Práctica Final: Letras

Dpto. Ciencias de la Computación e Inteligencia Artificial
E.T.S. de Ingenierías Informática y de Telecomunicación
Universidad de Granada

DECSAI

Estructuras de Datos

Doble Grado en Ingeniería Informática y Matemáticas
Grado en Ingeniería Informática

Índice de contenido

1. Introducción

Los objetivos de este guión de prácticas son los siguientes:

- Resolver un problema eligiendo la mejor estructura de datos para las operaciones que se solicitan

Los requisitos para poder realizar esta práctica son:

1. Haber estudiado el Tema 1: Introducción a la eficiencia de los algoritmos
2. Haber estudiado el Tema 2: Abstracción de datos. Templates.
3. Haber estudiado el Tema 3: T.D.A. Lineales.
4. Haber estudiado el Tema 4: STL e Iteradores.
5. Haber estudiado estructuras de datos jerárquicas: Árboles

2. Objetivo.

El objetivo de esta práctica es llevar a cabo el análisis, diseño e implementación de un proyecto. Con tal fin el alumno abordará un problema donde se requiere estructuras de datos que permiten almacenar grandes volúmenes de datos y poder acceder a ellos de la forma más eficiente.

3. Ejercicios

El alumno deberá implementar varios programas. Uno de ellos simulará al famoso juego "Cifras y Letras". En este caso solamente deberá generar la sección de Letras. Este juego consiste en dado una serie de letras escogidas de forma aleatoria obtener la palabra existente en el diccionario de mayor longitud.

Por ejemplo dadas las siguientes letras:

U D Y R M E T

una de las mejores soluciones sería TRUE.

3.1. Tareas a realizar

El alumno debe llevar a cabo las siguientes tareas:

1. Con objeto de que alumno practique con el T.D.A ArbolGeneral, se pide que implemente este tipo de dato. En base a este tipo de dato construya el T.D.A. Diccionario (almacena las palabras en un ArbolGeneral de un diccionario).
2. Definir el resto de T.DA. que vea necesario para la solución de los problemas propuestos.
3. Probar los módulos con programas test.
4. Construir los programas que a continuación se detallan.

Se puede usar la STL en todos los módulos excepto en la implementación del TDA ArbolGeneral.

A continuación se detallan los programas que se deberán desarrollar.

3.1.1. Test diccionario

Este programa permitirá comprobar el buen funcionamiento del T.D.A Diccionario, representado como un ArbolGeneral. Para ello el código fuente “testdiccionario.cpp” deberá funcionar con los T.D.A ArbolGeneral y Diccionario (ver sección 4) desarrollados.

```
1.  #include <fstream>
2.  #include <iostream>
3.  #include <string>
4.  #include <vector>
5.  #include "diccionario.h"
6.  int main(int argc, char * argv[]){
7.   if (argc!=2){
8.            cout<<"Los parametros son:"<<endl;
9.            cout<<"1.- El fichero con las palabras";
10.           return 0;
11.  }
12.  ifstream f(argv[1]);
13.  if (!f){
14.           cout<<"No puedo abrir el fichero "<<argv[1]<<endl;
15.   return 0;
16.  }
17.  Diccionario D;
18.  cout<<"Cargando diccionario...."<<endl;
19.  f>>D;
20.  cout<<"Leido el diccionario..."<<endl;
21.  cout<<D;
22.
23.  int longitud;
24.   cout<<"Dime la longitud de las palabras que quieres ver";
25.  cin>>longitud;
26.                                   vector<string> v=D.PalabrasLongitud(longitud);
27.
28.           cout<<"Palabras de Longitud "<<longitud<<endl;
29.  for (unsigned int i=0;i<v.size();i++)
30.   cout<<v[i]<<endl;
31.
32.  string p;
33.  cout<<"Dime una palabra: ";
34.  cin>>p;
35.  if (D.Esta(p))   cout<<"Sí esa palabra existe";
36.  else   cout<<"Esa palabra no existe";
37. }
```

En este código, se carga el diccionario en memoria y luego se imprime por la salida estándar. A continuación se muestra todas las palabras del diccionario de una longitud dada. Y finalmente dada una palabra por el usuario, el programa indica si existe tal palabra en el diccionario o no. El alumno creará por lo tanto el programa testdiccionario, que se deberá ejecutar en la línea de órdenes de la siguiente manera:

```
prompt%  testdiccionario spanish
```

El único parámetro que se da al programa es el nombre del fichero donde se almacena el diccionario.

