CorrectorOrtografico

Generado por Doxygen 1.7.5

Miércoles, 9 de Octubre de 2013 14:57:56

Índice general

Capítulo 1

Diccionario

Versión

v0

Autor

Juan F. Huete

1.1. Introducción

En esta practica se pretende avanzar en el uso de las estructuras de datos, para ello comenzaremos con un tipo de datos simplificado que llamaremos diccionario.

Un diccionario es un contenedor que permite almacenar un conjunto de pares de elementos, el primero será la clave que deber ser única y el segundo la definición. En nuestro caso el diccionario va a tener un subconjunto restringido de métodos (inserción de elementos, consulta de un elemento por clave, además de la consulta del elemento con mayor valor en la definición). Este diccionario "simulará" un diccionario de la stl, con algunas claras diferencias pue, entre otros, no estará dotado de la capacidad de iterar (recorrer) a través de sus elementos.

Asociado al diccionario, tendremos los tipos tipos

```
 diccionario::entrada
diccionario::size_type
```

que permiten hacer referencia al par de elementos almacenados en cada una de las posiciones del diccionario y los elementos del mismo, respectivamente. El primer campo de una entrada, first, representa la clave y el segundo campo, second, representa la definición. En nuestra aplicación concreta, la clave será un string representando una palabra válida del diccionario y el segundo campo es un entero que hace referencia a la frecuencia de ocurrencia de la palabra en el lenguaje.

1.2. Ejemplo de Uso:

```
#include "diccionario.h"

diccionario dic;
diccionario::entrada dato;
....


dic["hola"] = 5; // inserta el par ("hola",5) en el diccionario
dic["pero"];  // inserta el par ("pero",0) en el diccionario
dato = dic.find("hola");  // nos devuelve la entrada hola en el diccionario.


cout << "El valor mas frecuente es " << dic.max_element() << endl;

cout << dic << endl; // muestra todos los elementos en el diccionario.
```

1.3. ¿Cómo implementar un corrector ortográfico?.

Algunas veces te habrás plateado cómo Goolgle (Yahoo o Microsoft) realizan la corrección ortográfica. En general, se basan en tecnologías que utilizan estadísticas del lenguaje, cuyos detalles pueden ser complejos. En esta práctica utilizaremos una aproximación más simple, con la capacidad de corregir en torno al 80 % o 90 % de las palabras que se escriban mal (eso si, dependiendo de la base de datos de la que se estimen las estadísticas).

La idea es simple, por un lado tenemos una base de datos (en esta práctica utilizaremos de nuevo El Quijote como fichero de datos, aunque cualquier otro fichero puede ser válida). Procesaremos el fichero y, para cada palabra contaremos la frecuencia de aparición de la misma en el fichero, que se almacenarán en una estructura de datos diccionario. Obviamente, consideramos que sólo las palabras que están en el diccionario serán palabras válidas.

Por otro lado, tendremos la palabra que queremos corregir. Por ejemplo, "hola". En este caso, como la palabra está en el fichero, la consideraremos como una palabra válida. Pero, ¿Qué pasa si la escribo mal?. Supongamos que escribo "hoal". Aqui lo más probable es que exista una "transposición" de letras y la palabra correcta sea "hola", pero y si escribo "holo". Ahora no es tan fácil, pued podemos tener otras alternativas como "hilo", "hola", "halo", "hoyo", "solo", etc. ¿Cuál de estas sugerimos? La respuesta es simple, la más frecuente en el lenguaje, en nuestro caso en El Quijote, "solo" que aparece 10111 veces frente a las 314 ocurrencias de "hoyo", las 198 de "hilo", las 42 de "hola" o las 31 de "halo".

Para resolver en problema utilizaremos dos diccionarios, el primero para almacener todas las palabras correctas en el lenguaje con su frecuencia de aparición y el segundo en el que se almacenan todas las palabras del primer diccionario que podrían ser una sugerencia válida. De entre todas estas sugeriremos la más probable.

