Estructuras de Datos y Algoritmos - Curso 2003/04

Ejercicios del Tema III

Ejercicio 1.

Especificación:

Se pretende informatizar la gestión de una biblioteca. La información que se tiene de cada ejemplar de un libro es: autor, título, editorial y año de publicación.

Especificar algebraicamente el TAD libro con una operación generadora y cuatro operaciones para observar la información del libro. Pueden suponerse ya especificados los TAD's cadena de caracteres y entero.

Especificar algebraicamente el TAD biblioteca, con las siguientes operaciones: operación de inicialización (biblioteca nueva, sin libros), operación de compra de un ejemplar de un libro (añadir un ejemplar), operación de supresión (para cuando se pierde un ejemplar) y operación que dice si un cierto título está o no en la biblioteca.

Suponiendo ya especificado el TAD genérico pila, enriquecer la especificación del TAD biblioteca con una operación que proporciona la pila de ejemplares de la biblioteca escritos por un autor y con otra operación que proporciona la pila de ejemplares de la biblioteca que tratan sobre un cierto tema (se considera que un libro trata sobre un tema si y sólo si ese tema es una subcadena no vacía del título del libro; puede suponerse que el TAD cadena incluye la operación que detecta si una cadena es subcadena de otra).

Nota: en una biblioteca puede haber varios ejemplares de un mismo libro.

Representación e implementación:

Proponer una representación estática para los TAD's cadena y libro.

Proponer una representación dinámica para el TAD biblioteca que permita implementar la operación que dice si un cierto título está o no en la biblioteca con un coste en O(log N) en el peor de los casos, con N el número de libros distintos en la biblioteca (notar que el número de libros distintos puede ser menor que el número de ejemplares de la biblioteca).

Dada la representación anterior, diseñar un algoritmo no recursivo que escriba en pantalla todos los libros de la biblioteca que tratan sobre un cierto tema.

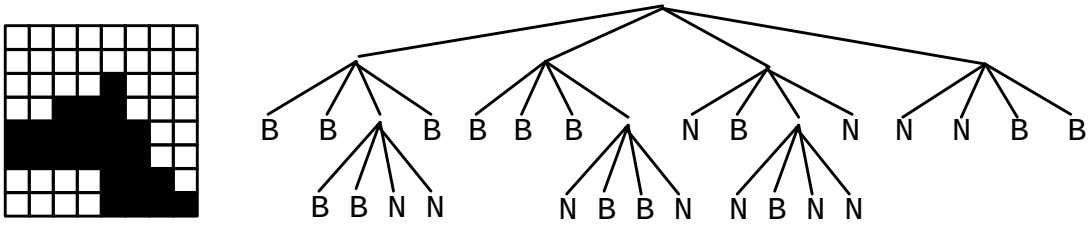
Para la misma representación, diseñar un algoritmo recursivo lo más eficiente posible que escriba en pantalla todos los libros de la biblioteca con título `t` tal que `c1`$\leq$`t`$\leq$`c2`, donde `c1` y `c2` son dos cadenas dadas y '$\leq$' representa el orden alfabético (puede utilizarse el operador '$\leq$' para comparar dos cadenas).

Ejercicio 2.

La estructura de datos *quad-tree* se usa en informática gráfica para representar figuras planas en blanco y negro. Se trata de un árbol en el cual cada nodo, o bien tiene exactamente cuatro hijos, o bien es una hoja. En este último caso, puede ser una hoja blanca o una hoja negra.

El árbol asociado a una figura dibujada dentro de un plano (que, para simplificar, podemos suponer un cuadrado de lado 2^k) se construye de la forma siguiente: se subdivide el plano en cuatro cuadrantes; los cuadrantes que estén completamente dentro de la figura corresponderán a hojas negras, los que estén completamente fuera de la región, a hojas blancas y los que estén parcialmente dentro y parcialmente fuera, a nodos internos; para estos últimos se aplica recursivamente el mismo

algoritmo. Como ejemplo, se muestra una figura en blanco y negro y su árbol asociado (considerando los cuadrantes en el sentido de las agujas del reloj, a partir del cuadrante superior izquierdo):



Escribid una representación para los *quad-trees* e implementad las siguientes operaciones:

```
algoritmo creaQuadTree(sal qt:quadtree; ent f:figura)
{ Construye en qt el quad-tree asociado a la figura f. }

algoritmo creaFigura(sal f:figura; ent qt:quadtree)
{ Reconstruye en f la figura representada en el quad-tree qt. }
```

Donde el tipo figura es el siguiente:

```
constante N = ... {N=2^k, para alguna constante k>0}
tipo figura = vector[1..N, 1..N] de booleano {verdad representa negro}
```

Ejercicio 3.

