# TDA CLINVAR

V1

Generado por Doxygen 1.8.11

# Índice



## 1. CLINVAR Clinical Variants DataSet

**Versión**

v1

**Autor**

Juan F. Huete y Carlos Cano

### 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.

En esta práctica no se considerará el uso de templates, por lo que debemos seguir los principios de la compilación separada, esto es, tendremos un fichero clinvar.h y clinvar.cpp pudiendo generar el código objeto asociado a clinvar.o, que será el que se enlazará con nuestro programa de prueba.

Finalmente, el alumno podrá dotar al tipo de dato de otros métodos que considere necesarios para la correcta realización de la práctica, pero debe respetar escrupulosamente la cabecera de los métodos detallados en este documento.

## 1.2. Introducción

ClinVar (https://www.ncbi.nlm.nih.gov/clinvar/) es una Base de Datos mantenida por el Instituto de Salud de USA (National Institute of Health) en la que se agrega información sobre mutaciones del genoma humano y sus relaciones con enfermedades. Con el objetivo de facilitar el acceso a estos datos, en esta práctica se propone construir una nueva interfaz que permita consultas más flexibles. Además, impondremos ciertos requisitos sobre la eficiencia de los métodos de gestión y acceso a la información de ClinVar, ya que la escalabilidad es un aspecto muy importante de esta nueva interfaz, que debe responder a un alto número de consultas.

Se debe diseñar un nuevo tipo de dato que llamaremos ClinVar que tiene que cumplir los siguientes requisitos:

- Representar las mutaciones con el TDA mutación y las enfermedades con el TDA enfermedad especificados en la práctica 2: "TDA Mutación y TDA Enfermedad".
- Obtener las mutaciones ordenadas por cromosoma/posicion.
- Permitir un acceso eficiente a las mutaciones por ID.
- Permitir recuperar todas las mutaciones que afectan a un gen determinado.
- Obtener las enfermedades que hay en la base de datos.
- Dado un identificador de enfermedad, identificar las mutaciones que están asociadas con la enfermedad. Si una enfermedad no dispone de identificador, puede ser ignorada.
- Consultar todas las enfermedades que contienen un determinado término en su nombre (por ejemplo, "cancer" o "prostate_cancer").
- Dado un determinado término para el nombre de una enfermedad (ejemplo, "cancer" o "prostate_cancer"), obtener la secuencia ordenada de las K mutaciones que con mayor probabilidad estan relacionadas (son causantes) de la misma.
- Dado el elevado número de mutaciones en la base de datos (más de 130.000), se debe evitar duplicar la información de la base de datos, esto es, NO está permitido mantener copias redundantes de los datos en distintas estructuras y TDAs. En su lugar, la información completa de cada mutación se almacenará una única vez y habrá estructuras adicionales que nos permitan alcanzar los objetivos propuestos.
- Se debe dotar de la capacidad de iterar sobre los conjuntos de enfermedades y mutaciones, para este último considerando diversos criterios, el primero iterar en orden creciente cromosoma/posición y el segundo iterar considerando orden creciende de identificador de gen. Éste último se verá mas adelante.
- Finalmente, se desea poder disponer del sistema funcionando antes del final del año 2016, por lo que la solución debe entregarse antes del día 27 de diciembre de 2016 a las 23:59

## 1.3. Tipo de dato ClinVar

La empresa de nueva creación EDGRX, ubicada en Granada, decide presentar su propuesta de diseño del tipo de dato ClinVar.

### 1.3.1. Parte Pública

En la parte pública podemos distinguir, entre otros que se considere necesarios, tres tipos de datos:

- iterator que nos permite iterar por las mutaciones en orden creciente de cromosoma/posicion
- enfermedad_iterator que nos permite iterar sobre el conjunto de enfermedades
- gen_iterator que itera sobre las mutaciones considerando el orden creciente del ID del gen.

En la sección de iteradores se entra en mas detalle sobre ellos.

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

```
//los siguientes typedef permiten identificar en las cabeceras de los métodos
//cuando un string se refiere a un ID de gen, mutacion o enfermedad, respectivamente.
typedef string IDgen;
typedef string IDmut;
typedef string IDenf;

class clinvar{
  public:
    ...
    void load (string nombreDB);
    void insert (const mutacion & x);
    bool erase (IDmut ID);
    iterator find_Mut (IDmut ID);
    enfermedad_iterator find_Enf(IDenf ID);

    vector<enfermedad> getEnfermedades(mutacion & mut);
    list<IDenf> getEnfermedades(string keyword);
    set<IDmut> getMutacionesEnf (IDenf ID);
    set<IDmut> getMutacionesGen (IDgen ID);
    set<mutacion,ProbMutaciones> topKMutaciones (int k, string keyword);


    /* Métodos relacionados con los iteradores */
    iterator begin();
    iterator end();
    iterator lower_bound(string cromosoma, unsigned int posicion);
    iterator upper_bound(string cromosoma, unsigned int posicion);

    enfermedad_iterator ebegin();
    enfermedad_iterator eend();

    gen_iterator gbegin();
    gen_iterator gend();
    ...
}
```

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 ClinVar.