La primera pregunta que nos podemos hacer es cómo se realizan las sugerencias. - Para ello se consideran modificaciones de la palabra original que se obtengan mediante un solo cambio, que puede ser de los siguientes tipos, consideraremos "holo" como palabra entrante:

- Borrado de un caracter: consideraremos como candidatas a "olo hlo hoo hol".
- Transposición de dos caracteres: obtenemos "ohlo hloo hool"
- Alteraciones en un caracter: por ejemplo la a o la d, "aolo halo hoao hola, dolo hdlo hodo" ...
- Inserciones de un caracter: por ejemplo la a o la t, "aholo haolo hoalo holao holoa tholo htolo hotlo holto holot" ...

Así podemos generar un total de 241 posibles resultados. Para cada uno de ellos miramos si está en el diccionario aprendido del Quijote. Si es cierto, (pertenece al Quijote) lo introducimos en un segundo diccionario donde se almancenan las palabras correctas que se pueden ofrecer como alternativa para la corrección. Finalmente, de entre todas las palabras candidatas se selecciona como sugerencia aquella que es más comun.

1.4. "Se Entrega / Se Pide"

1.4.1. Se entrega

En esta práctica se entrega los fuentes necesarios para generar la documentación de este proyecto así como el código necesario para resolver este problema. En concreto los ficheros que se entregan son:

- documentacion.pdf Documentación de la práctica en pdf.
- dox_diccionario Este fichero contiene el fichero de configuración de doxigen necesario para generar la documentación del proyecto (html y pdf). Para ello, basta con ejecutar desde la línea de comando

  ```
  doxygen dox_diccionario
  ```

  La documentación en html la podemos encontrar en el fichero ./html/index.html, para generar la documentación en latex es suficiente con hacer los siguientes pasos

  ```
  cd latex
  make
  ```

  como resultado tendremos el fichero refman.pdf que incluye toda la documentación generada.

- diccionario.h Especificación del TDA diccionario.
- diccionarioV1.hxx fichero donde debemos implementar la primera versión del diccionario.
- diccionarioV2.hxx fichero donde debemos implementar la segunda versión del diccionario.
- corrector.h clase corrector, que es la que se encarga de toda la lógica del algoritmo de corrección ortográfica.

- principal.cpp fichero donde se incluye el main del programa. En este caso, se toma como entrada el fichero de datos "quijote.txt" ya utilizado en la práctica anterior.

1.4.2. Se Pide

Se pide implementar el código asociado la tipo de dato diccionario considerando dos posibles representaciones basadas en el tipo de dato vector de la stl, analizando la eficiencia de las mismas. La primera implementación se entregará en un fichero denominado diccionarioV1.hxx y la segunda en un fichero denominado diccionarioV2.hxx

Para compilar con la primera implementación habrá que hacer

- g++ -D DICC_V1 -o correctorV1 principal.cpp

Para compilar con la segunda implementación se tendrá que utilizar

- g++ -D DICC_V2 -o correctorV2 principal.cpp

Por tanto, los alumnos deberán subir a la plataforma las dos implementaciones así como un análisis de la eficiencia de las mismas en los siguientes ficheros

- diccionarioV1.hxx
- diccionarioV2.hxx
- AnalisisCompartivo.pdf

1.5. Fecha Límite de Entrega: Lunes 28 de Octubre a las 23:59 horas.

1.6. Representaciones

El alumno deberá realizar dos implementaciones distintas del diccionario, utilizando como base el TDA vector de la STL, en la primera de ellas los elementos se almacenarán sin tener en cuenta el valor de la clave mientras que en la segunda debemos garantizar que los elementos se encuentran ordenados por dicho valor.

