Práctica Final: Sopa de Letras III

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

DECSAI

Estructuras de Datos

Grado en Ingeniería Informática C

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 objetivo al igual que en la práctica 2 y 3 es crear el clásico pasatiempo “Sopa de Letras”. En esta práctica tenemos a nuestra disposición un conjunto de ontologías con sus definiciones Una ontología es un medio para categorizar o agrupar palabras. Sobre este grupo de ontologías vamos a construir la Sopa de Letras. Para ello el usuario escoge un nivel de temática. Conforme el nivel sea más alto la temática será más especializada, si el nivel es más bajo será más generalista (más cerca de la raíz). Una vez escogido el nivel de temática se muestra los temas que se tratan en dicho nivel. Y a continuación el usuario escoge el número de temática y se crea la sopa de letras con las palabras finales (en las hojas del árbol) que se correspondan con dicha temática.

Las ontologías se han dado en una estructura jerárquica, árbol general, en el que en la raíz tenemos la clase que recoge a todas las palabras, la temática mas generalista. En el siguiente nivel tenemos todas sus subclases, y así sucesivamente como se puede observar en las Figuras 1 y 2. Así por ejemplo en la figura 1 la raíz representa la temática más general que es “entity”. A continuación las siguientes temáticas más generales son los hijos del nodo raíz “physical entity” y “ abstract entity, abtraction”. Esta última temática tiene dos sinónimos “abstract entity” y “abstraction”. En las hojas tenemos las palabras finales con sus sinónimos. Asi dentro de la temática “sea” tenemos “adriatic sea, adriatic” (dos sinónimos), “marmora, sea of marmora, sea of marmora, marmara denizi, marmara”, y “mediterranean sea, mediterranea”.

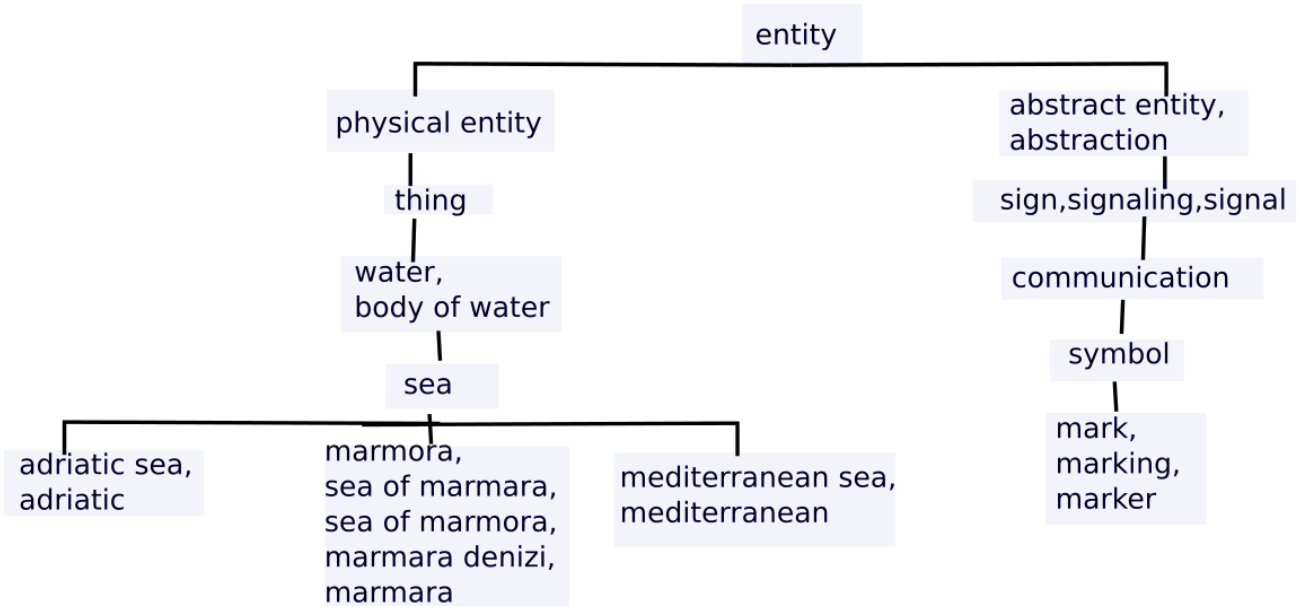


Figura 1: Ontologías dispuestas en un Árbol General para almacenar un conjunto de palabras con sus clasificaciones. En cada nodo interior tenemos el nombre de una temática con sus sinónimos. En los nodos hojas tenemos la palabra final con sus sinónimos. Todos los ancestros de un nodo hoja representan las temáticas a las que pertenece el nodo hoja. Así para la temática "sea" tenemos el mar Adriático (con sus posibilidades en inglés para nombrarlo), el mar Mármora y el mar Mediterráneo. El nodo raíz representa la temática más general, en este caso sería "entity".

