My Project

Generated by Doxygen 1.8.9.1

Tue Dec 22 2015 13:28:29

Contents

Chapter 1

City of Chicago Crime DataSet

Version

v0

Author

Juan F. Huete

1.1 NOTA IMPORTANTE

Esta práctica es individual, por lo que el alumno debe incluir una nota en la misma indicando que no ha utilizado material de otros compañeros o compañeras para su resolución.

1.2 Introducción

La ciudad de Chicago, en un intento de hacer un más flexible el acceso a los datos sobre delitos cometidos ha decidido replantearse un nuevo diseño del sistema de información que encargó a los alumnos en prácticas de la Universidad. Además de flexibilizar el acceso a los datos, otra de los requisitos que impone en el pliego de condiciones es que el acceso sea lo más eficiente posible. Este es un aspecto importante para el departamento ya que se esperan un alto número de consultas al sistema y este debe ser capaz de gestionarlas.

Se debe diseñar un nuevo tipo de dato que llamaremos CSS, acrónimo de Crime Search System, que tiene que cumplir los siguientes requisitos:

- Permitir un acceso eficiente a los delitos por ID.
- Permitir consultas por rango de fechas, por ejemplo se puede querer encontrar los delitos cometidos entre el 10 de Febrero de 2015 y el 30 de Marzo de 2015.
- Consultar por IUCR (Illinois Unifor Crime Reporting), código del crimen del estado de Illinois que permite representar la tipología del delito. Este es un campo de tipo String, para el que es importante considerar el orden entre los elementos. El orden que se considerará para la comparación de IUCR será el lexicográfico. Así, por ejemplo, todos los crímenes comprendidos con código en el rango [261,300] representan “CRIM SEXUAL ASSAULT”, la diferencia radica en la descripción secundaria del delito. Por ejemplo
- IUCR 261 = “CRIM SEXUAL ASSAULT”; AGGRAVATED: HANDGUN
- IUCR 266 = “CRIM SEXUAL ASSAULT”; PREDATORY

Una lista de la descripción del IUCR la podemos encontrar en

https://data.cityofchicago.org/Public-Safety/Chicago-Police-Department-Illinois-Uniform-

- La novedad más importante será que se permite la consulta considerando la descripción del crimen, así un ciudadano podrá consultar por términos aislados en la descripción, por ejemplo podría consultar por "BA↩ TTERY ASSAULT HANDGUN" y tendría como salida una secuencia ordenada de los delitos que contengan al menos una de estas palabras, de forma que cuanto mayor sea el número de palabras que aparecen en la descripción, mejor será la posición del delito en el orden (ocupará las primeras posiciones del mismo). Con la idea de facilitar las tareas de búsqueda, el usuario podrá indicar cuantos delitos desea que le sean retornados. Obviamente, el acceso a los datos se debe hacer lo más rápidamente posible pues el éxito del sistema depende en gran medida de esta consulta con texto libre que se pondrá a disposición del ciudadano. Para conseguir una mayor flexibilidad y potencia de las búsquedas, el departamento ha decidido que las búsquedas por términos tengan en cuenta los siguientes campos de la base de datos: Primary Type, Description y Location Description.

- Se debe poder acceder de forma eficiente a los crímenes que han ocurrido en una determinada posición geográfica, determinada por un cuadrilátero que viene determinados por las coordenadas de longitud y latitud de los vértices superior izquierda e inferior derecha del mismo.

- Dado el elevado número de delitos en la base de datos (más de 6 millones), se debe evitar duplicar la información de la base de datos, esto es, NO podemos crear distintos tipos de datos donde en cada uno de ellos se almacene un crimen completo, sino que la descripción completa del crimen se almacenará una única vez y habrá estructuras adicionales que nos faciliten alcanzar los objetivos planteados.

- Se debe dotar de la capacidad de iterar sobre los crímenes, tanto por ID como por IUCR y fecha.