3.1.2. Letras

Este programa construye palabras de longitud mayor (o puntuación mayor) a partir de una serie de letras seleccionadas de forma aleatoria. El programa letras se deberá ejecutar en la línea de órdenes de la siguiente manera:

```
prompt%  letras spanish letras.txt 8 L
```

Los parámetros de entrada son los siguientes:

1. El nombre del fichero con el diccionario
2. El nombre del fichero con las letras
3. El numero de letras que se deben generar de forma aleatoria
4. Modalidad de juego:
    - Longitud: Si el parámetro es *L* se buscará la palabra más larga.
    - Puntuación: Si el parámetro es *P* se obtendrá la palabra de mayor puntuación.

Tras la ejecución en pantalla aparecerá lo siguiente:

```
Las letras son: T       I       E       O       I       T       U       S
Dime tu solucion:tieso
tieso   Puntuacion: 5

Mis soluciones son:
otitis  Puntuacion: 6
tiesto  Puntuacion: 6
Mejor Solucion:tiesto
¿Quieres seguir jugando[S/N]?N
```

En primer lugar el programa genera 8 letras. Estas letras se escogen, de forma aleatoria, entre las dadas en el fichero letras (ver sección 3.2.2). Una vez generadas las letras, el programa pide al usuario su solución, en el ejemplo la solución dada es “tieso”. A continuación se muestra la solución del usuario junto con su puntuación. Y finalmente se muestra las soluciones dadas por el programa. Para generar de forma aleatoria las letras con la que construir la palabra, el alumno creará una *Bolsa de Letras* (contenedor de letras que se disponen de forma aleatoria) en la que el número de veces que aparece cada letras, en la *Bolsa de Letras*, viene dado por la columna *Cantidad* del fichero de letras.

En el caso de que el usuario haya escogido jugar en modo “*Puntuacion*”, como resultado se

```
******Puntuaciones Letras*******       Z       10
A       1
B       3                             Las letras son:
C       3                             N       S       A       O       T       O       A       I
D       2                             Dime tu solucion:sonata
E       1
F       4
G       2                             sonata  Puntuacion: 6
H       4                             Mis Soluciones son:
I       1                             asiano  Puntuacion: 6
J       8                             atonia  Puntuacion: 6
L       1                             ostion  Puntuacion: 6
M       3                             sonata  Puntuacion: 6
N       1                             sotana  Puntuacion: 6
O       1                             sotani  Puntuacion: 6
P       3                             sotano  Puntuacion: 6
Q       5                             tisana  Puntuacion: 6
R       1                             toison  Puntuacion: 6
S       1                             Mejor Solucion:toison
T       1                             ¿Quieres seguir jugando[S/N]?N
U       1
V       4
Y       4
```

obtendrán las palabras que acumulen una mayor puntuación. Para obtener la puntuación de la palabra simplemente tenemos que sumar las puntuaciones de las letras en la palabra (en el fichero de Letras viene descrita en la columna Puntos).

En ambas versiones, se le preguntará al usuario si quiere seguir jugando. En caso afirmativo se generará una nueva secuencia de letras aleatorias para jugar de nuevo. En otro caso el programa termina.

En la sección 3.2 se detallan los formatos de los ficheros de entrada al programa.

3.1.3. Cantidad de Letras

El programa cantidad_letras obtiene la cantidad de instancias de cada letra (ver fichero letras en la sección 3.2.2). El programa se deberá ejecutar en la línea de órdenes de la siguiente manera:

```
prompt%  cantidad_letras spanish letras.txt salida.txt
```

Los parámetros de entrada son los siguientes:

1. El nombre del fichero con el diccionario
2. El nombre del fichero con las letras
3. El fichero donde escribir el conjunto de letras con la cantidad de aparición calculada.

Este programa una vez haya cargado el fichero diccionario en memoria y el conjunto de letras, obtiene para cada letra en el conjunto el numero de veces que aparece en el diccionario, es decir encuentra la frecuencia de aparición. Finalmente obtiene el tanto por ciento, sobre el total de las frecuencias, del numero de veces que aparece cada letra. Este valor será la cantidad.

A modo de ejemplo para la anterior ejecución, el fichero salida.txt obtenido es:

```
#Letra Cantidad Puntos
A       16      1
B       2       3
C       6       3
D       5       2
E       10      1
F       2       4
G       2       2
H       1       4
I       9       1
J       1       8
L       5       1
M       4       3
N       7       1
O       10      1
P       3       3
Q       1       5
R       10      1
S       5       1
T       6       1
U       4       1
V       2       4
X       1       8
Y       1       4
Z       1       10
```

3.2. Ficheros

3.2.1. Fichero diccionario

El fichero diccionario se componen de un conjunto de palabras, cada una en un línea. Este conjunto de palabras serán las palabras que se consideren como válidas.