3.1. Tareas a realizar

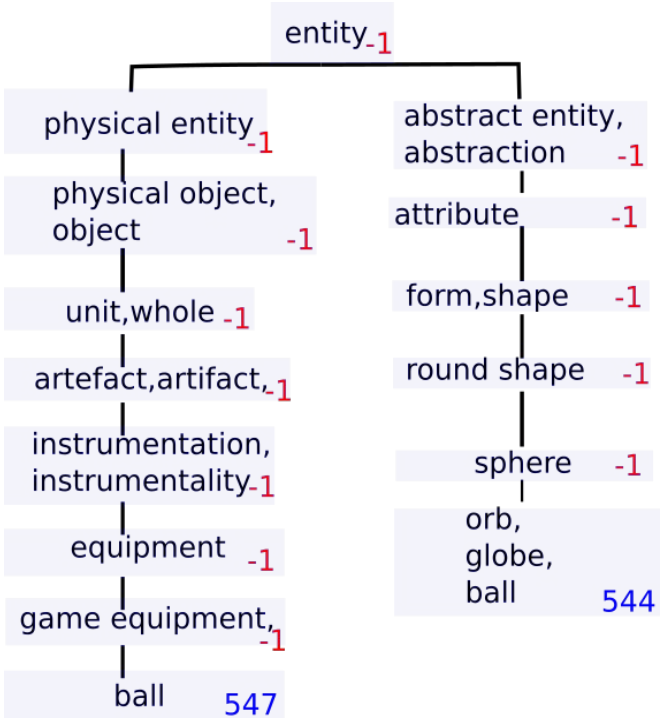
El alumno/a debe llevar a cabo las siguientes tareas:

1. Reusar los módulos matriz_dispersa y sopa_letras de la práctica 3.
2. Con objeto de que alumno/a practique con el T.D.A ArbolGeneral, se pide que defina el T.D.A Ontologías. Para representar Ontologias lo haremos con un objeto de tipo ArbolGeneral en el que las hojas representan las palabras finales. En cualquier nodo puede haber mas de una palabra (palabra final si es hoja) pero con igual significado (sinónimos). Así se muestra por ejemplo en la Figura 1. Además asociado a las palabras finales (hojas) tendremos una definición (ver Figura 2). Para ello en el árbol habrá un índice que nos conduce al significado de la palabra final (en el nodo hoja). En el caso de que el nodo no sea un nodo hoja este índice tiene valor -1. El fichero Ontologias.h se puede ver en la sección 5.2 . También este fichero aparece en el material asociado a la práctica. Por lo tanto con la interfaz propuesta el alumno/a deberá implementar los diferentes métodos y funciones necesarias. El alumno/a puede incluir además otras funciones que vea que son necesarias.
3. El alumno/a para poder mantener la información de la temática y nivel de temática escogido por el usuario deberá definir el TDA PreguntasED. Esta clase contiene el conjunto de ontologias del que se obtiene las palabras y definiciones escogidas por el

usuario. Con estas palabras y preguntas se creará la sopa de letras. La representación e interfaz de este módulo se puede ver en la sección 5.3. Este fichero también puede encontrarlo en el material asociado a la práctica. Por lo tanto con la interfaz propuesta el alumno/a deberá implementar los diferentes métodos y funciones necesarias. El alumno puede incluir además otras funciones que vea que son necesarias.

4. Definir el resto de módulos que vea necesario para la solución de los problemas propuestos.
5. Probar los módulos con programas test.
6. Construir los programas que a continuación se detallan.

Se puede usar la STL en todos los módulos.



Significados

....

544 a more or less rounded anatomical body

....

547 round object that is hit or thrownor kicked in games "the ball travelled 90 mph on his serve" "the mayor threw out the first ball" "the ball rolled into the corner pocket"

*Figura 2: Ontologías dispuestas en un Árbol General para almacenar un conjunto de palabras con sus clasificaciones. En este ejemplo se puede observar como una palabra "ball" puede pertenecer a diferentes conjuntos porque tiene diferentes significados. En cada nodo tenemos un índice que se corresponde con (-1) si es un nodo no hoja o con un valor entre 0 y n siendo el indice de la definición de la palabra hoja. Así "ball" como hijo de "*sphere" tiene como significado "a more or less rounded anatomical body".

A continuación se detallan los programas que se deberán desarrollar.

3.1.1. Test Ontologias y PreguntasED

Este programa permitirá comprobar el buen funcionamiento del T.D.A Ontologias y PreguntasED, además de familiarizarse con el T.D.A ArbolGeneral. Para ello el alumno/a dispone en el material el fichero "prueba_Ontologias_PreguntasEd.cpp" que se muestra a continuación:

```
//FICHERO PRUEBA ONTOLOGIAS Y PREGUNTASED
int main(int argc, char * argv[]){
 if (argc!=3){
    cout<<"Dime el fichero con la estructura jerarquica de las
       palabras"<<endl;//Arbol_Ontologias.txt
    cout<<"Dime el fichero con los significados"<<endl; //significados.txt
    return 0;
 }
/************************************************/
 //Seccion 1; probando el arbol
 ifstream f (argv[1]);
 ArbolGeneral<pair<set<string>,int> > ab;
 f>>ab;
 ArbolGeneral<pair<set<string>,int> >::iter_preorden it;
 for (it=ab.begin();it!=ab.end();++it){
   if (it.getlevel()<3){
     for (int i=0;i<=it.getlevel();i++)
       cout<<"----";
     cout<<it.getlevel()<<".-";
    pair<set<string>,int> aux= *it;
    set<string>::iterator sit=aux.first.begin();
    while (sit!=aux.first.end()){
        cout<<*sit<<", ";++sit;
    }
    cout<<endl;

   }
 }
//fin de prueba del arbol
/************************************************/
// Sección 2:probando ontologias
 Ontologias Ot;
 //Lee la estructura jerarquica de las palabras y sus
 //significados.
 Ot.Lee(argv[1],argv[2]);


 //comprobar que es correcta la lectura escribiendo ontologias.
 string test_salida1=argv[1]+"back";
 string test_salida2=argv[2]+"back";
 Ot.Escribe(test_salida1,test_salida2);


 int level;
 cout<<"Dime un nivel de tematica (1),(2),(3)"<<endl;
 cin>>level;

 cout<< "Las temáticas posibles a nivel"<<level<<" son:"<<endl;
 Ontologias::iterator_level itl;//iterador que itera por niveles del arbol
```

```
int cnt=1;
for (itl=Ot.beginl(level);itl!=Ot.endl(); ++itl){
   pair<set<string>,int> aux= *itl;
   set<string>::iterator sit=aux.first.begin();
   cout<<"Temática "<<cnt;
   cnt++;
   while (sit!=aux.first.end()){
        cout<<*sit<<", ";++sit;
   }
   cout<<endl;
 }
 cin.get();cin.get();
 ////fin de prueba de ontologias

//Sección 3: probando PreguntasED
 cout<<"Test PreguntasED*********************************"<<endl;
 PreguntasED Ask(Ot);
 Ask.MuestraTematicas();
 cout<<"Escoge una de las tematicas posibles:";
 int ntema;
 cin>>ntema;
 Ask.SetTematica(ntema);
 Ask.IniciaConceptosTemaEscogido();
 cout<<"*************************************"<<endl;
 cin.get();
 Ask.BarajarPreguntas();
 int i=0;
 while (i<Ask.num_preguntas()){
     pair<set<string>,string> p= Ask.SacaPregunta();
     cout<<p.second<<"?";
     string con;
     getline(cin,con);
     if (p.first.find(con)!=p.first.end()){
       cout<<"Correcta"<<endl;
     }
     else
        cout<<"NO es correcta la contestacion es cualquiera de las palabras: "<<p.first<<endl;
     ++i;
 }
}
//FIN DE PRUEBA ONTOLOGIAS PREGUNTASED
```