- Finalmente, se desea poder disponer del sistema funcionando antes de Navidad del 2015, por lo que deberá estar entregada la solución el día 21 de diciembre de 2015 a las 23:59

1.3 Tipo de dato CrimeSearch

La empresa de nueva creación EDGRX, ubicada en Granada, decide presentar su propuesta al departamento de policía. Para ello diseñan el tipo de dato CSS (CrimeSearchSystem),

1.3.1 Parte Pública

En la parte pública podemos distinguir tres tipos de datos:

- iterator que nos permite iterar por los delitos en orden creciente de ID
- Date_iterator que nos permite iterar por los delitos en orden creciente de fecha, del más antiguo al más reciente.
- IUCR_iterator que nos permite iterar según el valor del IUCR

Además, y entre otros métodos que sean de interés, se ofrecen los siguientes

```
void load(String nombreDB);
void insert( const crimen & x);
void erase( unsigned int ID);
iterator find_ID(const unsigned int ID);
void setArrest(const unsigned int ID, bool value);

vector<pair<ID,float> Query(list<string & q, int k);
list<ID> inArea(Longitud x1, Latitud y1, Longitud x2, Latitud y2 );

/* Métodos relacionados con los iteradores */
IUCR_iterator ibegin();
```

```
IUCR_iterator iend();
IUCR_iterator lower_bound(IUCR);
IUCR_iterator upper_bound(IUCR);

Date_iterator dbegin();
Date_iterator dend();
Date_iterator lower_bound(Fecha);
Date_iterator upper_bound(Fecha);

iterator begin();
iterator end();
```

Los pasamos a ver con más detenimiento

- void load(String nombreDB);

Se encarga de leer los elementos de un fichero dado por el argumento nombreDB, e insertar toda la información en la base de datos. Recordad que para esta práctica se os pide que extendáis el campo descripción a todas las descripciones que se encuentra en la base de datos. Pare ello será necesario modificar la clase crimen utilizada en la práctica anterior.

- void insert(const crimen & x);

Este método se encarga de insertar un nuevo crimen en CSS. Como prerrequisito se asume que el crimen no está ya almacenado en el conjunto. s

- bool erase(unsigned int ID);

En este caso, se trata de borrar un crimen de la base de datos dado su ID. Devuelve verdadero si el crimen ha sido borrado correctamente, falso en caso contrario.

No sólo borra el crimen del repositorio principal de datos en CSS sino que además se encarga de borrar toda referencia a dicho crimen dentro de él.

- iterator find (const unsigned int ID);

Busca el crimen con identificador ID dentro de CSS, si no lo encuentra devuelve end()

- void setArrest(const unsigned int ID, bool value);

Modifica el campo arrest de un crimen identificado por ID. Será necesario cuando se detenga un criminal con posterioridad a la inserción del delito en CSS.

- vector<pair<ID,float> > Query(list<string> & q, int k); Dada una consulta, expresada mediante un conjunto de términos en q, el sistema devuelve un vector ordenado con los k mejores ID (aquellos con mayor peso, definido como el número de términos de q que están en la descripción del delito), con sus respectivos pesos.

Cómo implementar esta función se explica en la sección “Consulta Libre” de este documento.

- list<ID> inArea(Longitud x1, Latitud y1, Longitud x2, Latitud y2);

Dada dos coordenadas geográficas, x e y, donde se asume que x1 $<$ x2 e y1 $>$ y2, nos devuelve la lista de ID que representan delitos que han sido cometidos dentro del área geográfica delimitada por las coordenadas.

- IUCR_iterator lower_bound(IUCR i);
- IUCR_iterator upper_bound(IUCR d);
- Date_iterator lower_bound(Fecha i);

- Date_iterator upper_bound(Fecha d);

Son métodos que permiten hacer la búsqueda por rango tanto considerando el IUCR como la Fecha del delito. El comportamiento es similar, lower_bound devuelve el iterador que apunta primer delito con IUCR(Fecha) mayor o igual a i, mientras que upper_bound devuelve el primer delito con IUCR(fecha) estrictamente mayor que d.

- IUCR_iterator ibegin();
- Date_iterator dbegin();
- iterator begin();

Devuelve el iterador correspondiente al primer delito que se encuentra según el criterio que sobre el que se itera.

- IUCR_iterator iend();
- Date_iterator dend();
- iterator end();

Devuelve el iterador que apunta al elemento siguiente al último delito en CSS según el criterio sobre el que se está itera.