- void insert (const mutacion & x);

Este método se encarga de insertar una nueva mutación en ClinVar. Esto implica actualizar todas las estructuras necesarias para mantener la coherencia interna de la representación propuesta.

- bool erase (IDmut ID);

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

No sólo borra la mutación del repositorio principal de datos sino que además se encarga de borrar toda referencia a dicho elemento dentro de él.

En el caso de que una enfermedad estuviese asociada únicamente a la mutación que está siendo eliminada, esta enfermedada también debe eliminarse de ClinVar.

- iterator find_Mut (IDmut ID);

Busca la mutación con identificador ID dentro de ClinVar, si no lo encuentra devuelve end()

- enfermedad_iterator find_Enf(IDenf ID);

Busca la enfermedad con identificador ID dentro de ClinVar, si no lo encuentra devuelve eend()

- vector<enfermedad> getEnfermedades(mutacion & mut);

Devuelve un vector con todas las enfermedades asociadas a una mutación en la base de datos clinvar.

- list<IDenf> getEnfermedades(const string & keyword);

Devuelve una lista de los identificadores de enfermedad que contienen la palabra keyword como parte del nombre de la enfermedad. Utilizar enfermedad.nameContains() para programar este método.

- set<IDmut> getMutacionesEnf (IDenf ID);

Devuelve un conjunto ordenado (en orden creciente de IDmut) de todas las mutaciones que se encuentran asociadas a la enfermedad con identificador ID. Si no tuviese ninguna enfermedad asociada, devuelve el conjunto vacío.

- set<IDmut> getMutacionesGen (IDgen ID);

Devuelve un conjunto de todas las mutaciones que se encuentran asociadas a un gen determinado dado por ID. Si no tuviese ninguno, devuelve el conjunto vacío.

- set<mutacion,ProbMutaciones> topKMutaciones (int k, string keyword); Dado un string 'keyword', el sistema recupera todas las enfermedades cuyo nombre contiene keyword, y devuelve un set ordenado de mutaciones, en orden decreciente de probabilidad, con las k mutaciones más frecuentes en la población asociadas con esas enfermedades.

Más información sobre cómo implementar esta función en la sección “TopKMutaciones” de este documento.

- iterator lower_bound(string cromosoma, unsigned int posicion);
- iterator upper_bound(string cromosoma, unsigned int posicion);

Son métodos que permiten hacer la búsqueda por rango considerando el par de valores cromosoma/posición. El comportamiento es similar al que hemos visto en prácticas anteriores, lower_bound devuelve el iterador que apunta a la primera mutación que es mayor o igual a los parámetros dados en la entrada, mientras que upper_↩ bound devuelve la primera estrictamente. En ambos casos, si no hay ninguna devuelve end.

- iterator begin();
- enfermedad_iterator ebegin();
- gen_iterator gbegin();

Devuelve el iterador correspondiente a la primera mutación/enfermedad/gen según el criterio de ordenación de cada tipo de elemento.

- iterator_iterator end();
- enfermedad_iterator eend();
- gen_iterator gend();

Devuelve el iterador que apunta al elemento siguiente al último elemento según el criterio de ordenación de cada tipo de elemento.

### 1.3.2. Parte privada

En este caso hablaremos de los atributos que se han escogido para representar la información. La siguiente imagen ilustra la idea.