Para poder ejecutar este código habrá que llamarlo desde la línea de ordenes como:

```
prompt%  prueba_Ontologias_PregutnasED datos/Arbol_Ontologias.txt datos/significados.txt
```

El primer parámetro se corresponde con el nombre del fichero que almacena la estructura jerárquica del conjunto de Ontologias (ver sección 4.1.1)y el segundo es el nombre del fichero que almacena los significados de las palabras finales (ver sección 4.1.2).

Este programa consta de tres secciones:

- **Sección 1. Prueba ArbolGeneral.** Carga en memoria, en un objeto de tipo ArbolGeneral, la estructura jerárquica dada en el fichero Arbol_Ontologias.txt. A continuación muestra toda la estructura comprendida hasta el nivel 3 usando un iterador de ArbolGeneral.

- **Sección 2. Prueba Ontologías.** Carga en memoria la estructura jerárquica con los significados correspondientes de las palabras finales en un objeto de tipo Ontología (Ot). A continuación escribe la ontología en dos ficheros con extensión back. Estos ficheros deben ser idénticos a los leídos en argv[1] y argv[2]. A continuación le pide al usuario que introduzca un nivel de temática y le muestra todos los temas asociados a dicho nivel.
- **Sección 3. Prueba PreguntasED.** Construye un objeto de tipo PreguntasED con el objeto Ot (de tipo Ontologias) creado en la sección 2 y el nivel de temática por defecto que es 4 (ver el fichero PreguntasED.h). A continuación muestra las temáticas en el nivel y le pide al usuario que escoja una de ellas. Con esta temática inicia todas las preguntas. La siguiente sentencia baraja las preguntas para que se muestren de forma aleatoria. Después de esto, al usuario le muestra una definición y este debe de insertar una palabra que se corresponde con la definición. El programa comprobara si es correcta o no la respuesta y así se lo dirá al usuario y pasara a la siguiente pregunta. Si no es correcta le indica la contestación correcta.

3.2. Sp3

El objetivo de este ejercicio es construir un programa que dado un conjunto de ontologías (estructura jerárquica y significados de las palabras finales) , y número de palabras, este genere una sopa de letras con tal número de palabras. Las palabras pertenecen a una temática escogida. Dos ejemplos de llamada al programa sería:

```
prompt% sp3 datos/Arbol_Ontologias.txt datos/significados.txt 5


prompt% sp3 datos/Arbol_Ontologias.txt datos/significados.txt 5 9
```

Para ambas llamadas se indica que se genere una sopa de letras con 5 palabras como máximo. Pero en la primera llamada se mostrará al usuario todas las temáticas posibles (en el directorio datos el alumno puede ver todas las temáticas posibles en el fichero Tematicas.txt), y se le pedirá que inserte el nivel de la temática. En la segunda llamada directamente en la línea de comandos el usuario está indicando que el nivel de temática que quiere es el 9. En el nivel 9 tenemos un total de 497 temáticas. Un subconjunto de temáticas posibles en el nivel 9 son las siguientes:

```
Tema 433.-waggon,wagon,
Tema 434.-nail,
Tema 435.-pin,
Tema 436.-coil,
Tema 437.-electro-acoustic transducer,
Tema 438.-general-purpose bomb,gp bomb,
Tema 439.-screen,sieve,
Tema 440.-blade,brand,steel,sword,
Tema 441.-gun,
Tema 442.-knife,
Tema 443.-lance,shaft,spear,
Tema 444.-missile,projectile,
Tema 445.-horologe,timekeeper,timepiece,
Tema 446.-magnifier,
Tema 447.-forte-piano,piano,pianoforte,
Tema 448.-bowed stringed instrument,string,
Tema 449.-brass,brass instrument,
Tema 450.-wood,woodwind,woodwind instrument,
Tema 451.-shaper,shaping machine,
Tema 452.-engine,
Tema 453.-acoustic gramophone,gramophone,
Tema 454.-typesetting machine,
Tema 455.-loom,
Tema 456.-electric switch,electrical
switch,switch,
Tema 457.-regulator,
Tema 458.-diaphragm,stop,
Tema 459.-machine,simple machine,
Tema 460.-striker,
Tema 461.-vibrator,
Tema 462.-brace,bracing,
Tema 463.-upright,vertical,
Tema 464.-fishing net,fishnet,
Tema 465.-gas burner,gas jet,
Tema 466.-reproducer,
Tema 467.-net,
Tema 468.-checker,chequer,
Tema 469.-chess piece,chessman,
Tema 470.-tape machine,tape recorder,
Tema 471.-gymnastic horse,horse,
Tema 472.-club,golf club,golf-club,
```

