interna de la implementación de los nodos de un árbol binario de búsqueda ordinario, un árbol AVL, un árbol B y un árbol B+ ¿Cuál es la principal ventaja de un árbol AVL frente a un árbol binario de búsqueda ordinario? ¿Y de un árbol B + frente a un árbol AVL?

ABB y AVL

Un árbol binario de búsqueda ordinario se caracteriza por tener una estructura donde todos los elementos del subárbol de la izquierda de un nodo son menores que este, y todos los del subárbol de la derecha son mayores.

Un árbol AVL de igual forma es un árbol binario de búsqueda, sin embargo se diferencia en que tiene condiciones a cumplir que permiten mantener el balance del árbol. A cada nodo se le añade un dato denominado factor de equilibrio que almacena la diferencia entre la altura del subárbol derecho y la del izquierdo. De esta forma, cuando este valor es menor de 2 para cada nodo, el árbol está equilibrado, en caso contrario se debe realizar una estrategia de reestructuración para nuevamente equilibrar dicho árbol.

B y B+

Por otro lado, los árboles B son árboles siempre equilibrados donde todas las hojas se encuentran en el mismo nivel. Los nodos se agrupan en páginas a las que se accede en bloques y cada una debe cumplir diferentes condiciones a partir de un orden m establecido para el árbol.

Los árboles B+ son optimizaciones de los árboles B donde todos los datos se encuentran en las páginas hoja y las páginas superiores contienen datos duplicados que sirven de índice, y tienen la ventaja de que en caso de borrado los datos que estaban previamente duplicados no requieren ser eliminados ya que siguen cumpliendo la función de índices. Para árboles que tengan muchas modificaciones ahorra el coste de reestructura.

Ventaja AVL frente ABB
El mantener el equilibrio de un árbol permite aprovechar la organización de este y se presenta como la principal ventaja de los árboles AVL frente a los ABB ordinarios.

Ventaja B+ frente AVL
Para la gestión de grandes cantidades de datos los árboles B+ presentan una clara ventaja sobre los árboles AVL, donde el uso de nodos en AVL sería inviable en memoria principal y

el uso de la memoria externa tardaría mucho tiempo en encontrar los datos. Los árboles B+ reducen significativamente la altura del árbol y el agrupar los nodos en páginas permite traer varios nodos de golpe reduciendo el número de accesos necesarios además de que todos los datos se encuentran en las hojas

.----------

2. 2 (1) Explica cómo se ordenan los elementos en un montículo binario. ¿Para qué es útil una estructura de este tipo? Pon un ejemplo de aplicación

Los montículos binarios son árboles binarios semicompletos donde se organizan los datos a partir de sus prioridades. Pueden ser de mínimos (elementos menores con mayor prioridad) o de máximos (elementos mayores de mayor prioridad). Es útil para cumplir funciones de una cola de prioridad.
Dentro de sus aplicaciones se puede encontrar la planificación de procesos en un sistema multiusuario, gestión de enfermos en servicio de urgencias o planificación de futuras operaciones. Situaciones donde se desea saber quién va siguiente como en aeropuertos, aparcamientos, colas de impresión, etc.

.----------

3. (1.5) Observa el siguiente grafo de conexiones entre todas la islas de la provincia de Tenerife. Para cada conexión (que puede ser de dirección única), además de saber la isla de origen y la isla destino, se conoce el coste del viaje.

    a. La estructura interna de un grafo puede representarse de dos formas, mediante estructuras estáticas o mediante estructuras dinámicas. Indica en este caso cuál de las dos usarías y explica por qué.

Matriz de adyacencia (estático).
Primero, se sabe que los nodos no se van a eliminar ya que son elementos fijos: islas de una comunidad autónoma de un país, ni se van a añadir más.
Segundo, son pocas islas por lo que no sería un uso de espacio demasiado desaprovechado en el caso de haber pocos arcos entre ellas.

    b. Para cada par de islas se desea saber cuál es el camino de coste mínimo que las une (siempre que exista alguno). Indica qué algoritmo usarías.