1.3.2 Parte privada

En este caso hablaremos de los atributos que se han escogido para representar la información.

```
typedef float Longitud;
typedef float Latitud;
typedef unsigned int ID;
typedef string Termino;
typedef string IUCR;

class CSS{
private:
    map<ID,Crimen> baseDatos;

    multimap<Date, map<ID,Crimen>::iterator > > DateAccess;

    map<IUCR,set<ID> > IUCRAccess;

    unordered_map<Termino, set<ID> > index;

    map<Longitud,multimap<latitud, ID> > posicionGeo;

};
```

Pasamos a detallar cada uno de ellos

- map<ID,Crimen> baseDatos;

Los distintos delitos se almacenan por orden creciente de ID en un diccionario (map), que llamaremos baseDatos, donde la clave será el ID (ya que asumimos que no hay dos delitos con el mismo valor de ID), y en la descripción tenemos almacenada toda la información relativa al crimen.

Es importante destacar que la inserción y borrado de elementos en el map no invalida los iteradores. Esto nos facilitará las labores de implementación de los métodos.

- multimap<Date, map<ID,Crimen>::iterator > > DateAccess;

Esta estructura se utilizará para permitir un acceso eficiente por fecha. Como es importante la secuencia cronológica, consideraremos un contenedor asociativo. Además, como puede haber más de un crimen en la misma fecha hemos seleccionado como estructura el multimap. En el campo definición solamente almacenamos un iterador que apunta a la dirección del map donde se encuentra el elemento.

Cuando un nuevo crimen se inserta en baseDatos debemos de insertar la posición en la que se insertó en el multimap y en el caso de borrarlo de la base de datos, también debe ser borrado del multimap.

- map<IUCR,set<ID> > IUCRAccess;

Esta estructura se utiliza para facilitar el acceso por IUCR, en este caso para cada IUCR tenemos el conjunto de delitos, representados por su ID, que han sido clasificados mediante dicho código. Utilizamos un map por ser importante el orden de los delitos dentro del conjunto.

Esta estructura se actualiza cada vez que se inserta o borra un nuevo delito.

- unordered_map<Termino, set<ID> > index;

En este caso, para cada termino en la descripción de un delito almacenamos en un conjunto ordenado los IDs de los delitos que han sido descritos mediante dicho termino.

Esta estructura se actualiza cada vez que se inserta o borra un nuevo delito.

- map<Longitud,multimap<Latitud, ID> > posicionGeo;

En este caso la posición geográfica la almacenamos en una estructura ordenada donde la clave, que asumimos única, se corresponde con la longitud donde se produce el delito. En este caso, los delitos están ordenados en orden creciente por este valor de coordenada. Para cada una de ellas almacenamos las coordenadas de latitud donde se cometió el delito. Como para una misma coordenada x,y puede haber más de un delito en el tiempo, se considera una estructura de multimap.

Esta estructura se actualiza cada vez que se inserta o borra un nuevo delito.

1.4 Iterando sobre CSS

Se han definido distintos criterios para poder iterar sobre un CSS. Como se ha visto, para cada criterio guardamos una estructura que de forma eficiente me permite avanzar por los delitos siguiendo el orden establecido.

Pasamos a ver cuál sería la representación interna que se propone para cada uno de ellos.

```
class CSS {
public:


    class iterator {
    private:
        /* @brief it  itera sobre los ID del map
      */
        map<ID,Crimen>::iterator it;
    public:
           pair<const ID, Crimen > & operator*();

    };
    class IUCR_iterator {
    private:
      /* @brief it_m itera sobre los IUCR del map
      */
      map<IUCR,set<ID> >::iterator it_m;
    /* @brief it_s itera sobre los ID del set
      */
      set<ID>::iterator it_s;
    public:
        pair<const ID, Crimen > & operator*();

     };
    class Date_iterator {
private:
         multimap<Date, map<ID,Crimen>::iterator > >::iterator it_mm;
      public:
        pair<const ID, Crimen > & operator*();

     };
```

1.4.1 Algunos Métodos sobre iteradores

En general los métodos sobre iteradores son sencillos de implementar, sólo hay que envolver los iteradores primitivos (sobre map o multimap) con el paraguas de la clase CSS. Sin embargo, hay algunas distinciones que se deberían hacer a la hora de abordar su implementación.

- Operador de derefernciación, operator∗().

En este caso, el valor devuelto por el operador ∗ de todos los iteradores es pair<const ID,Crimen> &

```
pair<const ID, Crimen > &  operator*();
```

Notar el const ID, dentro del primer campo del pair. Esto se hace para evitar que se pueda modificar dicha clave desde fuera de CSS. Los valores de la definición si se pueden modificar. Asumimos que el usuario no modificará el ID de la definición del crimen, para no permitir que un crimen tuviese distintos valores de ID en la clave y en la definición. Lo ideal sería que no estuviese el ID dentro del crimen.