Si a continuación el usuario escoge el tema 469, el programa crea la sopa con la temática ***"chess piece,chessman".***

Una posible salida es la siguiente:

```
Colocadas las palabras
Definición--> a chessman shaped to resemble the head of a horse can move two squares horizontally
and one vertically (or vice versa)
Definición--> (board games) the lighter pieces
Definición--> (chess) a piece that can be moved diagonally over unoccupied squares of the same
color
Definición--> a female sovereign ruler
Definición--> (chess) the weakest but the most important piece
TITULO : chess piece,chessman,  Numero de Palabras Ocultas: 5 Numero de Palabras Descubiertas: 0

*********************************
                    -3      -2      -1      0       1       2       3       4
               -3   B       Q       U       E       E       N       M       F
               -2   K       I       C       T       T       Z       Z       E
               -1   I       J       S       I       G       G       J       Z
               0    N       C       E       H       O       R       S       E
               1    G       E       K       W       O       T       Y       C
               2    H       J       V       R       C       P       F       A

Dime una palabra: HORSE
Dime la fila :0
Dime la columna :0
Direccion Arriba (vu), Abajo (vd), Izquierda (hi) , Derecha (hd), Diagonal abajo derecha
(dd),Diagonal abajo izquierda (di) :hd
```

El usuario viendo la primera definición inserta HORSE en la fila 0 columna 0 en dirección horizontal derecha. Como es correcta el programa la marca en negrita y la elimina de las preguntas que debe volver a hacer. Obteniendo la siguiente salida.

```
Definición--> (board games) the lighter pieces
Definición--> (chess) a piece that can be moved diagonally over unoccupied squares of the same color
Definición--> a female sovereign ruler
Definición--> (chess) the weakest but the most important piece
TITULO : chess piece,chessman,  Numero de Palabras Ocultas: 4 Numero de Palabras Descubiertas: 1

*********************************
                    -3      -2      -1      0       1       2       3       4
               -3   B       Q       U       E       E       N       X       M
               -2   K       I       J       T       O       E       C       V
               -1   I       D       S       I       K       Q       S       B
               0    N       X       R       H       O       R       S       E
               1    G       L       N       W       O       F       S       S
               2    N       C       U       B       U       P       O       B


Dime una palabra:
```

Así continuaría el programa pidiendo al usuario que inserte palabra hasta que descubra todas las palabras.

```
TITULO : chess piece,chessman,  Numero de Palabras Ocultas: 0 Numero de Palabras Descubiertas: 5

********************************
                    -3      -2      -1      0       1       2       3       4
                -3  B       Q       U       E       E       N       J       V
                -2  K       I       A       T       C       Y       Q       Y
                -1  I       Z       S       I       V       W       D       J
                0   N       Q       W       H       O       R       S       E
                1   G       M       M       W       O       F       S       L
                2   X       K       X       P       F       P       J       X
Congratulations!!
```

3.2.1. Como colocar las palabras en la Sopa de Letras

La forma más evidente de que nuestro programa coloque las palabras sobre la sopa de letras es escoger de forma aleatoria tanto la fila, columna y orientación. Pero como una opción posible que se tendrá en cuenta como **algo adicional a** realizar es crear la sopa de letras con el menor número de filas y columnas. Para ello la primera palabra se escoge colocarla en una fila y columna de forma aleatoria. A continuación se busca la palabra que coincide con esta en alguna letra como máximo. En el ejemplo de ejecución anterior por ejemplo se colocó HORSE y a continuación WHITE usando la H como letra común. También se uso la H para insertar BISHOP. Por otro QUEEN usa la E de WHITE. Las palabra restante, KING, no se pudo poner sobre estas y por lo tanto se escogieron posiciones de forma que no hicieran crecer mucho la Sopa de Letras.

Fijaros que con esta técnica crear el juego SCRABBLE no sería difícil, pero bueno esto será para otro año.

4. Ficheros de datos

4.1.1. Fichero de Arbol de Ontologías

El fichero de ontologías es un fichero que describe la estructura jerárquica de las ontologías. En este sentido se almacena un árbol en preorden con nodos virtuales (con valor 'x') para indicar que ya no existe hijo a la izquierda o hermanos a la derecha o con un valor 'n' si existe y a continuación el valor del nodo. Así para el árbol que se muestra en la figura 3 una parte de lo que se almacenaría en disco sería:

n0nanmnanrxxnonrxxxnbna....

Siendo las n para indicar que existe nodo y las x para que indicar que no existe. En nuestro caso para almacenar las ontologias en vez de carácter almacenamos las palabras que describen la temática o la palabra final si se refiere a un nodo hoja. Para el ejemplo de la figura 2 tendríamos en disco:

n1 entity-1n1 physical entity,-1n2 physical object,object,-1 n2 unit, whole,-1 n 2 artefact, artifact,-1 n2 instrumentation, instrumentality,-1n1 equipment,-1n1 game equipment,-1n1 ball,547....

En este ejemplo cuando existe nodo se escribe el carácter 'n' y a continuación el número de palabras sinónimas (separados por coma) y el índice del significado en el fichero significados.txt. Si no es una palabra final (o nodo hoja) este índice es -1.