Para saber el camino de coste mínimo podría usar el algoritmo de Floyd o Dijkstra.
Escogiendo el algoritmo de Floyd, se sabe que este consiste en dos matrices: una matriz de distancias que indica la menor entre cualquier par de vértices y una matriz de vértices previos que indica cuales son los vértices a recorrer para obtener la ruta de menor coste.

    c. En caso de que todos los arcos fueran bidireccionales, indica qué buscarías para encontrar los puntos de la red que, si fallan, producen un fallo general de la red de comunicaciones.

Con la implementación de un grafo, se puede realizar un algoritmo que busca los puntos de articulación del mismo. Lo que diría en qué islas un fallo (si el nodo desaparece) produciría un fallo general de la red dividiendo la conexión.

d. En caso de que todos los arcos fueran bidireccionales, ¿cómo podríamos saber si podemos eliminar alguna de las conexiones minimizando el coste? ¿Qué algoritmo aplicarías?

Se podría usar el algoritmo de Kruskal, donde calcula el árbol de expansión mínima a partir de un grafo con los mismos nodos que el inicial que representa las islas, sin arcos y va añadiendo aquellos que sean mínimos en cada iteración que unan componentes conexas distintas para finalizar en un grafo conexo.

.----------

4. (2) Decide las estructuras de datos más apropiadas (pilas, colas, listas, árboles, grafos) para simular los procesos que sigue un buscador de Internet para recopilar e indexar información de la web, y organizar la información para poder calcular el grado de autoridad de los sitios web.

   Los buscadores de Internet, tipo Google, utilizan unos programas llamados robots web que se encargan de explorar la web recopilando datos de cada sitio web y navegando por los enlaces (links) que encuentran en cada página web. Cada cierto tiempo, los robots envían la información (contenidos de páginas web y enlaces con otras páginas) a un repositorio de datos que se encarga de organizar e indexar de forma óptima dicha información. Posteriormente, hay que aplicar un algoritmo de ránking que permita calcular el grado de importancia de cada sitio web. Estos algoritmos se basan en el concepto de autoridad, cada sitio web tiene el valor de autoridad que determina el número de sitios web que interactúan con él.

   En concreto, define y justifica la elección de las estructuras de datos más empleadas para representar:

   a) La información de los sitios web encontrados por los robots. Esta información tiene que guardarse temporalmente en memoria de la forma más eficiente teniendo en cuenta que los robots están enviando continuamente nuevas páginas que se van procesando para organizarlas e indexarlas.

Teniendo en cuenta que la cantidad de datos es muy elevada para reducir los accesos a memoria, una implementación de árboles B+ permitiría una búsqueda de elementos rápida con pocos accesos. Además, el contenido de las páginas se almacenaría solo en los nodos hoja y en las páginas del interior se encuentran claves de ordenación que brindan la posibilidad de accesos más rápidos. Para finalizar, las constantes modificaciones por parte del robot permiten aprovechar la característica de los árboles B+ de retrasar las reestructuraciones lo máximo posible.

b) La indexación de las páginas web. Hay que indexar los sitios web por orden alfabético de tal forma que la búsqueda de los mismos sea lo más rápida posible. Hay que seleccionar una estructura de datos para crear este indexador de tal forma que se permita la instrucción de sitios web nuevos conservando el orden alfabético y, si ya existen, actualizar el número de visitas que nuestro robot haya realizado.

Se pueden usar árboles AVL, que mantienen un orden alfabético y además permite aprovechar la estructura de árbol al mantenerse siempre en equilibrio.

c) Los enlaces entre los sitios web. Los robots almacenan los enlaces que salen de cada sitio web en un fichero. Decide cuál es la estructura más apropiada para guardar esta información de forma que permita poder calcular la autoridad de cada sitio web de forma sencilla.

Sería ideal utilizar un grafo dirigido, donde se pueden calcular el grado saliente y el entrante de cada vértice.

.----------

5. Explica las ventajas y desventajas en cuanto a eficiencia (tiempo, espacio de almacenamiento) en las operaciones de inserción, búsqueda y ordenación cuando se usa una tabla hash, un vector o una lista enlazada para almacenar un número dado de datos N. Indica qué diferencia habría entre usar una tabla hash con recolocación y una tabla hash con encadenamiento.