Un ejemplo de fichero diccionario es el siguiente:

```
a
aaronica
aaronico
ab
abab
ababillarse
ababol
abaca
abacera
abaceria
abacero
abacial
abaco
abad
abada
abadejo
abadenga
abadengo
```

3.2.2. Fichero Letras

Un ejemplo de fichero de letras es el que se muestra a la derecha.

El formato del fichero es el siguiente:

1. En primer lugar aparece una línea encabezada con el carácter # donde se describe las columnas del fichero (Letra Cantidad Puntos)
2. A continuación cada línea se corresponde con la información de una letra:
   - Valor de la letra
   - Número de veces que aparece la letra en la Bolsa de Letras
   - Puntos asignados a la letra.

```
#Letra Cantidad Puntos
A       12      1
E       12      1
O       9       1
I       6       1
S       6       1
N       5       1
L       1       1
R       6       1
U       5       1
T       4       1
D       5       2
G       2       2
C       5       3
B       2       3
M       2       3
P       2       3
H       2       4
F       1       4
V       1       4
Y       1       4
Q       1       5
J       1       8
X       1       8
Z       1       10
```

4. Módulos a desarrollar

Módulo Arbol General

Para crear los programas propuestos hará falta desarrollar el T.D.A ArbolGeneral. Este módulo debe ser lo suficiente flexible para poder desarrollar los tres programas propuestos. La interfaz y representación propuesta para ArbolGeneral la podéis encontrar en el fichero ArbolGeneral.h. El alumno implementará los métodos propuestos de acuerdo a la especificación dada. El T.D.A ArbolGeneral implementa un árbol en el que cada nodo puede tener un numero indeterminado de hijos. La información que almacena un nodo es:

- El elemento de tipo Tbase (plantilla)
- Un puntero al hijo más a la izquerda
- Un puntero al padre
- Un puntero al hermano a la derecha.

Con esta definición de nodo el T.D.A ArbolGeneral se puede representar como un puntero al nodo raiz. Por ejemplo un objeto de tipo ArbolGeneral, instanciando cada elemento a char, es el que se muestra en la Figura 1.

