Técnicas Avanzadas de Programación: Práctica 4ª: Estructuras auto-organizativas

Acedo Legarre, Aitor
Faci Miguel, Santiago

Agosto de 2004

Powered by LaTeX.

Contents

1 Introducción al problema

El problema que se nos plantea en esta práctica es el comparar la eficiencia práctica de una implementación de lista lineal auto-organizativa con otra de árbol auto-organizativo ("splay tree") para el TAD diccionario que se viene desarrollando a lo largo de las anteriores sesiones de prácticas.

Todo esto deberá llevarse a cabo buscando o desarrollando previamente las implementaciones de ambas estructuras de datos en lenguaje Ada con las operaciones de crear, buscar, insertar y borrar, para posteriormente definir una batería de tests para aplicarlos sobre dichas estructuras con el fin de obtener resultados comparativos en un fichero de resultados.

2 Solución propuesta

Como estructuras de datos no habia ninguna posibilidad de elección. Se han utilizado una implementación de lista lineal auto-organizativa y otra de árbol auto-organizativo, ambas en el lenguaje de programación Ada tal y como indicaba el enunciado.

La implementación de lista lineal auto-organizativa ha sido implementada por nosotros mismos siguiendo las indicaciones de los apuntes de la asignatura Técnicas Avanzadas de Programación (apartado 3.6), optando por una heurística basada en "contador de frecuencias" por la que los nodos más accedidos, historicamente, salen beneficiados al situarse siempre al frente de la lista. Se ha escogido esta implementación por tratarse de un compromiso entre los tres tipos de heurísticas de manera que así se obtendrán resultados para un caso intermedio de lista lineal auto-organizativa.

Para la implementación de árbol auto-organizativo ("splay tree") se ha utilizado la implementación que en los mismos apuntes de la asignatura de Técnicas Avanzadas de Programación (apartado 3.7) aparece. Indicar que a la implementación encontrada en los apuntes se añadieron ligeras modificaciones de manera que se adaptará al TAD diccionario que debemos usar en todas las sesiones de prácticas.

Además, se añadió algo de código para proporcionar a la estructura la posibilidad de actualización de las definiciones de las palabras que ya se encuentran insertadas ajustándose así a los requisitos en cuanto a funcionalidad de las implementaciones.

Todas las modificaciones realizadas sobre el código original se encuentran perfectamente comentadas en el propio código. Fue necesario modificar tanto el fichero de especificación *.ads*, para

actualizar la estructura, como el fichero de implementación *.adb* para añadir las nuevas funcionalidades descritas.

Así, con estas implementaciones se implementó un fichero de pruebas a partir de las cuales se obtuvieron los resultados que permiten comparar en eficiencia práctica ambas estructuras.

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 Lista lineal auto-organizativa

- La estructura de la lista. Al tratarse de una lista lineal, se tratará simplemente de un puntero a lo que será el primer nodo de la lista. Más adelante se puede observar como cada nodo posee un campo puntero al siguiente nodo de la lista.

```
-- Puntero al primer nodo
type lista is access nodo;
```

- La estructura para cada nodo de la lista. Para cada nodo de la lista

```
private
   type nodo is
   record
      -- Palabra que representa el nodo
      palabra: tipo_palabra;
      -- Definición del nodo
      definicion: tipo_definicion;
      -- Contador del número de accesos a ese nodo
```

```
      frecuencia: Integer;
      -- Puntero al siguiente nodo de la lista
      siguiente: lista;
   end record;
```

- Procedimientos implementados. Se han implementado los procedimientos de *crear*, *buscar*, *insertar*, *ordenar*, *borrar* y *dibujar*. El procedimiento *crear* crea una lista vacía, *buscar* busca una palabra en la lista lineal, *insertar* inserta una palabra y su definición en la lista como un nuevo nodo, *ordenar* se encarga de ordenador por frecuencias la lista cada vez que un nodo es accedido (heurística por frecuencias), *borrar* elimina una palabra y su definición de la lista, y *dibujar* realiza un boceto de la estructura de la lista y el contenido de ésta.

```
-- Crea una lista vacía
procedure crear(l: out lista);
-- Buscar una determinada palabra en toda la lista.
-- Si la encuentra, incrementará en uno el número de accesos
-- (frecuencia) a esa palabra
procedure buscar(palabra: in tipo_palabra; l: in out lista);
-- Inserta una palabra en la lista.
-- Si ésta ya existe actualiza su definición
procedure insertar(palabra: in tipo_palabra;
   definicion: in tipo_definicion; l: in out lista);
-- Ordena la lista. Tras cada inserción, se invoca
-- a este procedimiento para que mantenga la lista ordenada
-- por número de accesos (frecuencia) que se dan en cada nodo.
procedure ordena(node: in out lista; l: in out lista);
```

```
-- Elimina, si existe, la palabra y su definición de la lista
procedure borrar(palabra: in tipo_palabra; l: in out lista);
-- Dibuja, a modo de ayuda, la lista auto-organizativa
-- del primero al último elemento.
procedure dibujar(l: in lista);
```