El el caso del iterator, y también Date_iterator su implementación es simple, sólo tenemos que devolver el pair que se accede directamente de it o bien accediendo al elemento que esta en la posición second del it_mm

Sin embargo, para el IUCR iterator que internamente tiene dos atributos: el primero que itera sobre el map y el segundo que itera sobre el set. Este segundo atributo es que realmente hace referencia al ID del delito concreto al que apunta el iterador. Por tanto, para obtener el pair que debemos devolver sólo debemos de coger el ID apuntado por it_s, esto es, ∗it_s y buscar el crimen con ese ID en el map, por ejemplo haciendo baseDatos.find(∗it_s)

- IUCR_iterator & operator++();

La implementación requiere avanzar al siguiente delito en el orden correcto, para ello es suficiente con avanzar el it_s hasta que se alcance el final del set, en cuyo caso debemos avanzar it_m al siguiente IUCR del map, y posicionarnos en lo que sería el primer elemento de su set correspondiente, si existe.

1.5 Consulta Libre

Veamos cómo implementar el siguiente método de forma eficiente

```
vector<pair<ID,float> > Query(list<string> & q, int k);
```

Para ello utilizaremos la estructura index que está en el CSS. Esta estructura es un unorderd_map, que para cada término tiene el conjunto de ID para los que dicho término forma parte de la descripción.

```
unordered_map<Termino, set<ID> > index;
```

Así podemos distinguir tres casos:

- 1) Si la lista tiene un único termino, buscamos el término en el index y devolvemos una lista con todos los ID del conjunto asociado, con el valor second del par a 1.0;
- 2) Si la lista tiene dos términos, debemos hacer una unión de las dos listas de IDs asociadas a los términos, pero en el caso en que exista un ID que esté en ambos conjuntos (los dos términos están en la descripción del ID), debemos contar su peso como 2, en caso contrario como los IDs pertenecen sólo a una lista y por tanto su peso será 1. En este proceso se puede aprovechar que los conjuntos están ordenados en orden creciente de ID. Para ello, podemos diseñar un método privado de la clase CSS con el siguiente prototipo

```
map<ID,float> unionPeso( const set<ID> & t1, const set<ID> &t2);
```

- 3) Más de dos términos.

En este caso, podemos utilizar el método anterior para hacer la unión de las dos primeras listas de ID, y utilizar el siguiente método para ir actualizando en el map la información asociada al resto de términos de la consulta

```
void unionPeso( map<ID,float> & m, set<ID> & t_i);
```

Así podemos hacer algo como

```
Para cada termino t_i de la consulta, con i = 3, 4, ...
   Obtener el set<ID> asociado a t_i, s_i
   unionPeso(mi_map, s_i)
```

1.5.1 Selección de los k mejores elementos

Al finalizar el proceso anterior, tenemos en el map los ID con su peso asociado, el paso final sería ordenar estos elementos por el valor del peso. Dado que el conjunto de crímenes es del orden de 6 millones de registros podemos esperar que el conjunto de elementos a ordenar será muy grande, del orden de decenas de miles. Sin embargo, nosotros sólo necesitamos seleccionar los k (por ejemplo, 50 elementos con mayor peso).

Para hacer este proceso eficiente se debe utilizar una priority_queue, donde se insertarán elementos de forma que el que tenga menor peso esté en el tope de la cola. Esta cola nunca tendrá un tamaño mayor que k.

Recorreremos el map resultante de la etapa anterior e insertaremos los primeros k elementos. Para el resto de elementos, iteramos sobre ellos y comprobaremos si el peso del elemento apuntado por el iterador, it, es mayor que el elemento que está en el tope. De ser cierto, se sacará este elemento de la cola y se sustituirá por el elemento apuntado por it. En caso contrario lo podemos descartar pues estamos seguros que dicho delito no estará entre los k primeros.

Chapter 2

Todo List

Class crimen

Implementa esta clase, junto con su documentación asociada

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 4

Class Documentation

4.1 crimen::ComparacionPorFecha Class Reference

Public Member Functions

- bool **operator()** (const crimen &a, const crimen &b)

The documentation for this class was generated from the following file:

- crimen.h

4.2 crimen::ComparacionPorID Class Reference

Public Member Functions

- bool **operator()** (const crimen &a, const crimen &b)

The documentation for this class was generated from the following file:

- crimen.h

4.3 crimen::ComparacionPorIUCR Class Reference

Public Member Functions

- bool **operator()** (const crimen &a, const crimen &b)

The documentation for this class was generated from the following file:

- crimen.h

4.4 crimen Class Reference

Clase crimen, asociada a la definición de un crimen.

