Técnicas Avanzadas de Programación:
Práctica 1ª:
AVL versus Rojinegros

Acedo Legarre, Aitor
Faci Miguel, Santiago

Agosto de 2004

Powered by LaTeX.

Contents

1 Introducción al problema

El problema que se plantea consiste en comparar la eficiencia práctica de dos implementaciones: una de árbol AVL y otra de árbol rojinegro para el TAD diccionario con las operaciones citadas en el enunciado. Para ello se deberán buscar o desarrollar implementanciones para ambas estructuras de datos y realizar una batería de tests con el fin de obtener resultados comparativos presentando éstos en un fichero.

2 Solución propuesta

Para la estructura de árbol AVL se ha escogido la implementación del profesor de la asignatura, Javier Campos, disponible desde su página web ya que proporciona toda la funcionalidad requerida para la práctica y adaptada al TAD diccionario.

Para la estructura de árbol Rojinegro se ha seguido la implementación en pseudocódigo que se proporciona en el libro *Introduction to Algorithms* de Cormen, Leiserson y Rivest.

2.1 Estructuras de datos utilizadas

A continuación, se especifican los detalles de las estructuras de datos utilizados para la solución propuesta.

2.1.1 Árbol AVL

- La estructura del árbol.

```
type avl is access nodo;
```

- La estructura para cada nodo del árbol.

```
type nodo is
   record
      -- Almacena la palabra
      clave:tp_clave;
      -- Almacena la definición de la palabra
      valor:tp_valor;
      -- Indica el equilibrado del nodo
      equilibrio:factor_equilibrio;
      -- Hijos izquierdo y derecho del nodo
```

```
      izq,dch:avl;
   end record;
```

- Procedimientos implementados. Están implementados los procedimientos de *vacio*, *buscar*, *modificar* y *borrar*. El procedimiento *vacio* crea un árbol AVL vacío, *buscar* busca una palabra en algún nodo del árbol, *modificar* inserta una palabra y su definición en el árbol como un nuevo nodo, *borrar* elimina una palabra y su definición del árbol. Los procedimientos que se detallan aqui son los requisitos en cuanto a cumplir las restricciones para implementar un TAD diccionario, pero es necesario una serie de procedimientos auxiliares para mantener equilibrado el árbol que también se encuentran implementados en el código.

```
procedure vacio(a:out avl);
-- devuelve un diccionario vacío
procedure modificar(a:in out avl; clave:in tp_clave;
   valor:in tp_valor);
-- inserta una nueva "clave" con su "valor" en el
-- diccionario;
-- si la "clave" ya estaba, actualiza su "valor";
procedure borrar(a:in out avl; clave:in tp_clave);
-- borra la "clave" (y su "valor") del diccionario;
-- si la "clave" no estaba en el diccionario,
-- lo deja igual
procedure buscar(a:in avl; clave:in tp_clave;
   exito:out boolean; valor:out tp_valor);
-- si la "clave" está en el diccionario devuelve su
-- "valor" (y "exito" es true);
-- en caso contrario, "exito" toma el valor false
```

2.1.2 Árbol Rojinegro

- La estructura del árbol. Se trata de un puntero a un nodo. Ese nodo hará de nodo raíz por lo tanto su nodo padre siempre sera igual a *null*. A partir de ahi se irán expandiendo los nodos a través de los punteros "hijos" de cada nodo.

```
type tree is access nodo;
```

- La estructura para cada nodo del árbol. Además de almacenar la palabra y su definición (TAD diccionario) se guardan punteros a los hijos izquierdo y derecho del nodo y al nodo padre (necesario para realizar la rotación). También tendremos un campo que nos indicará el color del nodo: rojo — negro.

```
type nodo is
   record
      -- Palabra del nodo
      palabra: tipo_palabra;
      -- Definición
      definicion: tipo_definicion;
      -- Nodos hijo izquierda, derecha y nodo padre
      izq, dch, padre: tree;
      -- Campo que indica el color del nodo
      color: colores;
   end record;
```

- Procedimientos implementados. Se han implementado los procedimientos *vacio*, *buscar*, *insertar*, *borrar*, *rotacion_dcha*, *rotacion_izq* y *dibujar*. El procedimiento *crear* crea un

árbol rojinegro vacío, *buscar* busca una palabra determinada en todo el árbol, *insertar* inserta un nuevo nodo con palabra y definición, si la palabra ya existía se actualiza su definición; *borrar* elimina una palabra que exista en el árbol junto con su definición, *rotacion_dch / rotacion_izq* rota a derechas/izquierdas el árbol rojinegro siempre que sea necesario (tras algunas inserciones o borrados) y *dibujar* realiza un boceto de la estructura del árbol y su contenido.

```
-- Crea un árbol Rojinegro vacío
procedure vacio (t: out tree);
-- Inserta una palabra con su definición en el árbol,
-- si esta palabra ya existe se actualiza su definición
procedure insertar (t: in out tree; palabra: in tipo_palabra;
   definicion: in tipo_definicion);
-- Borra el nodo que almacena la palabra que se pasa como
-- parámetro
procedure borrar (t: in out tree; palabra: in tipo_palabra);
-- Busca el nodo que contiene a la palabra. Además, dicho nodo
-- es apuntado por el parámetro de salida "el_nodo"
procedure buscar (t: in tree; palabra: in tipo_palabra;
   encontrada: out boolean; el_nodo: out tree);
-- Rota a izquierdas el árbol Rojinegro
procedure rotacion_dch (t: in out tree; x: in out tree);
-- Rota a derechas el árbol Rojinegro
procedure rotacion_izq (t: in out tree; x: in out tree);
 -- Realiza un boceto de la estructura del árbol y su contenido
procedure dibujar (t: in tree);
```

2.2 Paquete ustrings

Para el manejo de las cadenas de texto de tipo *ustring* se ha utilizado en ambas implementaciones el paquete *ustrings*, que se encontraba disponible desde la página de la asignatura.

3 Programas de prueba

Para la obtención de resultados se ha creado un pequeño programa con un procedimiento (uno para cada implementación) que se encarga de realizar diversas operaciones de inserción y búsqueda sobre las estructuras de datos descritas anteriormente, y obtiene datos de eficiencia práctica de las mismas.

Para medir los rendimientos de ambas implementaciones se han probado estructuras de diferentes tamaños (número de elementos) y se han medido los resultados. Se han realizado diferentes número de búsquedas sobre tamaños de estructuras de 10, 100, 1000 y 10000 elementos para ambas implementaciones; de esta manera se pueden comparar para diferentes tamaños y a su vez observar la escalabilidad de los mismos, es decir, como actúa ante el crecimiento del número de elementos una misma implementación de cualquiera de las estructuras de datos estudiadas.

Todos los resultados de esta comparativa se obtienen ejecutando el programa *test_rbtree* / *test_avl*, según se quiera probar una u otra implementación. Dichos resultados se escriben en el fichero *resultados_practica1.txt* [1].

El fichero de resultados tiene el siguiente formato:

```
TEST ROJINEGRO | Tamaño:   100, Búsquedas:  100, Borrados:  100
Inserción | Tiempo:       1093
```

[1] Los tiempos se indican en microsegundos

```
Búsqueda | Tiempo:        638
Borrado | Tiempo:        537
```

4 Bibliografía

- Apuntes de la asignatura de Técnicas Avanzadas de Programación
- Introduction to Algorithms, Thomas H. Cormen, Charles E. Leiserson, Ronals L. Rivest