1.7. Primera Representación:

1.7.1. Función de Abstracción :

Función de Abstracción: AF: Rep => Abs

dado D=(vector<entrada> dic, int mayor) ==> Diccionario Dic;

Un objeto abstracto, Dic, representando una colección de pares (string,int) se instancia en la clase diccionario como un vector de entradas, definidas como diccionario::entrada. Dada una entrada, x, en D, x.first representa a una palabra válida (clave) y x.second representa el número de veces que ocurre x (definición).

D.dic[D.mayor] hace referencia a la entrada más frecuente en el diccionario.

1.7.2. Invariante de la Representación:

Propiedades que debe cumplir cualquier objeto

```
Dic.size() == D.dic.size();


Para todo i, 0 <= i < D.dic.size() se cumple
        D.dic[D.mayor] >= D.dic[i].second;
        D.dic[i].second > 1;
        D.dic[i].first != "";
        D.dic[i].fisrt != D.dic[j].first, para todo j!=i.
```

1.8. Segunda Representación:

En este caso, la representación que se utiliza es un vector ordenado de entradas, teniendo en cuenta el valor de la clave.

1.8.1. Función de Abstracción :

Función de Abstracción: AF: Rep => Abs

dado D=(vector<entrada> dic, int mayor) ==> Diccionario Dic;

Un objeto abstracto, Dic, representando una colección de pares (string,int) se instancia en el diccionario como un vector ordenado de entradas, diccionario::entrada. Dada una entrada, x, en D, x.first representa a una palabra válida (clave) y x.second representa el número de veces que ocurre x (definición).

D.dic[D.mayor] hace referencia a la entrada más frecuente en el diccionario.

1.8.2. Invariante de la Representación:

Propiedades que debe cumplir cualquier objeto

```
Dic.size() == D.dic.size();

Para todo i, 0 <= i < D.dic.size() se cumple
        D.dic[D.mayor].second >= D.dic[i].second;
        D.dic[i].second > 1;
```

```
        D.dic[i].first != "";pq.size() == pq.V.size();

Para todo i, 0 <= i < D.dic.size()-1 se cumple
        D.dic[i].first< D.dic[i+1].first
```

Capítulo 2

Lista de tareas pendientes

Miembro diccionario::cheq_rep () const

implementa esta función

implementa esta función

Miembro diccionario::diccionario ()

implementa esta función

implementa esta función

Miembro diccionario::diccionario (const entrada &nula)

implementa esta función

implementa esta función

Miembro diccionario::diccionario (const diccionario &d)

implementa esta función

implementa esta función

Miembro diccionario::empty () const

implementa esta función

implementa esta función

Miembro diccionario::find (const string &s) const

implementa esta función

implementa esta función

Miembro diccionario::max_element () const

implementa esta función

implementa esta función

Miembro diccionario::operator= (const diccionario &org)

implementa esta función

implementa esta función

Miembro diccionario::operator[] (const string &s)

implementa esta función

implementa esta función

Miembro diccionario::operator[] (const string &s) const

implementa esta función

implementa esta función

Capítulo 3

Índice de clases

3.1. Lista de clases

Lista de las clases, estructuras, uniones e interfaces con una breve descripción:

Capítulo 4

Documentación de las clases

4.1. Referencia de la Clase corrector

Métodos públicos

- std::string correct (const std::string &word)

  determina la palabra corregida
- void load (const std::string &filename)

  lectura del fichero de entrenamiento

Métodos privados

- void edits (const std::string &word, std::vector< std::string > &result)

  Genera todas las posibles modificaciones tras una "edicion" de la cadena word.
- void known (std::vector< std::string > &results, diccionario &candidates)

  busca ocurrencias en un diccionario

Atributos privados

- diccionario **dictionary**

4.1.1. Documentación de las funciones miembro

4.1.1.1. std::string corrector::correct (const std::string & *word*) [inline]

determina la palabra corregida

Parámetros

in	*word*	palabra origen

Devuelve

palabra que se sugiere como correccion.