4.1.2. Fichero con los significados

El fichero con los significados contiene un índice o clave de la palabra final y a continuación el significado. Un ejemplo de este fichero es el siguiente:

```
0 the 1st letter of the Roman alphabet
1 (biochemistry) purine base found in DNA and RNA pairs with thymine in DNA and with uracil in RNA
2 the basic unit of electric current adopted under the Systeme International d'Unites "a typical
household circuit carries 15 to 50 amps"
3 a metric unit of length equal to one ten billionth of a meter (or 0.0001 micron) used to specify
wavelengths of electromagnetic radiation
4 any of several fat-soluble vitamins essential for normal vision prevents night blindness or
inflammation or dryness of the eyes
5 one of the four nucleotides used in building DNA all four nucleotides have a common phosphate
group and a sugar (ribose)
6 the blood group whose red cells carry the A antigen
7 a city in western Germany near the Dutch and Belgian borders formerly it was Charlemagne's
northern capital
8 (Old Testament) elder brother of Moses and first high priest of the Israelites created the golden
calf
9 United States professional baseball player who hit more home runs than Babe Ruth (born in 1934)
10 (Old Testament) Cain and Abel were the first children of Adam and Eve born after the Fall of Man
Abel was killed by Cain
11 Norwegian mathematician (1802-1829)
12 French philosopher and theologian lover of Heloise (1079-1142)
13 a town in western Washington
14 a city in northeastern Scotland on the North Sea
15 city recognized by the United States as the capital of the Ivory Coast largest city of the Ivory
Coast
16 a town in central Kansas to the west of Topeka home of Dwight D. Eisenhower
17 a city in central Texas
...
```

5. Módulos a desarrollar

5.1. Módulo Árbol General

Aunque en esta práctica se le da al alumno implementado el T.D.A ArbolGeneral, si es recomendable que estudie su implementación y representación para un mejor uso. La interfaz y representación propuesta para ArbolGeneral la podéis encontrar en el fichero ArbolGeneral.h. El T.D.A ArbolGeneral implementa un árbol en el que cada nodo puede tener un número indeterminado de hijos. La información que almacena un nodo es:

- El elemento de tipo Tbase (plantilla)
- Un puntero al hijo más a la izquierda
- 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 raíz. Por ejemplo un objeto de tipo ArbolGeneral, instanciando cada elemento a char es el que se muestra en la Figura 3.

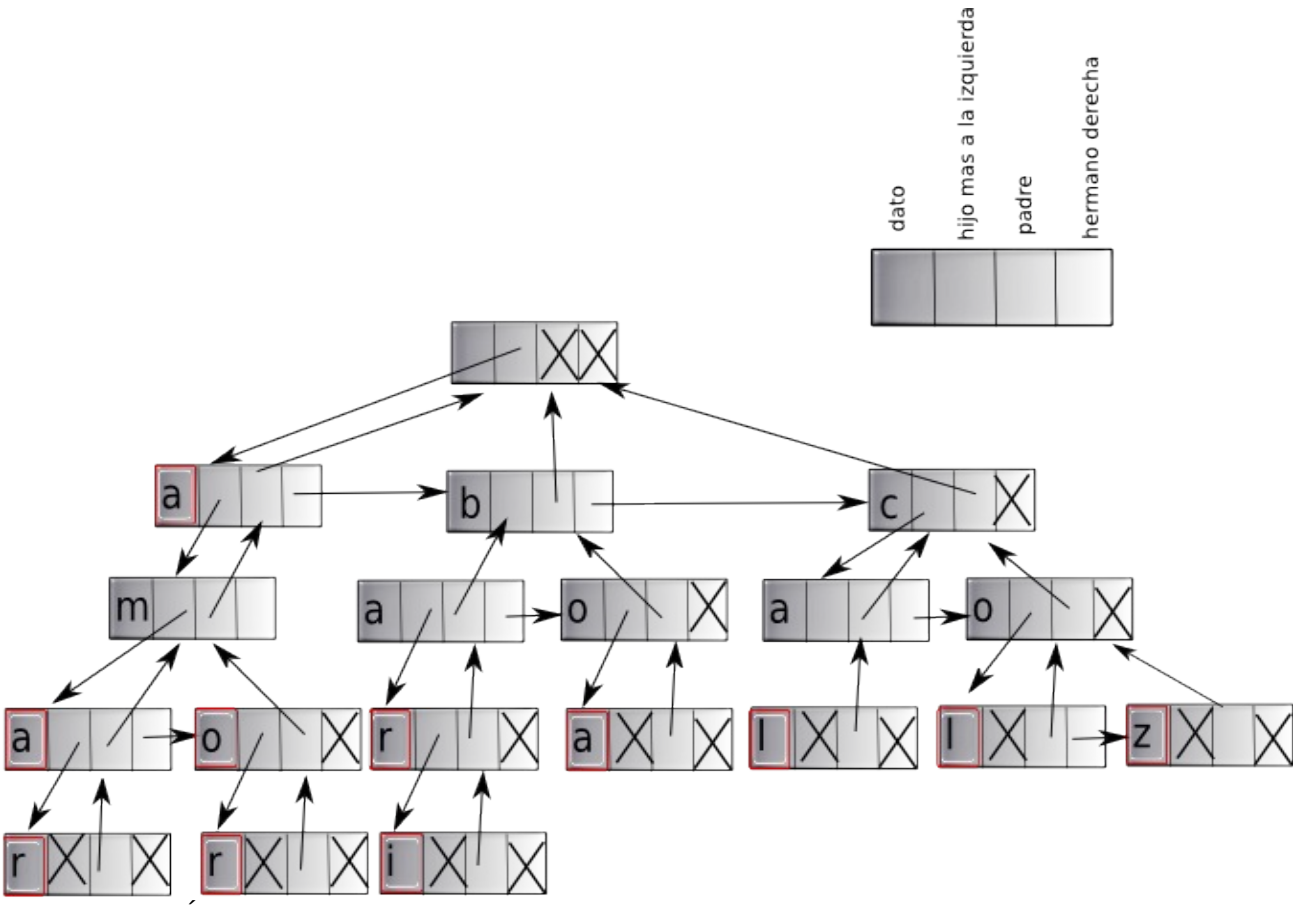


Figura 3: Árbol General para almacenar un conjunto de palabras. En cada nivel tenemos una letra de la palabra. Si la letra se encuentra encuadrada en rojo significa que desde el nivel 1 hasta ella se ha formado una palabra válida

Con respecto a las operaciones que se detallan, cabe destacar que se ha implementado un iterador, *iter_preorden,* para poder iterar por los nodos del ArbolGeneral siguiendo el recorrido 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();

       Nodo GetNodo ();

        bool Hoja()

        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.