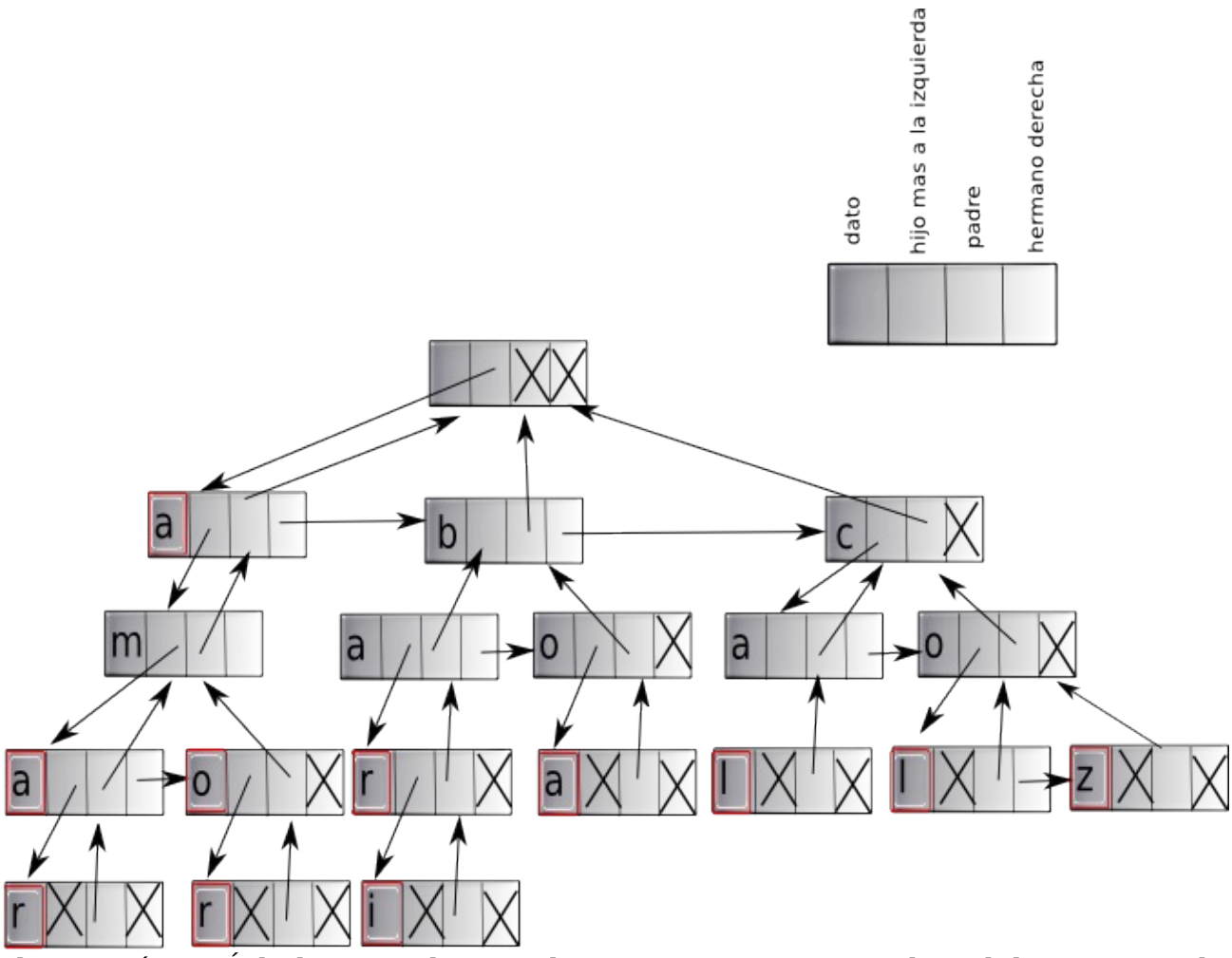


Ilustración 1: Árbol General para almacenar un conjunto de palabras. En cada nivel tenemos una letra de la palabra. Si la letra se encuentra encuadrada en rojo significa que desde el nivel 1 hasta ella se ha formado una palabra válida

Con respecto a las operaciones que se detallan, cabe destacar la posibilidad que el alumno implemente dentro de la clase ArbolGeneral un iterador que permita recorrer la clase en preorden.

```
#ifndef __ArbolGeneral_h__
#define __ArbolGeneral_h__
template <class Tbase>
class ArbolGeneral{
private:
….
public:
    ...
   class iter_preorden{
       private:
         Nodo it;
         Nodo raiz;
         int level; //altura del nodo it dentro del arbol  con raiz “raiz”
       public:
         iter_preorden();

         Tbase & operator*();

         int getlevel();

         iter_preorden & operator ++();

         bool operator == (const iter_preorden &i);

         bool operator != (const iter_preorden &i);

         friend class ArbolGeneral;
    };


     iter_preorden begin();

     iter_preorden end();

};
#endif
```

El método begin inicializa un iter_preorden para que apunte a la raiz (el primer nodo en recorrido preorden). La función end inicializa un iter_preorden para que apunte al nodo nulo.

Módulo Diccionario

También será necesario construir el T.D.A Diccionario. Un objeto del T.D.A. Diccionario almacena palabras de un lenguaje. El T.D.A Diccionario será representado como un ArbolGeneral instanciado a *info*. En el código que se muestra a continuación el ArbolGeneral se instancia a *info* que es un *struct* con dos campos carácter y un booleano que indica si desde la raíz hasta el nodo se construye una palabra que está en el diccionario. Así en líneas generales el módulo Diccionario se detalla a continuación:

```
#ifndef __Diccionario_h__
#define __Diccionario_h__
#include “ArbolGeneral.h”
struct info{
  char c; ///<< caracter que se almacena en un nodo
  bool final; ///< nos indica si es final o no de palabra
  info(){
    c='\0';
    final=false;
  }
  info(char car, bool f):c(car),final(f){}
};
class Diccionario{
 private:
   ArbolGeneral<info> datos;
 public:
      /**
        @brief Construye un diccionario vacío.
      **/

      Diccionario()
      /**
        @brief Devuelve el numero de palabras en el diccionario
      **/
      int size() const ;

      /**
        @brief Obtiene todas las palabras en el diccionario de un longitud dada
        @param longitud: la longitud de las palabras de salida
        @return un vector con las palabras de  longitud especifica en el parametro de entrada
      **/
      vector<string> PalabrasLongitud(int longitud);

       /**
        @brief Indica si una palabra está en el diccionario o no
        @param palabra: la palabra que se quiere buscar
        @return true si la palabra esta en el diccionario. False en caso contrario
      **/
      bool Esta(string palabra);
       /**
        @brief Lee de un flujo de entrada un diccionario
        @param is:flujo de entrada
        @param D: el objeto donde se realiza la lectura.
        @return el flujo de entrada
      **/
      friend   istream & operator>>(istream & is,Diccionario  &D);
       /**
        @brief Escribe en un flujo de salida un diccionario
        @param os:flujo de salida
        @param D: el objeto diccionario que se escribe
        @return el flujo de salida
      **/
      friend ostream & operator<<(ostream & os, const Diccionario  &D);
};

#endif
```