2.1.2 Árbol auto-organizativo

- La estructura del árbol. Se compone de dos nodos. Uno, *raiz* es el nodo raíz del árbol auto-organizativo, el segundo, *nodoNulo* simplemente es un nodo centinela utilizado para simplificar la codificación de esta estructura de datos.

  ```
  type splay is
     record
        -- Nodo raíz del árbol
        raiz: ptNodo;
        -- Nodo centinela
        nodoNulo: ptNodo;
     end record;
  ```

  Donde *ptNodo* es un puntero a un nodo.

  ```
  -- Puntero a un nodo
  type ptNodo is access nodo;
  ```

- La estructura para cada nodo del árbol. Al tratarse de una estructura en árbol, cada nodo posee dos punteros, uno para el hijo izquierdo y otro para el derecho. Además, se almacenan la palabra y definición para cumplir las especificaciones del TAD diccionario.

```
type nodo is
   record
   -- Palabra del nodo
   palabra: tipo_palabra;
   -- Definición de la palabra
   definicion: tipo_definicion;
   -- Nodos hijo izquierda y derecha
   izq, dch: ptNodo;
   end record;
```

- Procedimientos implementados. Se han implementado los procedimientos *crear*, *splay*, *buscar*, *insertar*, *borrar* y *dibujar*. El procedimiento *crear* crea un árbol auto-organizativo vacío, *splay* realiza la reorganización del árbol cada vez que se accede a un nodo existente o se insertar uno nuevo (éste nodo se convierte en el nodo raíz del árbol), *buscar* busca una palabra determinada en todo el árbol (este método desencadena una llamada a splay), *insertar* inserta un nuevo nodo con palabra y definición, si la palabra ya existía se actualiza su definición (este método desencadena también una llamada a splay); *borrar* elimina una palabra que exista en el árbol junto con su definición y *dibujar* realiza un boceto de la estructura del árbol y su contenido.

```
-- Crea un splay tree vacío
procedure crear(s: in out splay);
-- Realiza la reorganización del árbol cada vez que se
-- inserta un nuevo nodo o se accede a uno ya existente
procedure splay(palabra: in tipo_palabra; p: in out ptNodo;
   elNodoNulo: in out ptNodo);
-- Busca una palabra en todo el árbol
```

```
procedure buscar(palabra: in tipo_palabra; s: in out splay;
   encontrado: out boolean);
-- Inserta un nuevo nodo en el árbol, si éste ya existe
-- se actualiza su definición
procedure insertar(palabra: in tipo_palabra;
   definicion: in tipo_definicion; s: in out splay);
-- Borrar la palabra y la definición de ésta
procedure borrar(palabra: in tipo_palabra; s: in out splay);
-- Dibuja un boceto del árbol y el contenido del mismo
procedure dibujar(s: in splay);
```

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_spla_tree / test_lista_auto*, según se quiera probar una u otra implementación. Dichos resultados se escriben en el fichero *resultados_practica4.txt* [1].

El fichero de resultados tiene el siguiente formato:

```
TEST LISTA | Tamaño:   1000, Búsquedas:  1000, Borrados:   100
Inserción | Tiempo:      79574
Búsqueda | Tiempo:      90223
Borrado | Tiempo:      50173
```

Asi, se pueden ver el tamaño de la estructura, el número de búsquedas que se han realizado y posteriormente los tiempos empleados en la inserción de todos los elementos y para realizar todas las búsquedas indicadas (estas búsquedas se realizan sobre elementos de la estructura de forma aleatoria).

En el fichero *resultados_practica4.txt* se presentan ya algunos resultados comparativos tal y como se indica en este apartado. Pero el programa de test está preparado para ser ejecutado e ir anotando otras comparativas añadiendo nuevos resultados al fichero (diferentes tamaños, cantidad de búsquedas diferente).

4 Bibliografía

- Apuntes de la asignatura de Técnicas Avanzadas de Programación
- http://www.eli.sdsu.edu/courses/fall96/cs660/notes/splay/splay.html

[1] Los tiempos se indican en microsegundos

- http://www.eli.sdsu.edu/courses/fall96/cs660/notes/splayPerf/splayPerf.h