4.1.1.2. void corrector::edits (const std::string & *word,* std::vector< std::string > & *result*) [inline, private]

Genera todas las posibles modificaciones tras una "edicion" de la cadena word.

Parámetros

in	*word*	cadena de entrada
out	*result*	vector con las posibles palabras que se obtienen al realizar borrados, transposiciones, alteraciones o inserciones en la cadena word.

4.1.1.3. void corrector::known (std::vector< std::string > & *results,* diccionario & *candidates*) [inline, private]

busca ocurrencias en un diccionario

Parámetros

in	*results*	conjunto de palabras a buscar
in,out	*candidates*	conjunto de palabras en results cuyas entradas tambien se encuentra en el diccionario

4.1.1.4. void corrector::load (const std::string & *filename*) [inline]

lectura del fichero de entrenamiento

Parámetros

in	*filename*	nombre del fichero a leer

La documentación para esta clase fue generada a partir del siguiente fichero:

- corrector.h

4.2. Referencia de la Clase diccionario

Clase diccionario.

`#include <diccionario.h>`

Tipos públicos

- typedef pair< string, int > **entrada**
- typedef unsigned int **size_type**

Métodos públicos

- diccionario ()

  constructor primitivo.
- diccionario (const entrada &nula)

  constructor primitivo.
- diccionario (const diccionario &d)

  constructor de copia
- bool empty () const

  vacia Chequea si el priority_queue esta vacio (size()==0)
- const entrada & find (const string &s) const

  busca una cadena en el diccionario
- const string & max_element () const

  devuelve una referencia constante a la entrada con un mayor numero de ocurrencias en el diccionario.
- const entrada & null () const

  entrada nula del diccionario
- diccionario & operator= (const diccionario &org)

  operador de asignación
- int & operator[] (const string &s)

  Consulta/Inserta una entrada en el diccionario.
- const int & operator[] (const string &s) const

  Consulta una entrada en el diccionario.
- size_type size () const

  numero de entradas en el diccionario

Métodos privados

- bool cheq_rep () const

  Chequea el Invariante de la representacion.

Atributos privados

- vector< entrada > **dic**
- int **pos_max**

Amigas

- ostream & operator<< (ostream &, const diccionario &)

  imprime todas las entradas del diccionario

4.2.1. Descripción detallada

Clase diccionario.

diccionario:: diccionario, end, find, operator[], size, max_element Tipos diccionario-::entrada, diccionario::size_type Descripción

Un diccionario es un contenedor que permite almacenar un conjunto de pares de elementos, el primero será la clave que deber ser única y el segundo la definición. En nuestro caso el diccionario va a tener un subconjunto restringido de métodos (inserción de elementos, consulta de un elemento por clave, además de la consulta del elemento con mayor valor en la definición). Este diccionario "simulará" un diccionario de la stl, con algunas claras diferencias pue, entre otros, no estará dotado de la capacidad de iterar (recorrer) a través de sus elementos.

Asociado al diccionario, tendremos el tipo

```
diccionario::entrada
```

que permite hacer referencia al par de elementos almacenados en cada una de las posiciones del diccionario. Así, el primer campo de una entrada, first, representa la clave y el segundo campo, second, representa la definición. En nuestra aplicación concreta, la clave será un string representando una palabra válida del diccionario y el segundo campo es un entero que hace referencia a la frecuencia de ocurrencia de la palabra en el lenguaje.

El número de elementos en el diccionario puede variar dinámicamente; la gestión de la memoria es automática.

4.2.2. Documentación del constructor y destructor

4.2.2.1. diccionario::diccionario ( )

constructor primitivo.

Postcondición

define la entrada nula como el par ("",-1)

Tareas pendientes implementa esta función

Tareas pendientes implementa esta función

4.2.2.2. diccionario::diccionario (const entrada & *nula*)

constructor primitivo.