En la figura 1, se puede ver un Diccionario que almacena las palabras :

De igual forma que lo hicimos en la clase ArbolGeneral podríamos generar un iterador sobre el T.D.A Diccionario que nos permita recorrer de manera ordenada todas las palabras del diccionario. Una posible especificación y representación de este iterador es:

```
#ifndef __Diccionario_h__
#define __Diccionario_h__
#include “ArbolGeneral.h”
struct info{
 ….
};
class Diccionario{
 private:
   ArbolGeneral<info> datos;
 public:
  …

 class iterator{
    private:
       ArbolGeneral<info>::iter_preorden it;
       string cad; //mantiene los caracteres desde el nivel 1 hasta donde se encuentra it.
    public:
       iterator ();
       string operator *();
       iterator & operator ++();
       bool operator ==(const iterator &i)
       bool operator !=(const iterator &i)
       friend class Diccionario;
 };
 iterator begin();

 iterator end();


};

#endif
```

Módulo Letra y Conjunto de Letras

El T.D.A Letra almacena una letra. Una letra se especifica con tres valores:

1. El carácter de la propia letra
2. La cantidad de veces que puede aparecer.
3. La puntuación de una letra.

El T.D.A Conjunto_Letras permitirá tener en memoria un fichero Letras. Este T.D.A se define como una colección de letras, en las que no hay letras repetidas.

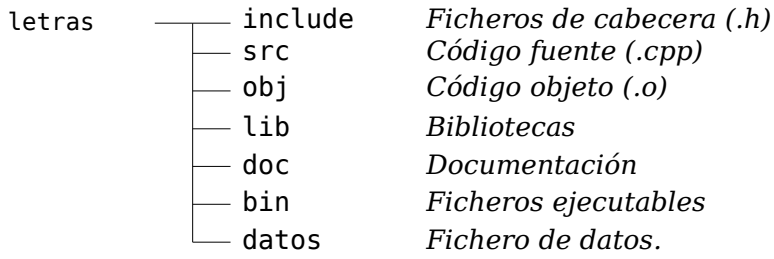
Módulo Bolsa de Letras

Este módulo será útil para el programa *letras.* El T.D.A Bolsa_Letras almacena caracteres correspondientes a una letra de un Conjunto de Letras. Este carácter aparece en la Bolsa_Letras repetido tantas veces como diga el campo cantidad de la letra. Por lo tanto en la Bolsa de Letras aparecen las letras de forma aleatoria. En el programa *"letras"* la secuencia de letras con las que se juega se cogen de la Bolsa de Letras.

5. Práctica a entregar

El alumno deberá empaquetar todos los archivos relacionados en el proyecto en un archivo con nombre "*letras.tgz*" y entregarlo antes de la fecha que se publicará en la página web de la asignatura. Tenga en cuenta que no se incluirán ficheros objeto, ni ejecutables, ni la carpeta datos. Es recomendable que haga una "limpieza" para eliminar los archivos temporales o que se puedan generar a partir de los fuentes.

El alumno debe incluir el archivo *Makefile* para realizar la compilación. Tenga en cuenta que los archivos deben estar distribuidos en directorios*:*



Para realizar la entrega, en primer lugar, realice la limpieza de archivos que no se incluirán en ella, y sitúese en la carpeta superior (en el mismo nivel de la carpeta "*letras*") para ejecutar:

```
prompt% tar zcv letras.tgz letras
```

tras lo cual, dispondrá de un nuevo archivo letras.tgz que contiene la carpeta letras así como todas las carpetas y archivos que cuelgan de ella.

6. Referencias

- [GAR06b] Garrido, A. Fdez-Valdivia, J. "*Abstracción y estructuras de datos en C++*". Delta publicaciones, 2006.