El campo level del iter_preorden indica el nivel por el que va el iterador. Por convenio level se inicializa a -1 en begin (que apunta al nodo raiz).

5.2. Módulo Ontologías

El alumno debe implementar los métodos del T.D.A Ontologias. Un objeto del T.D.A. Ontologias representa un conjunto de palabras clasificadas por temáticas. El T.D.A Ontologías tendrá como parte de su representación un ArbolGeneral que almacene la clasificación de las palabras por temáticas y además para cada palabra final en una temática almacena su significado. El fichero Ontologías será el siguiente

```
#ifndef __ONTOLOGIAS__H
#define __ONTOLOGIAS__H

#include "ArbolGeneral.h"
….

class Ontologias{
  private:
    ArbolGeneral<pair<set<string>,int> >ab; //almacena la estructura de palabras por tematica
                                            //el entero contiene la clave de significados
                                            //de la palabra final o -1 en caso de
                                            //que no sea palabra final.
    map<int, string> significados; //almacena un código de significado de una palabra y
                                   //su significado
    int n_palabras; //numero de palabras finales
  public:
```

```
    /**
     * @brief Constructor por defecto.
     **/
    Ontologias(){
       n_palabras=0;
    }
    /** @brief devuelve el numero de palabras finales.
     */
    int size()const{ return n_palabras;}

    //declaración adelanta de los iteradores
    class iterator;
    class const_iterator;


    /**
     * @brief Devuelve si una palabra  esta en el conjunto. Si esta devuelve un iterador a el
     * @param o: palabra a buscar
     * @return una pareja que contiene si la palabra esta y en caso afirmativo un iterador a el
     *         en otro caso devuelve end y false
    */
    pair<bool, const_iterator> Esta(const string &o)const;

    /**
     * @brief Elimina todos las ontologias
     */
    void clear();

    /**
     * @brief Devuelve los significados asociados a todos los sinónimos a una palabra y la misma
       palabra
     * @param palabra: palabra a buscar
     * @return una coleccion de   pares significados y palabras sinónimas a la palabra introducida
     * por el usuario y la misma palabra que tiene ese significado clave
    */
    map<string,set<string> > GetSinonimos(const string & palabra)const;

    /**
     * @brief Devuelve  todas las temáticas (con sus sinónimos) hasta la temática raíz de una
               palabra con un significado concreto.
     * @param palabra: palabra a buscar
     * @param signi: significado de la palabra
     * @return una lista con todas las temáticas a las que pertenece  una palabras con un
               significado
     * @example:  Preguntamos por la palabra  madrid con significado spanish capital. El método
       devuelve:national capital,;capital,;seat,;eye,heart,middle,centre,center,;
       country,area,;region,;location,;physical object,object,;physical entity,;entity,;
        Si la jerarquía de madrid con ese significado era :
        spanish capital,capital of spain,madrid,;nationalcapital,;capital,;seat,;
       eye,heart,middle,centre,center,;country,area,;region,;location,;physical
       object,object,;physical entity,;entity,;
     * @exception: devuvelve una lista vacía sin no tiene es una palabra que existe en la
       ontologias     o palabra no tiene ninguna superpalabra.
    */
    list<set<string> > GetSuperPalabra(const string & palabra,const string & signifi)const;
```

```
    /**
     * @brief Obtiene el significado de una posición determinada
     */
    string GetDefinicion(int pos);

    /**
    @brief Lectura de los Significados de un fichero
    @param fich_sig: nombre del fichero con los significados
    El fichero contiene por cada linea índice y significado. El índice
    se corresponde con el índice del significado de una palabra final.
    */
    bool lee_significados(const char * fich_sig);

   /**
     * @brief Lee el las palabras descritas en forma jerárquica en temáticas y sus significados.
     * @param fich_jerarquia: nombre del fichero que almacena las palabras dispuestas en forma
        jerárquica
     * @param fich_significados: nombre del fichero con los significados.
     * @return true si el proceso de lectura es correcto. False en caso contrario
    */
    bool Lee(const char * fich_jerarquia,cons char * fic_significados);

    /**
     * @brief Escribe las ontologías en disco
     * @param fich_jerarquia: nombre del fichero donde se almacenará las palabras en forma
        jerarquica
     * @param fich_significados: nombre del fichero que almacenará  los significados.
    */
    bool Escribe(const char * fich_jerarquia,cons char * fic_significados);

    /**
     * @brief Numero de palabras finales.
     */
    int Numero_Nodos()const{ return ab.size();}

    /****************************************/
    /**
     * @brief iterator_level es un iterador que itera por las temáticas o palabras finales
     * de un determinado nivel
     */
    class iterator_level{
    private:
      ArbolGeneral<pair<set<string>,int> >::iter_preorden it;
      int level;
    public:
      /**
       * @brief constructor
       */
      iterator_level(){ level=-1;}
      /**
       * @brief  Consulta/modificador la información del iterador
       * @note el primer elemento del par es el conjunto de palabras sinonimas en dicho nodo
       * y el segundo elemento del par es el índice del significado
       * Para acceder al significado desde este indice se puede usar GetDefinicion
       */
      pair<set<string>,int> & operator *(){
```

```
        return *it;
      }

      /**
       * @brief Ver si dos interadores de nivel son iguales.
       */
      bool operator==(const iterator_level &i)const {
        return i.it==it;
      }

      /**
       * @brief Ver si dos interadores de nivel son diferentes.
       */
      bool operator!=(const iterator_level &i)const{
        return !(i.it==it);