Parámetros

in	*nula*	representa a la entrada nula para el diccionario

Postcondición

define la entrada nula

Tareas pendientes implementa esta función

Tareas pendientes implementa esta función

4.2.2.3. diccionario::diccionario (const **diccionario** & *d*)

constructor de copia

Parámetros

in	*d*	diccionario a copiar

Tareas pendientes implementa esta función

Tareas pendientes implementa esta función

4.2.3. Documentación de las funciones miembro

4.2.3.1. bool diccionario::cheq_rep ( ) const [private]

Chequea el Invariante de la representacion.

Devuelve

true si el invariante es correcto, falso en caso contrario

Tareas pendientes implementa esta función

Tareas pendientes implementa esta función

4.2.3.2. bool diccionario::empty ( ) const

vacia Chequea si el priority_queue esta vacio (size()==0)

Tareas pendientes implementa esta función

Tareas pendientes implementa esta función

4.2.3.3. const diccionario::entrada & diccionario::find (const string & *s*) const

busca una cadena en el diccionario

Parámetros

s	cadena a buscar

Devuelve

una copia de la entrada en el diccionario. Si la palabra s no se encuentra devuelve end()

Postcondición

no modifica el diccionario.

```
Uso
if (D.fin("hola")!=D.end()) cout << "Esta" ;
else cout << "No esta";
```

Tareas pendientes implementa esta función

Tareas pendientes implementa esta función

4.2.3.4. const string & diccionario::max_element ( ) const

devuelve una referencia constante a la entrada con un mayor numero de ocurrencias en el diccionario.

Postcondición

No se modifica el diccionario.

Tareas pendientes implementa esta función

Tareas pendientes implementa esta función

4.2.3.5. const diccionario::entrada & diccionario::null ( ) const

entrada nula del diccionario

Devuelve

Devuelve la entrada nula del diccionario.

Postcondición

no modifica el diccionario

4.2.3.6. diccionario & diccionario::operator= (const diccionario & *org*)

operador de asignación

Parámetros

in	*org*	diccionario a copiar. Crea un diccionario duplicado exacto de org.

Tareas pendientes implementa esta función

Tareas pendientes implementa esta función

4.2.3.7. int & diccionario::operator[ ] (const string & *s*)

Consulta/Inserta una entrada en el diccionario.

Busca la cadena s en el diccionario, si la encuentra devuelve una referencia al numero de ocurrencias de la misma en caso contrario la inserta, con frecuencia cero, devolviendo una referencia a este valor.

Parámetros

in	*s*	cadena a insertar
out	*int*	& referencia a la definicion asociada a la entrada

Postcondición

Si s no esta en el diccionario, el size() sera incrementado en 1.

Tareas pendientes implementa esta función

Tareas pendientes implementa esta función

4.2.3.8. const int & diccionario::operator[ ] (const string & *s*) const

Consulta una entrada en el diccionario.

Busca la cadena s en el diccionario, si la encuentra devuelve una referencia constante al numero de ocurrencias de la misma, si no la encuentra da un mensaje de error.

Parámetros

in	*s*	cadena a insertar
out	*int*	& referencia constante a la definicion asociada a la entrada

Postcondición

No se modifica el diccionario.

Tareas pendientes implementa esta función

Tareas pendientes implementa esta función

4.2.3.9. diccionario::size_type diccionario::size ( ) const

numero de entradas en el diccionario

Postcondición

No se modifica el diccionario.

4.2.4. Documentación de las funciones relacionadas y clases amigas

4.2.4.1. ostream& operator<< (ostream & *sal,* const diccionario & *D*) [friend]

imprime todas las entradas del diccionario

Postcondición

No se modifica el diccionario.

Tareas pendientes implementar esta funcion

Tareas pendientes implementa esta función

Tareas pendientes implementa esta función

La documentación para esta clase fue generada a partir de los siguientes ficheros:

- diccionario.h
- diccionarioV1.hxx
- diccionarioV2.hxx