`#include <crimen.h>`

Classes

- class ComparacionPorFecha
- class ComparacionPorID
- class ComparacionPorIUCR

Public Member Functions

- crimen ()

  Constructor primitivo de la clase.
- crimen (const crimen &x)

  Constructor de copia de la clase.
- void setID (long int &id)

  Establecer el ID de un crimen.
- void setCaseNumber (const string &s)

  Establecer el número del caso de un crimen.
- void setDate (const fecha &d)

  Establecer la fecha de un caso.
- void setArrest (bool a)

  Establecer si se produce un arresto o no.
- void setDomestic (bool d)

  Establecer si es un crimen doméstico.
- unsigned int getID () const

  Obtener el ID de un crimen.
- string getCaseNumber () const

  Obtener el número del caso.
- string getDescription () const

  Obtener la descripción del crimen.
- bool getArrest () const

  Obtener si hay arresto.
- string getIUCR () const

  Obtener el IUCR del crimen.
- void setIUCR (string new_IUCR)

  Asignar un IUCR a un crimen.
- void setDescription (string new_Descr)

  Asignar una descripción a un crimen.
- fecha getDate () const

  Devuelve la fecha del crimen.
- double getLatitude () const

  Devuelve la latitud del crimen.
- double getLongitude () const

  Devuelve la longitud del crimen.
- crimen & operator= (const string &datos)

  Copia en un crimen los datos de otro pasado como string.
- crimen & operator= (const crimen &c)

  Iguala dos crímenes.
- bool operator== (const crimen &x) const

  Compara si son iguales dos crímenes.
- bool operator< (const crimen &x) const

  Compara si un conjunto es mayor que otro.

Friends

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

  Imprime todas las entradas del crimen.

4.4.1 Detailed Description

Clase crimen, asociada a la definición de un crimen.

crimen::crimen, ... Descripción contiene toda la información asociada a un crimen.

Todo Implementa esta clase, junto con su documentación asociada

4.4.2 Constructor & Destructor Documentation

4.4.2.1 crimen::crimen ( )

Constructor primitivo de la clase.

Clase crimen, asociada a la definición de un crimen.

crimen::crimen, ... Descripción contiene toda la información asociada a un crimen.

/**Constructor primitivo de la clase.

Postcondition

Se crea un nuevo objeto del tipo crimen.

4.4.2.2 crimen::crimen (const crimen & *x*)

Constructor de copia de la clase.

Parameters

in	*c*	crimen a copiar.
in	*x*	Crimen del que se copian los datos.

Postcondition

Se crea un nuevo objeto del tipo crimen con los datos del objeto x.

4.4.3 Member Function Documentation

4.4.3.1 bool crimen::getArrest ( ) const

Obtener si hay arresto.

Returns

bool true si hay arresto, false en caso contrario.

4.4.3.2 string crimen::getCaseNumber ( ) const

Obtener el número del caso.

Returns

Devuelve el número del caso.
El número del caso.

4.4.3.3 fecha crimen::getDate ( ) const

Devuelve la fecha del crimen.

Returns

Devuelve la fecha del crimen.
La fecha del crimen.

4.4.3.4 string crimen::getDescription ( ) const

Obtener la descripción del crimen.

Returns

string con la descripción del caso.

4.4.3.5 unsigned int crimen::getID ( ) const

Obtener el ID de un crimen.

Returns

Devuelve el ID.
El ID del crimen.

4.4.3.6 string crimen::getIUCR ( ) const

Obtener el IUCR del crimen.

Returns

string con el IUCR del caso.

4.4.3.7 double crimen::getLatitude ( ) const

Devuelve la latitud del crimen.

Returns

Devuelve la latitud del crimen.

4.4.3.8 double crimen::getLongitude ( ) const

Devuelve la longitud del crimen.

Returns

Devuelve la longitud del crimen.

4.4.3.9 bool crimen::operator< (const crimen & *x*) const

Compara si un conjunto es mayor que otro.

Compara si un conjunto es menor que otro.

Parameters

x	Crimen a comparar.