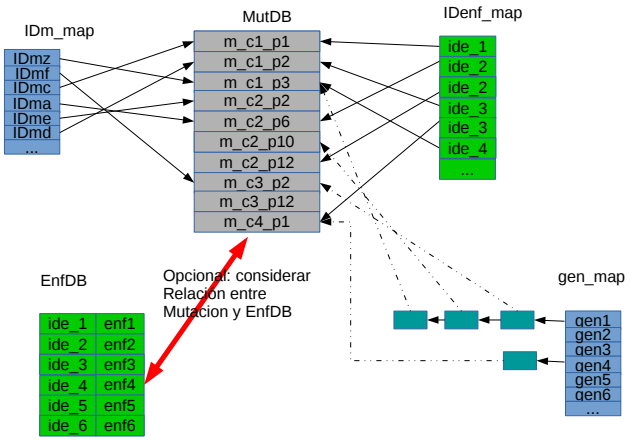


**Figura 1 Atributos de la clase clinvar**

```
class clinvar{
...
  private:
    set<mutacion> mutDB; //base de datos que contiene toda la información asociada a una mutacion

    unordered_map<IDmut,set<mutacion>::iterator> IDm_map; // Asocia IDmutacion con mutación

    map<IDgen, list< set<mutacion>::iterator> > gen_map; // Asocia genes con mutaciones

    map<IDenf,Enfermedad> EnfDB; // Base de datos de enfermedades

    multimap<IDenf,set<mutacion>::iterator> IDenf_map; // Asocia enfermedad con mutaciones

};
```

#### 1.3.3. Detallando los atributos

Pasamos a detallar cada uno de los atributos de la clase clinvar.

- set<mutacion> mutDB;

Toda la información relativa a las mutaciones se almacena en un conjunto (set), ordenados en orden creciente de cromosama/posición que llamaremos mutDB.

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

- unordered_map<IDmut,set<mutacion>::iterator> IDm_map;

Esta estructura se utiliza para permitir un acceso eficiente por IDmut. Para cada ID de mutación, almacenamos un iterador al objeto completo (que está almacenado en el set mutDB). En este caso, ya que la especificación no nos indica que sea importante el orden entre los distintos IDmut y, además, los IDmut son únicos, utilizaremos un unordered_map para facilitar las operaciones de acceso (O(1)).

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

- map<IDgen, list< set<mutacion>::iterator> > gen_map;

Esta estructura se utilizará para permitir un acceso eficiente al conjunto de mutaciones por IDgen. Resulta útil para los usuarios finales listar las mutaciones ordenadas según el identificador del gen, por lo que consideraremos este contenedor asociativo. Además, como un mismo IDgen puede estar asociado a distintas mutaciones, hemos optado por representar para cada IDgen la lista de iteradores que apuntan a las mutaciones asociadas a este gen en el set.

Cuando una nueva mutación se inserta en mutDB se deben insertar iteradores a esta mutación en todas las listas de gen_map asociadas a los genes relacionados con esta mutación. En el caso de borrado de una mutación, también deben borrarse sus iteradores asociados del map.

- map<IDenf,Enfermedad> EnfDB;

Esta estructura se utiliza para facilitar el acceso a las enfermedades, y contendrá toda la información asociada a las misma. Las tendremos ordenadas por IDenf y por tanto utilizamos un map.

Esta estructura se actualiza cada vez que se inserta o borra una nueva mutación en el conjunto.

- multimap<IDenf, set<mutacion>::iterator> IDenf_mmap;

Esta estructura se utiliza para facilitar el acceso a mutaciones asociadas a una enfermedad determinada. Como cada enfermedad puede estar asociada a varias mutaciones, proponemos en este caso un multimap en el que la clave (key) es el identificador de enfermedad y el valor asociado es un iterador a la mutación en el set. Esto lo hacemos así porque una misma enfermedad (IDenf) puede estar asociada a varias mutaciones. Por tanto, debemos permitir que haya varias entradas con la misma clave, y por esto utilizamos multimap. Es decir, si una enfermedad dada tiene asociadas N mutaciones, el multimap tendrá N entradas asociadas a dicha enfermedad, y en cada una de estas entradas se guardará el iterador de una de las N mutaciones.

Esta estructura se actualiza cada vez que se inserta o borra una nueva mutación en el conjunto.

#### 1.3.4. Modificar TDA mutacion (OPCIONAL)

Antes de continuar con la descripción de la práctica, hay una parte que es opcional (esto quiere decir que el alumno puede optar por no hacerla). El objetivo de la misma es el de tratar de cumplir el requisito de no duplicar información que nos viene impuesto por ClinVar. Así, teniendo en cuenta que en el TDA clinvar disponemos como atributo de la base de datos de enfermedades, se considera conveniente modificar la representación del TDA mutación. Para ello, en lugar de almacenar para cada mutación toda la información de las enfermedades con las que se asocia (esto es, el vector de enfermedades), es suficiente con almacenar el IDEnf, el resto de la información de la misma la podemos obtener de la base de datos EnfDB.