Objetivo general:

Resolver el problema de recuento de palabras en un texto utilizando un árbol AVL.

Un árbol AVL es un árbol binario de búsqueda (cada nodo, excepto las hojas, es mayor o igual que todos los elementos de su subárbol izquierdo y menor que todos los elementos de su subárbol derecho) y equilibrado (para cada nodo las alturas de sus dos subárboles difieren como mucho en 1) .

Descripción:

Se debe diseñar un programa en Ada que lea una secuencia de palabras y determine el número de veces que aparece cada una de ellas en la secuencia. Para ello, se dispondrá de un árbol AVL inicialmente vacío. Cada palabra leída se buscará en el árbol; si se encuentra, se incrementará su contador; si no, se insertará en el árbol como nueva palabra (con el contador inicializado a 1). Una vez leído el texto, se deberán escribir en pantalla, por orden alfabético, las palabras leídas junto con el número de apariciones de cada una de ellas en la secuencia.

Tareas:

1. Implementar en Ada la operación de búsqueda con inserción que consiste en: dados una palabra y un árbol AVL de elementos del tipo “registro palabra-contador”, busca la palabra en el árbol; si se encuentra, se incrementa su contador; si no, se inserta en el árbol como nueva palabra (con el contador inicializado a 1), manteniendo el árbol en la clase AVL.
2. Implementar en Ada la operación de recorrido en in-orden que consiste en: dado un árbol AVL de elementos del tipo “registro palabra-contador”, escribe en pantalla los elementos del árbol (palabra y contador), obtenidos recorriendo el mismo en orden simétrico (in-orden).

3. Escribir un programa en Ada que lea una secuencia de palabras, las guarde en un árbol AVL de elementos del tipo “registro palabra-contador” y escriba en pantalla las palabras leidas, por orden alfabético, y el número de apariciones de cada una de ellas en la secuencia.

Ejercicio 4.

Objetivo:

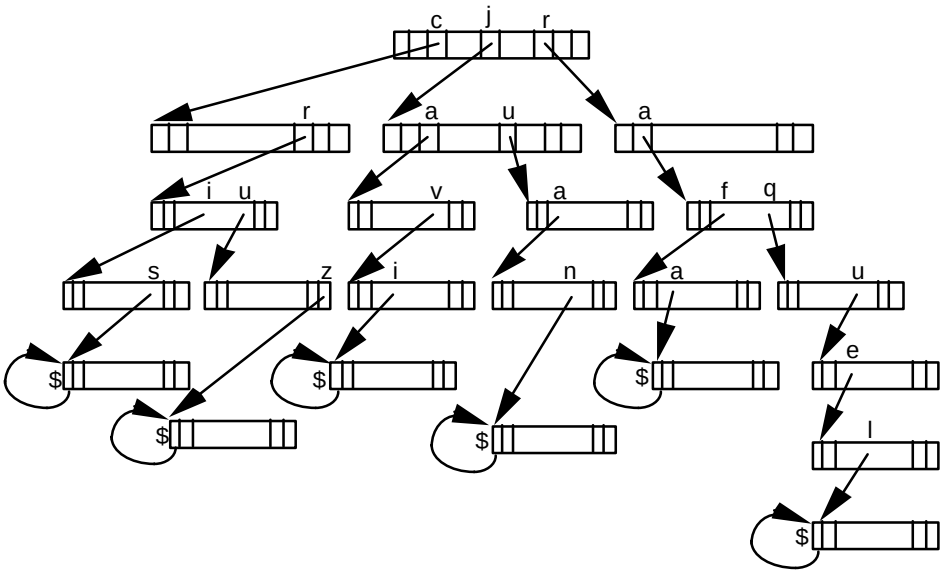
Implementación de un paquete que exporte el tipo *trie* y realización de un programa que utilice el tipo.

Descripción detallada:

Se propone realizar una implementación dinámica de tries en la que los nodos del trie sean del siguiente tipo:

```
type cars is ('$','a',...,'z');
type NODO_TRIE;
type TRIE is access NODO_TRIE;
type NODO_TRIE is array(cars) of TRIE;
```

De esta manera una palabra queda almacenada en el trie por un camino de nodos del árbol en el que los índices de los punteros de los nodos del camino indican las letras de la palabra. El $ se utiliza para indicar que el camino recorrido hasta ese nodo es una palabra completa. Por ejemplo, un trie que contenga las palabras {cruz, rafa, javi, juan, raquel, cris} se almacenaría de la siguiente forma:



Para indicar que un camino tiene una palabra completa se hará apuntar el puntero de índice $ al mismo nodo, y en caso contrario tendrá el valor null.