Returns

Devuelve true si x es mayor que el que lo llama. False en otro caso.

Parameters

in	*x*	Crimen a comparar.

Returns

true si x es mayor que el que lo llama. False en otro caso.

4.4.3.10 crimen & crimen::operator= (const string & *datos*)

Copia en un crimen los datos de otro pasado como string.

Parameters

in	*string*	con los datos a copiar de otro crimen.

4.4.3.11 crimen & crimen::operator= (const crimen & *c*)

Iguala dos crímenes.

Parameters

c	Crimen a copiar en el que lo llama.

Returns

Devuelve una copia del crimen.

Parameters

in	*c*	Crimen a copiar en el que lo llama.

Returns

Una copia del crimen.

4.4.3.12 bool crimen::operator== (const crimen & *x*) const

Compara si son iguales dos crímenes.

Parameters

x	Crimen a comparar.

Returns

Devuelve true si son iguales y false si no lo son.

Parameters

in	*x*	Crimen a comparar.

Returns

true si son iguales y false si no lo son.

4.4.3.13 void crimen::setArrest (bool *a*)

Establecer si se produce un arresto o no.

Parameters

	a	Se produce arresto -> True / No se produce arresto -> False
in	*a*	Se produce arresto -> True / No se produce arresto -> False

4.4.3.14 void crimen::setCaseNumber (const string & *s*)

Establecer el número del caso de un crimen.

Parameters

	in	s Número del caso.
in	*s*	Número del caso.

4.4.3.15 void crimen::setDate (const fecha & *d*)

Establecer la fecha de un caso.

Parameters

	d	Fecha a establecer.
in	*d*	Fecha a establecer.

4.4.3.16 void crimen::setDescription (string *new_Descr*)

Asignar una descripción a un crimen.

Parameters

in	*string*	con la descripción a asignar.

4.4.3.17 void crimen::setDomestic (bool *d*)

Establecer si es un crimen doméstico.

Parameters

	d	Es crimen doméstico -> True / No es crimen doméstico -> False
in	*d*	Es crimen doméstico -> True / No es crimen doméstico -> False

4.4.3.18 void crimen::setID (long int & *id*)

Establecer el ID de un crimen.

Parameters

in	*id*	ID a establecer.

4.4.3.19 void crimen::setIUCR (string *new_IUCR*)

Asignar un IUCR a un crimen.

Parameters

in	*string*	con el IUCR a asignar.

4.4.4 Friends And Related Function Documentation

4.4.4.1 ostream& operator<< (ostream & , const crimen & *x*) [friend]

Imprime todas las entradas del crimen.

Postcondition

No se modifica el crimen original.

The documentation for this class was generated from the following files:

- crimen.h
- crimen.hxx

4.5 CSS Class Reference

Classes

- class Date_iterator
- class iterator
- class IUCR_iterator

Public Member Functions

- void load (string nombreDB)

  Se encarga de leer los elementos de un fichero dado por el argumento nombreDB, e insertar toda la información en la base de datos. Recordad que para esta práctica se os pide que extendáis el campo descripción a todas las descripciones que se encuentra en la base de datos.Pare ello será necesario modificar la clase crimen utilizada en la práctica anterior.
- void insert (const crimen &x)

  Este método se encarga de insertar un nuevo crimen en CSS. Como prerrequisito se asume que el crimen no está ya almacenado en el conjunto.

- bool erase (unsigned int ID)

  En este caso, se trata de borrar un crimen de la base de datos dado su ID. Devuelve verdadero si el crimen ha sido borrado correctamente, falso en caso contrario.

- void setArrest (const unsigned int ID, bool value)

  Modifica el campo arrest de un crimen identificado por ID. Será necesario cuando se detenga un criminal con posterioridad a la inserción del delito en CSS.

- vector< pair< ID, float > > Query (list< string > &q, int k)

  Si la lista tiene un único termino, buscamos el término en el index y devolvemos una lista con todos los ID del conjunto asocidado, con el valor second del par a 1.0. Si la lista tiene dos términos, debemos hacer una unión de las dos listas de IDs asociadas a los términos, pero en el caso en que exista un ID que esté en ambos conjuntos (los dos términos están en la descripción del ID), debemos contar su peso como 2, en caso contrario como los IDs pertenecen sólo a una lista y por tanto su peso será 1. En este proceso se puede aprovechar que los conjuntos están ordenados en orden creciente de ID. Para ello, podemos diseñar un método privado llamado "unionPeso". Para más de dos términos podemos utilizar el método anterior para hacer la unión de las dos primeras listas de ID, y utilizar el siguiente método para ir actualizando en el map la información asociada al resto de términos de la consulta.