      }
      /**
       *@brief avanza a la  siguiente temática o palabra final del mismo nivel
       *@return un iterador a la siguiente temática o palabra final.
      */
      iterator_level &operator ++(){
       do{
        ++it;

         if (it.getlevel()==-1) {

           return *this;
         }
         else if (it.getlevel()==level-1 && !it.Hoja()) {
                  return *this;
         }
        }while (it.getlevel()!=-1 );
        return *this;
      }

     friend class Ontologias;
     friend class const_iterator_level;
     friend class iterator;
    };

    //versión constante
    class const_iterator_level{
    private:
      ArbolGeneral<pair<set<string>,int> >::const_iter_preorden it;
      int level;
    public:
      const_iterator_level(){
        level=-1;
      }
      const_iterator_level(const iterator_level  &i):it(i.it){

      }
       const pair<set<string>,int> & operator *(){
          return *it;
      }
```

```
  bool operator==(const const_iterator_level &i)const{
    return i.it==it;
  }
  bool operator!=(const const_iterator_level &i)const{
    return !(i.it==it);
  }

  const_iterator_level &operator ++(){
   do{
    ++it;

     if (it.getlevel()==-1) {

       return *this;
     }
    else if (it.getlevel()==level-1 && !it.Hoja()) {
              return *this;
     }
    }while (it.getlevel()!=-1 );
    return *this;
  }
 friend class Ontologias;

};


/****************************************/
/**
 * @brief Itera por las palabras finales.
 */
class iterator{
private:
  ArbolGeneral<pair<set<string>,int> >::iter_preorden it;

public:
  /**
   * @brief Constructor
   */
  iterator(){

  }
  /**
   * @brief Constructor a partir de un iterador por niveles.
   */
  iterator(iterator_level & i){
    it=i.it;
    if (!it.Hoja()) ++(*this);

  }
  /**
   * @brief Consulta/Modificador del elemento que apunta el iterador
   */
  pair<set<string>,int> & operator *(){
      return *it;
```

```
}


/**
 * @brief Obtiene las temáticas a las que pertenece la palabra final que apunta
 * el iterador
 * @return una lista conteniendo todas las temáticas (con sus sinónimos). Así
 * cada elemento de la lista contiene una temática. Esta temática se almacena
 * en un conjunto (set<string>) donde se mantiene todos los sinónimos de esa temática incluida
 ella misma
 */
list<set<string> >  GetJerarquia(){
 list<set<string> > laux;
 ArbolGeneral<pair<set<string>,int> >::iter_preorden auxit=it;
 do {
   laux.push_front((*auxit).first);
   auxit=auxit.padre();

 }while (!auxit.Nulo());
 return laux;
}


/**
 * @brief Obtiene la temáticas justo en el nivel superior (la mas especializada).
 * @return un conjunto con los sinonimos a la temática a la que pertenece la palabra
 * final apuntada por el iterador (temática más especializada).
 */
set<string> GetSuperPalabra(){
  ArbolGeneral<pair<set<string>,int> >::iter_preorden aux= it.padre();
  if (!aux.Nulo()){
     return (*aux).first;
  }
  else return set<string>();

}
/**
 * @brief Dos iteradores son iguales
 */

bool operator==(const iterator &i)const {
  return i.it==it;

}
/**
 * @brief Dos iteradores son diferentes
 */
bool operator!=(const iterator &i)const{
  return !(i.it==it);
}
/**
* @brief Avanza a la siguiente palabra final
**/

iterator &operator ++(){
 do{
```

```
      ++it;
       if (it.getlevel()==-1) {

         return *this;
       }

      }while (it.getlevel()!=-1 && !it.Hoja());
      return *this;
    }
   friend class Ontologias;
   friend class const_iterator;
  };

  //version constante
  class const_iterator{
  private:
    ArbolGeneral<pair<set<string>,int> >::const_iter_preorden it;
    int level;
  public:
    const_iterator(){

    }
    const_iterator(const iterator  &i):it(i.it){

    }
     const pair<set<string>,int> & operator *(){
        return *it;
    }

   list<set<string> >  GetJerarquia(){
     list<set<string> > laux;
     ArbolGeneral<pair<set<string>,int> >::const_iter_preorden auxit=it;
     do {
       laux.push_front((*auxit).first);
       auxit=auxit.padre();

     }while (!auxit.Nulo());
     return laux;
    }

    set<string> GetSuperPalabra(){
      ArbolGeneral<pair<set<string>,int> >::const_iter_preorden aux= it.padre();
      if (!aux.Nulo()){
         return (*aux).first;
      }
      else return set<string>();

    }

    bool operator==(const const_iterator &i)const{
      return i.it==it;
    }
    bool operator!=(const const_iterator &i)const{
      return !(i.it==it);
    }
```

```
  const_iterator &operator ++(){
   do{
    ++it;

     if (it.getlevel()==-1) {

       return *this;
     }

    }while (it.getlevel()!=-1 && !it.Hoja());
    return *this;
  }
 friend class Ontologias;

};

/***************BEGIN y END*************/

/**
 * @brief Inicia un iterador a la primera palabra final.
 * */
iterator begin(){
  iterator i;
  i.it = ab.begin();
  ++i;
  return i;
}
/**
 @brief Inicia un iterador al final de las palabras finales (no apunta
 a una palabra valida).
 */
iterator end(){
  iterator i;
  i.it = ab.end();
  return i;
}
// versiones constantes
const_iterator begin()const{
  const_iterator i;
  i.it = ab.begin();
  ++i;
  return i;
}

const_iterator end()const{
  const_iterator i;
  i.it = ab.end();
  return i;
}

/*
 * @brief Inicia un iterador por niveles a un determiando nivel
 */
iterator_level beginl(int l){
```

```
      iterator_level i;
      i.it = ab.begin();
      i.level= l;
      ++i;
      return i;
    }
    /*
     * @brief Inicia un iterador por niveles al final de un nivel de tematicas
     *  La posicion ya no es valida
     */

    iterator_level endl(){
      iterator_level i;
      i.it = ab.end();

      return i;
    }
    /* versiones constantes
     */

    const_iterator_level cbeginl(int l)const{
      const_iterator_level i;
      i.it = ab.begin();
      i.level= l;
      ++i;
      return i;
    }

    const_iterator_level cendl()const{
      const_iterator_level i;

      i.it = ab.end();
      return i;
    }

};
#endif
```