Se pide implementar un módulo en ADA que exporte el tipo trie con las operaciones: crear_vacío, inserción, búsqueda y borrado de una palabra. Además realizar un programa que lea de un fichero de texto 'dicciona.rio' el conjunto de palabras válidas de un lenguaje (formato del fichero: una palabra por línea, en minúsculas), las almacene en un trie, y utilice el diccionario para leer un fichero de texto 'datos.txt' e indicar en otro fichero de texto 'resultad.txt' las palabras de 'datos.txt' que no pertenezcan al diccionario y la línea en la que se encuentran.

Ejercicio 5.

Objetivo:

Enriquecer el tipo trie del ejercicio anterior para el desarrollo de un programa.

Descripción detallada:

Debe desarrollarse un programa dirigido por menús para facilitar la gestión de un diccionario. El diccionario estará almacenado en un fichero de texto 'dicciona.rio' con una palabra por línea (en minúsculas). El programa debe cargar al comienzo de la ejecución el diccionario en memoria principal y permitir la inserción, búsqueda y borrado de palabras, así como el listado de todo el diccionario. Cuando el usuario no quiera realizar más modificaciones en el diccionario, antes de finalizar la ejecución se debe actualizar el fichero 'dicciona.rio'.

Para realizar el listado del diccionario y la actualización del fichero donde se almacena es necesario enriquecer el módulo desarrollado en el ejercicio anterior con una operación de listado del trie completo.

Se pide incorporar al módulo de implementación del tipo trie del ejercicio anterior la operación "listado" que recorra el trie en preorden y dé como resultado el listado completo de las palabras contenidas en el trie por orden alfabético. Posteriormente desarrollar el programa de gestión de diccionarios.

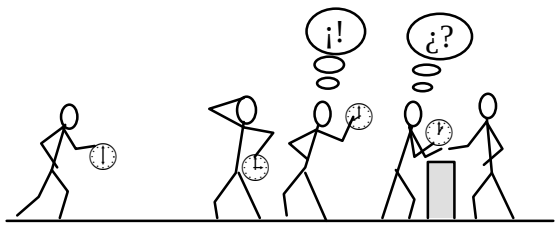
Ejercicio 6.

Una *Cola con Prioridades y Servidor Imprevisible*[1] (CPSI) es un sistema de espera al que llegan elementos con una cierta prioridad asociada para recibir servicio. Debido al extraño carácter del servidor, no siempre es servido en primer lugar el elemento de la cola que tiene mayor prioridad (como ocurre en una cola con prioridades ordinaria). Dependiendo del estado de su ánimo, el servidor decide servir en cada momento bien al elemento con mayor prioridad (si el servidor está contento), bien al que tiene menor prioridad (si está enfadado).

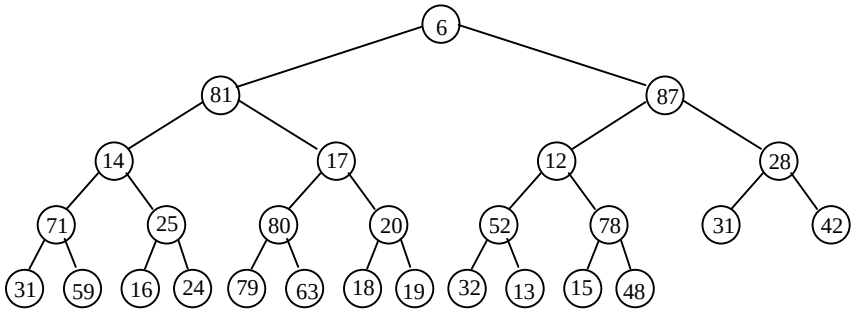
a) Especificar algebraicamente el tipo genérico CPSI con las operaciones de: añadir un elemento, observar el primero de la cola (que es el de mayor prioridad si el servidor está contento o el de menor en caso contrario), eliminar el primero de la cola (con iguales reglas) y observar si la cola está vacía o no. Para poder realizar la especificación, definir además un género que identifique el estado de ánimo del servidor.

b) Diseñar la interfaz de un módulo que implemente el TAD genérico CPSI. Proponer una representación estática de los datos de tipo CPSI que permita implementar todas las operaciones con un coste del orden del logaritmo del número de elementos en la cola, en el peor de los casos. Para ello, obsérvese que una CPSI puede representarse como un *montículo min-max*. Un montículo min-max tiene una estructura similar a la de un montículo ordinario (árbol binario parcialmente ordenado casi-completo), pero la ordenación parcial consiste en que los elementos que se encuentran en un

[1] CPSI:



nivel par (0, 2, 4, …) son menores o iguales que sus elementos descendientes, mientras que los elementos que se encuentran en nivel impar son mayores o iguales que sus descendientes (véase la figura).



c) Implementar la operación de añadir un elemento, con un coste en el peor caso del orden del logaritmo del número de elementos en la cola.