- list< ID > inArea (Longitud x1, Latitud y1, Longitud x2, Latitud y2)

  En este caso la posición geográfica la almacenamos en una estructura ordenada donde la clave, que asumimos única, se corresponde con la longitud donde se produce el delito. En este caso, los delitos están ordenados en orden creciente por este valor de coordenada. Para cada una de ellas almacenamos las coordenadas de latitud donde se cometió el delito. Como para una misma coordenada x,y puede haber más de un delito en el tiempo, se considera una estructura de multimap.

- iterator find_ID (const unsigned int ID)

  Devuelve un iterador al ID pasado por parámetro y en caso de no existir devuelve un iterador al principio del conjunto.

- iterator begin ()

  Devuelve un iterador al primer crimen del conjunto ordenado por ID.

- iterator end ()

  Devuelve un iterador al último crimen del conjunto ordenado por ID.

- IUCR_iterator ibegin ()

  Devuelve un iterador al primer crimen del conjunto ordenado por IUCR.

- IUCR_iterator iend ()

  Devuelve un iterador al último crimen del conjunto ordenado por IUCR.

- Date_iterator dbegin ()

  Devuelve un iterador al primer crimen del conjunto ordenado por fecha.

- Date_iterator dend ()

  Devuelve un iterador al último crimen del conjunto ordenado por fecha.

- IUCR_iterator lower_bound (IUCR)

  Devuelve la cota inferior en el contenedor ordenado por IUCR del IUCR pasado como parámetro.

- IUCR_iterator upper_bound (IUCR)

  Devuelve la cota superior en el contenedor ordenado por IUCR del IUCR pasado como parámetro.

- Date_iterator lower_bound (fecha)

  Devuelve la cota inferior en el contenedor ordenado por fecha de la fecha pasada como parámetro.

- Date_iterator upper_bound (fecha)

  Devuelve la cota superior en el contenedor ordenado por fecha de la fecha pasada como parámetro.

The documentation for this class was generated from the following files:

- css.h
- css.hxx

4.6 CSS::Date_iterator Class Reference

Public Member Functions

- pair< const ID, crimen > & **operator*** ()
- Date_iterator **operator++** (int)
- Date_iterator & **operator++** ()
- bool **operator==** (const Date_iterator &it)
- bool **operator!=** (const Date_iterator &it)

Friends

- class **CSS**

The documentation for this class was generated from the following files:

- css.h
- css.hxx

4.7 fecha Class Reference

Public Member Functions

- fecha ()

  Constructor primitivo de la clase.
- fecha (const string &s)

  Constructor de copia de la clase.
- fecha & operator= (const fecha &f)

  operador de asignación
- fecha & operator= (const string &s)

  operador de asignación
- string toString () const

  Muestra el valor de sus atributos.
- string DarFormato (int param) const

  Imprime datos en formato correcto (horas, minutos, etc)
- bool operator== (const fecha &f) const

  Compara si son iguales dos fechas.
- bool operator< (const fecha &f) const

  Compara si una fecha es mayor que otra.
- bool operator> (const fecha &f) const

  Compara si una fecha es menor que otra.
- bool operator<= (const fecha &f) const

  Compara si una fecha es mayor o igual que otra.
- bool operator>= (const fecha &f) const

  Compara si una fecha es menor o igual que otra.
- bool operator!= (const fecha &f) const

  Es un comparador de desigualdad.

Friends

- ostream & operator<< (ostream &os, const fecha &f)

  Imprime todas las entradas de las fechas.

4.7.1 Constructor & Destructor Documentation

4.7.1.1 fecha::fecha ( )

Constructor primitivo de la clase.

fichero de implementacion de la clase fecha

Constructor sin parametros de la clase.

Postcondition

Se crea un nuevo objeto fecha con parametros por defecto.

4.7.1.2 fecha::fecha (const string & *x*)

Constructor de copia de la clase.

Parameters

in	*s*	string a copiar:
in	*x*	fecha del que se copian los datos.

Postcondition

Se crea un nuevo objeto del tipo fehca con los datos del objeteo x.