En el código de Ontologías se puede observar que se han definido dos tipos de iteradores (cada uno con sus versiones constates). Así tenemos **iterator** y **iterator_level:**

- **iterator:** itera por las palabras finales.
- **iterator_level:** itera por las temáticas o palabras finales asociadas a un nivel.

Asociados a estos iteradores disponemos en la clase Ontologias de las funciones begin y end. El alumno puede observar que para iniciar un iterator_level debe indicar el nivel (nivel de la temática) sobre la que se quiere iterar.

5.3. Módulo PreguntasED

Este modulo permite obtener sobre el conjunto de palabras que se ocultarán en la sopa de letras. Para ello a través de este módulo se mantendrá la información del conjunto de palabras y definiciones escogidas dentro del conjunto de ontologías para crear la sopa de letras. La interfaz y representación que deberá usar el alumno/a es la que se muestra a continuación:

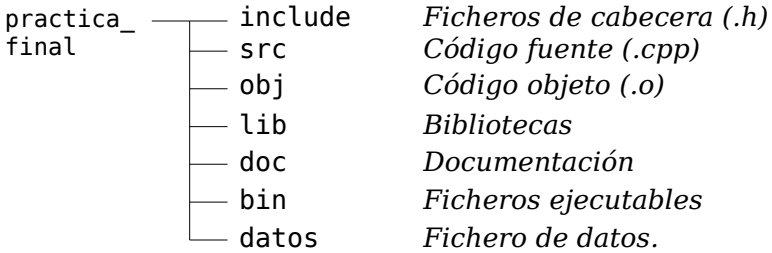
```
#ifndef __PREGUNTAS_ED_H
#define __PREGUNTAS_ED_H
#include "Ontologias.h"
…
class PreguntasED{
  private:
    int tema_level;//nivel de la tematica escogida
    Ontologias Ot; //conjunto de ontologias
    set<int>preguntas_hechas; //de entre las preguntas escogidas las realizadas
    int tematica_escogida; //tematica escogida entre todas del mismo nivel.
    vector<Ontologias::iterator_level>temas; // los temas en dicho nivel
    vector<Ontologias::iterator> preguntas_tema; //preguntas asociadas a la tematica escogida.
    int next_pregunta=-1; // la siguiente pregunta a realizar

    /**
     * @brief Obtiene todas las tematicas de un nivel fijado
     * @note Este metodo inicia el miembro temas con iteradores apuntando a dichas tematicas
     * Ver iterator_level de Ontologias
     */
     void CreaTematica();

  public:
    /**
     * @brief Constructor
     * @param O: conjunto de ontologias para establecer las preguntas
     * @param tl: nivel de tematica sobre la que construir las preguntas
     */
    PreguntasED(const Ontologias &O, int tl=4);

    /**
     * @brief Muestra todas las tematicas en el nivel
     *
     * */
    void MuestraTematicas();

    /**
     @brief Modifica la tematica escogida a un valor
     @param i: valor de tematica escogida
     */
    void SetTematica(int i) ;

    /**
     *@brief Obtiene el titulo de la tematica (el nombre)
     * @return el nombre de la tematica
     */
    string GetTitleTematica();

    /**
     *@brief Almacena todos las definiciones de las temática escogida
     *@post tras la ejecución el miembro preguntas_tema tiene  las preguntas (significados de las
        palabras finales)
     *de la temática escogida. Además next_pregunta a 0 tienen un valor 0
     *
     */
```

```
    void IniciaConceptosTemaEscogido();

   /**
     *@brief Baraja todas las preguntas de forma aleatoria.
     *
     **/
    void BarajarPreguntas();

    /**
     *@brief Devuelve la siguiente pregunta a realizar
     *@post next_pregunta se modifica a la siguiente pregunta.
     */
    pair<set<string>,string> SacaPregunta();

    /**
     * @brief Devuelve el número de preguntas en la temática
     */
    int num_preguntas()const{ return preguntas_tema.size();}

    /**
     *@brief Obtiene una pregutna
     *@param i: indice de la pregunta a devolver
     *@return un par  con el conjunto de palabras sinónimos que responden a esa pregunta y la
       definición.
     */
    pair<set<string>,string> GetPregunta(int i);

};
#endif
```

6. Práctica a entregar

El alumno deberá empaquetar todos los archivos relacionados en el proyecto en un archivo con nombre "practica_final.*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 "*practica_final*") para ejecutar:

```
prompt% tar zcv practica_final.tgz practica_final
```

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