Los vectores son estructuras estáticas de posiciones contiguas de memoria que necesitan tener el tamaño a implementar en el momento de su creación y hay que conocer los datos totales a priori. Presentan la ventaja de que el acceso a los datos sabiendo su posición es directo; sin embargo, para la búsqueda de un elemento sin saberla puede recorrer la cantidad total de elementos en el peor de los casos.

La lista enlazada es una estructura dinámica que crea posiciones a medida que se insertan datos lo que tiene la ventaja de que no requiere saber la cantidad de datos a priori. Necesitan almacenamiento aparte del dato, para punteros a los datos vecinos. Por otra parte, como desventaja a la hora de buscar un dato es necesario recorrer la lista entera elemento a elemento en el peor de los casos.

La tabla hash es un vector con tamaño fijo que permite ubicar elementos a partir de una clave. Sin embargo, puede haber coincidencias de estas y dependiendo de su implementación la memoria se ve afectada.

Si se usa con recolocación solo se encuentra el vector con tamaño fijo y para la búsqueda, esta puede ser inmediata o en el peor de los casos si se usa una recolocación lineal por ejemplo, habría que verificar todas las posiciones de la tabla derivando en un coste lineal.

A diferencia de las tabla hash con implementación de recolocación, se encuentra la de encadenamiento que tiene listas enlazadas en cada posición para que al coincidir dos claves, estos datos se añadan a la lista de dicha posición. Así, ahora el tamaño no es fijo y depende de la cantidad de datos almacenados. El coste en la búsqueda sería igual que el de una lista enlazada.

.----------r----------

6. (1) Dado el esquema general de un algoritmo voraz.

Indica qué representan las siguientes variables o funciones:

C: representa el conjunto de elementos candidatos pendientes de seleccionar. En un inicio C es el conjunto de todos los elementos seleccionables.
S: representa el conjunto de candidatos de C seleccionado para ser solución.
x: es el candidato del conjunto C estudiado para ser insertado en un momento dado.

Solución(S): verifica que todo el grafo sea una sola componente conexa
Seleccionar(C): selecciona el arco con coste mínimo de los candidatos.
Factible(S,x): verifica si el candidato seleccionado sería el indicado en la iteración, si cumple la condición de que una componentes conexas distintas.
Insertar(S,x): añade un elemento x al conjunto S de candidatos seleccionados para la solución.
Objetivo(S): dada la solución S, devuelve el coste mínimo asociado al árbol.

.----------

7. (1.5) Explica el proceso de exploración del espacio de soluciones en la estrategia de ramificación y poda en un problema de optimización en el que el método utilizado para definir la solución óptima es la minimización del coste.

Siendo ramificación y poda un método que permite determinar la solución de un problema de asignación eliminando del árbol de posibilidades las ramas que no son solución y que vayan a dar una solución descartable añadiendo al algoritmo un coste de asignación a cada nodo del árbol un beneficio estimado, para la optimización de minimización de costes se realiza lo siguiente:

1. Establecer los valores iniciales:

LNV: lista de nodos vivos y se inserta solo la raíz
C: cota superior de la raíz
S: el conjunto solución comienza vacío
La condición de poda será que la cota inferior de la raíz sea mayor que C

2. Método a seguir:

Seleccionar de la lista de nodos vivos el nodo más prometedor con el menor coste y en caso de empate habría que establecer si se usa una estrategia LIFO o FIFO.

El nodo de LNV: en caso de que la cota inferior del nodo sea superior que la variable de poda: se poda el nodo, y en caso contrario:

- Para cada hijo posible de X, en caso de que sea solución y esa solución sea mejor que la que ya se tenía que establecer como nueva y se actualiza la variable de poda como el mínimo entre la nueva solución y la variable de poda actual.
- En caso que no sea solución y la cota inferior sea menor que la variable de poda, es un nodo prometedor por lo que se añade a la lista de nodos vivos. Y se actualiza la variable de poda como el mínimo de la cota superior del nodo y la variable de poda.