4.7.2 Member Function Documentation

4.7.2.1 string fecha::DarFormato (int *param*) const

Imprime datos en formato correcto (horas, minutos, etc)

Parameters

in	*param*	entero que se comprueba y/o formatea

4.7.2.2 bool fecha::operator!= (const fecha & *f*) const

Es un comparador de desigualdad.

Compara si una fecha.

Parameters

in	*f*	Fecha a comparar.

Returns

Devuelve true cuando f es distinta del que la llama.False en otro caso.

Parameters

in	*f*	fecha a comparar.

Returns

true si es mayor.

4.7.2.3 bool fecha::operator< (const fecha & *f*) const

Compara si una fecha es mayor que otra.

Compara si una fecha es menor que otra.

Parameters

in	*f*	Fecha a comparar.

Returns

Devuelve true si f es mayor que el que lo llama. False en otro caso.

Parameters

in	*f*	fecha a comparar.

Returns

true si es menor.

4.7.2.4 bool fecha::operator<= (const fecha & *f*) const

Compara si una fecha es mayor o igual que otra.

Compara si una fecha es menor o igual que otra.

Parameters

in	*f*	Fecha a comparar.

Returns

Devuelve true si f es mayor o igual que el que lo llama.False en otro caso.

Parameters

in	*f*	fecha a comprar.

Returns

true si es menor.

4.7.2.5 fecha & fecha::operator= (const fecha & *f*)

operador de asignación

Iguala dos fechas.

Parameters

in	*f*	fecha a copiar.

Postcondition

Crea una fecha duplicada exacta de f

Parameters

in	*f*	fecha a copiar en el que lo llama.

Returns

Una copia de la fecha.

4.7.2.6 fecha & fecha::operator= (const string & *s*)

operador de asignación

Iguala dos fechas.

Parameters

in	*s*	string a copiar.
in	*s*	fecha a copiar en el que lo llama.

Returns

Una copia de la fecha.

4.7.2.7 bool fecha::operator== (const fecha & *f*) const

Compara si son iguales dos fechas.

Parameters

in	*f*	Fecha a comparar.

Returns

Devuelve true si son iguales y false si no lo son.

Parameters

in	*f*	fecha a comparar.

Returns

true si son iguales y false si no lo son.

4.7.2.8 bool fecha::operator> (const fecha & *f*) const

Compara si una fecha es menor que otra.

Compara si una fecha es mayor que otra.

Parameters

in	*f*	Fecha a comparar.

Returns

Devuelve true si f es menor que el que lo llama. False en otro caso.

Parameters

in	*f*	fecha a comprar.

Returns

true si es mayor.

4.7.2.9 bool fecha::operator>= (const fecha & *f*) const

Compara si una fecha es menor o igual que otra.

Compara si una fecha es mayor que otra.

Parameters

in	*f*	Fecha a comparar.

Returns

Devuelve true si f es menor o igual que el que lo llama.False en otro caso.

Parameters

in	*f*	fecha a comprar.

Returns

true si es mayor.

4.7.2.10 string fecha::toString ( ) const

Muestra el valor de sus atributos.

Copiar los datos de una fecha a un string.

Returns

Devuelve la fecha convertida a un string

Postcondition

Se crea un objeto string con los datos de fecha

4.7.3 Friends And Related Function Documentation

4.7.3.1 ostream& operator<< (ostream & *os,* const fecha & *f*) [friend]

Imprime todas las entradas de las fechas.

Postcondition

No se modifica la fecha original

Parameters

in	*os*	flujo
in	*f*	fecha a mostrar.

Returns

La fecha por salida estandar.

The documentation for this class was generated from the following files:

- fecha.h
- fecha.hxx

4.8 CSS::iterator Class Reference

Public Member Functions

- pair< const ID, crimen > & **operator*** ()
- iterator **operator++** (int)
- iterator & **operator++** ()
- bool **operator==** (const iterator &it)
- bool **operator!=** (const iterator &it)

Friends

- class **CSS**

The documentation for this class was generated from the following files:

- css.h
- css.hxx

4.9 CSS::IUCR_iterator Class Reference

Public Member Functions

- pair< const ID, crimen > & **operator*** ()
- IUCR_iterator **operator++** (int)
- IUCR_iterator & **operator++** ()
- bool **operator==** (const IUCR_iterator &it)
- bool **operator!=** (const IUCR_iterator &it)

Friends

- class **CSS**

The documentation for this class was generated from the following files:

- css.h
- css.hxx