Técnicas Avanzadas de Programación: Práctica 3ª: Patricia y un chequeador de palabras

Acedo Legarre, Aitor
Faci Miguel, Santiago

Agosto de 2004

Powered by LaTeX.

Contents

1 Introducción al problema

El objetivo de la práctica era el de proponer una implementación para el tipo de datos *conjunto de palabras* con las operaciones básicas de *insertar*, *borrar* (opcional), *buscar*, utilizando un *Patricia*.

2 Solución propuesta

La primera de las decisiones a tomar a la hora de implementar el conjunto de palabras estará relacionada con las cadenas de caracteres.

Lo primero de todo será definir el conjunto de caracteres que serán válidos para formar parte de nuestras cadenas, en este caso los caracteres válidos serán las letras de la A a la Z, razón por la cual antes de introducir cualquier cadena en el texto deberemos transformarla a mayúsculas.

Otro detalle importante sería el de obligar a que ninguna palabra sea prefijo de otra, esto favorece enormemente a la simplicidad y corrección de los algoritmos, para implementar esto hemos decidido concatenar a todas las cadenas un caracter terminador, el *$* en nuestro caso, con el que evitamos que cualquier cadena sea prefijo de otra perteneciente al conjunto.

Un apunte significativo es que no hemos utilizado ninguna tabla predefinida para transformar los caracteres a ritras de bits, sino que simplemente se han transformado su código ascii, número entero, en una ristra de bits por el método normal de cambio de base.

En cuanto a los algoritmos de búsqueda e inserción se refiere, se puede decir que hemos seguido al pie de la letra las indica-

ciones propuestas por Sedgewick en su libro *Algorithms in C++*, cuyas copias fueron repartidas en clase.

2.1 Estructuras de datos utilizadas

2.1.1 Ristras de bits

Definición de los tipos necesarios para implementar una secuencia de bits:

```
  subtype bit is integer range 0..1;

  type array_bits is array (1..256) of bit;
  -- Palabras de hasta 32 letras

  --Definicion del tipo ristra de bits
  type ristra_bits is record
    ristra: array_bits := (others => 0);
--Inicializamos todas las posiciones del vector a 0
    ind: integer := 0;
--Indice de la secuencia de bits hasta donde
--hay bits significativos
  end record;
```

2.1.2 Patricia

Definición de los tipos necesarios para Patricia:

```
    --Definicion del tipo item
    type tp_item is record
        clave: ristra_bits;
--ristra de bits utilizada en la busqueda y
--la insercion
```

```
        p: ustring; --la palabra almacenada
    end record;

    type nodo_patricia;
    type ptr_nodo_patricia is access nodo_patricia;

    --Definicion del nodo que forma el Patricia
    type nodo_patricia is record
        it: tp_item;
--encapsulamos la clave con su valor por la
--implementacion de Sedgewick
        bit: integer;
--El orden del bit que tenemos que mirar en
--cada comparacion
        i, d: ptr_nodo_patricia;
--enlaces internos y externos
    end record;
```

2.1.3 Conjunto de palabras

El conjunto de palabras será un registro que encapsula la raíz de un Patricia.

```
    type conj is record
        raiz: ptr_nodo_patricia; --Raiz del Patricia
    end record;
```

3 Programa de prueba

El programa con el que vamos a comprobar el correcto funcionamiento de nuestra implementación del conjunto de palabras

no es otro que el *chequear.adb*, gracias a la precisa enumeración de tareas que se debían implementar en dicho programa tenemos una idea clara de como debemos probar el código. Esto hace que el único aspecto de la prueba que puede ser modificable sea el fichero de entrada.

El fichero de entrada que proporcionamos con la práctica 3ª tiene las 10 primeras líneas del fichero *ospd.txt original*, al igual que el fichero *ospd.txt* que aportamos, esto es debido a que nos dimos cuenta de que ocupaba mucho tamaño aportar el fichero *ospd.txt íntegro*, tanto como diccionario como fichero de entrada.

Para verificar el correcto funcionamiento de nuestra implementación del algoritmo buscar e insertar, se han cambiado dos líneas del fichero de entrada, la tercera y la quinta, con nuestros respectivos nombres con el fin de que queden registradas esas dos entradas en el fichero de salida, de este modo comprobaremos que funciona todo correctamente.

La prueba más exigente que hemos aplicado a nuestro código ha sido la de utilizar el propio fichero de diccionario, el *ospd.txt*, como fichero de entrada, provocando así que ninguna entrada quede registrada en el fichero de salida, puesto que todos los elementos del fichero de entrada estaban en el conjunto de palabras. Esta prueba puede ser realizada por el profesor, ya que consideramos que es la prueba más clara de que todo funciona correctamente incluso con ficheros de entrada tan grandes, así que simplemente se tendrán que suprimir el fichero ospd.txt y el entrada.txt que proporcionamos por el ospd.txt original, y simplemente tendremos que volver a ejecutar el chequear.

4 Modo de funcionamiento

Con ánimo de facilitar la tarea de la generación de los ejecutables para probar el funcionamiento de la práctica hemos creído oportuno añadir un fichero *makefile*, que automatiza totalmente este proceso, simplemente tendremos que escribir *make*, sin ninguna opción, en la ubicación donde hayamos descomprimido los fuentes y automáticamente la aplicación se generará, solamente nos quedará ejecutar los ejecutables generados.

Este proceso cuenta además con una serie de opciones que vamos a comentar a continuación:

- *make cheq*: simplemente compilará el fichero chequear que a su vez hará que todos los fuentes se compilen. La diferencia de ejecutar *make* a secas es que con la opción cheq no borraremos los ficheros generados al compilar.
- *make clean*: borrará los ficheros *.o *.ali .
- *make exportar*: modifica la variable de entorno TMPDIR por si hiciera falta en la compilación.

5 Bibliografía

- Hilliam, B.
  Introduction to abstract data types using Ada
  Prentice Hall International Editions

- Barnes, J.
  Programming in Ada 95
  Addison Wesley

- Sedgewick, R.
  Algorithms in C++
  Addison Wesley 1992

- Knuth, D.
  The art of computer programming
  Addison Wesley 1973

- String Representation. Tries. Seminar DatenStrukturen and algorithmen, Bern

- Guía mínima de Ada
  Dirección: http://www.gedlc.ulpgc.es/docencia/mp_/GuiaAda/index.html