Se repite el bucle hasta que la LNV esté vacía.

---
---
---

# EXAMEN AED 2017

1. Algoritmo de Dijkstra

Calcula el camino de longitud mínima entre un vértice origen y todos los demás vértices del grafo.

Es un algoritmo voraz que va seleccionando la mejor opción en cada iteración.

C: Conjunto de vértices candidatos (todavía sin estudiar).
• S: Conjunto de vértices ya escogidos.
• Camino especial: Camino que sale del vértice origen (a partir del cual se estudian los caminos más cortos) y que pasa por todos los vértices de S excepto probablemente el último.
• D: Vector de distancias, que mantiene en cada uno de sus campos la longitud del camino especial m´as corto entre el v´ertice origen y el v´ertice del grafo al que representa la posici´on
del campo del vector.
• A: Matriz de pesos, necesaria para poder calcular los caminos.
• P: Vector auxiliar que contiene en cada uno de sus campos el predecesor inmediato del camino especial mínimo para el vértice que representa la posición del campo en el vector.
Para realizar el algoritmo, se parte de un v´ertice origen, y a partir de ´el se establece en D el peso de los arcos con sus vértices adyacentes, poniendo en P (en el campo que corresponda a estos vértices) el número del vértice inicial. Se marca el v´ertice original como ya estudiado (se a˜nade a S), y se selecciona el v´ertice con el que se estableci´o el camino m´as corto de todos los caminos establecidos hasta este punto. Se repiten estos pasos sobreescribiendo solo el valor de aquellos caminos que tuvieran una longitud mayor a la obtenida por medio del v´ertice de
estudio actual.

Como se ve, tiene la desventaja de que si queremos saber el camino m´ınimo entre cualquier par de v´ertices, tenemos que realizar el algoritmo partiendo de o todos los v´ertices del grafo y luego comparar.

2. Escribir el código de un recorrido inorden

```
//Recorrido recursivo inorden: I-R-D
void inorden(abin A) {
   tipoelem E;
   if (!esVacio(A)) {
      inorden(izq(A));
      leer(A, &E);
      printf("%d\t", E);
      inorden(der(A));
   }
}

//Recorrido recursivo preorden: R-I-D
void preorden(abin A) {
   tipoelem E;
   if (!esVacio(A)) {
      leer(A, &E);
      printf("%d\t", E);
      preorden(izq(A));
      preorden(der(A));
   }
}

//Recorrido recursivo postorden: I-D-R
void postorden(abin A) {
   tipoelem E;
   if (!esVacio(A)) {
      postorden(izq(A));
      postorden(der(A));
      leer(A, &E);
      printf("%d\t", E);
   }
}

//Recorrido no recursivo inorden: utiliza pila como estructura auxiliar
void inordenNR(abin A) {
   tipoelem E;
   abin aux;
   pila p;
   aux = A;
   crearPila(&p);
   do {
```

```
        while (!esVacio(aux)) {
            //se meten en la pila todos los punteros izquierdos
            leer(aux, &E);
            push(&p, aux);
            aux = izq(aux);
        }
        if (!esVaciaPila(p)) {
            aux = tope(p);
            pop(&p);
            leer(aux, &E);
            printf("%d\t", E);
            aux = der(aux); //se sigue por la derecha del tope de la pila
        }
    } while (!esVaciaPila(p) || !esVacio(aux));
    destruirPila(&p);
}

//Recorrido no recursivo en anchura: utiliza cola como estructura auxiliar
void anchura(abin A) {
    tipoelem E;
    abin aux;
    cola c;
    crearCola(&c);
    if (!esVacio(A)) //Insertamos nodo raiz, primer nivel
        insertarCola(&c, A);
    while (!esVaciaCola(c)) {
        aux = primero(c); //Se extrae un elemento de la cola
        suprimirCola(&c);
        leer(aux, &E);
        printf("%d\t", E);
        if (!esVacio(izq(aux)))
            insertarCola(&c, izq(aux));
        if (!esVacio(der(aux)))
            insertarCola(&c, der(aux));
    }
    destruirCola(&c);
}
```

a.