Por tanto, podremos de modificar el TDA mutación de forma que se consiga dicho objetivo, tanto a nivel de representación como de los métodos que la utilicen.

```
class mutacion {
 ...
 void setEnfermedades (const std::vector<IDenf> & enfermedades); //
 const  vector<IDenf> & getEnfermedades () const;
 ...
 private:
 ...
 vector<IDenf> enfermedades;
 ...
};
```

## 1.4. Iterando sobre clinvar

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

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

```
class clinvar {
public:

    /* @brief iterador sobre mutaciones en orden creciente de cromosoma/posicion
      */
    class iterator {
      private:
        set<mutacion>::iterator it;
      public:
        const mutacion & operator*();    //const - no se puede modificar la mutacion y alterar el orden del
       set

    };

    /* @brief iterador sobre enfermedades
      */
    // Nos vale utilizar el iterador del map
    typedef  map<IDenf, Enfermedad>::iterator enfermedad_iterator;


    /* @brief iterador sobre mutaciones considerando el orden creciente del ID del gen
      */
    class gen_iterator {
      public:
        const mutacion & operator*();   //const - no se puede modificar la mutacion y alterar el orden del
       set
      private:
        map<IDgen, list< set<mutacion>::iterator> > >::iterator itmap;
        list<set<mutacion>::iterator> >::iterator itlist;
        clinvar *ptrclinvar;
    };

};
```

#### 1.4.1. Algunos Métodos sobre iteradores

En general los métodos sobre iteradores son sencillos de implementar: sólo hay que envolver los iteradores primitivos con el paraguas de la clase clinvar. De hecho, como hemos visto, enfermedad_iterator es directamente el iterador sobre el map de enfermedades. Para abordar la implementación de los otros iteradores, debemos hacer notar las siguientes indicaciones.

- Operador $*()$.

El valor devuelto por el operador $*$ de todos los iteradores asociados a mutación debe ser de tipo const mutacion &, para que no se pueda modificar la mutación dereferenciada y exista el riesgo potencial de desordenar el set en el que se almacena.

```
const mutacion & operator*();
```

- gen_iterator

El objetivo de este iterador es recorrer todas las mutaciones en orden creciente según el ID del gen asociado. Dado que el map gen_map mantiene los genes ordenados en orden creciente según IDgen, para programar este iterador tendremos que recorrer las mutaciones en el orden indicado por este map. Para ello, los objetos gen_iterator tienen tres atributos: el primero para iterar sobre el map de genes, el segundo para iterar sobre la lista de mutaciones asociadas a un gen, y un tercero que será un puntero a clinvar. El segundo atributo es el que realmente hace referencia a las mutaciones concretas. Por ejemplo, el método:

```
gen_iterator & operator++();
```

requiere avanzar a la siguiente mutación en el orden correcto, para lo que es suficiente con avanzar el itlist si no se ha alcanzado el final de la lista. Así podemos recorrer todas las mutaciones asociadas a un gen. Si se alcanza el final de la lista de mutaciones asociadas a un gen, para continuar el recorrido por las mutaciones del siguiente gen, debemos avanzar itmap al siguiente gen, y posicionarnos en el primer elemento de su lista de mutaciones correspondiente.

Así, por ejemplo podemos ver que si utilizamos el siguiente código

```
clinvar X;

for (clinvar::iterator it=X.begin();it!=X.end();it++)
  cout « *it « endl;

for (clinvar::gen_iterator itg=X.gbegin(); itg!=X.gend();++itg)
 cout « *itg « endl;
```

Nos listará el mismo conjunto de mutaciones, pero considerando órdenes distintos, el primero siguiendo el criterio cromosoma/posición y el segundo por orden creciente de genID.

### 1.5. TopKMutaciones

Veamos cómo implementar el siguiente el método de forma eficiente:

```
set<mutacion,ProbMutaciones> topKMutaciones (int k, string keyword);
```

Dado un string 'keyword', el sistema recupera todas las enfermedades cuyo nombre contiene keyword, y devuelve un set ordenado de mutaciones, en orden decreciente de probabilidad, con las k mutaciones más frecuentes en la población asociadas con esas enfermedades.

#### 1.5.1. Sobre el tipo set<mutacion,ProbMutaciones>

Como el set de mutaciones tiene que estar ordenado por la probabilidad de la mutación, lo representaremos como un set<mutacion> con un functor llamado ProbMutaciones. ProbMutaciones determinará cuando una mutación es mas probable que otra y nos permitirá establecer la ordenación del set.

Entendemos que una mutación es más probable cuanto MENOR sea su valor caf[0], es decir, cuanto menos frecuente sea la base de referencia en la población (y más frecuente, por tanto, sean las bases alternativas especificadas en la mutación).

Para programar el functor ProbMutaciones, basta entonces con considerar que la probabilidad de una mutación en la población es 1-caf[0] (es decir, la probabilidad de que la referencia no ocurra). El functor debe calcular esta probabilidad para cada una de las dos mutaciones dadas como argumento y devolver la salida adecuada para implementar una ordenación DECRECIENTE (de mayor a menor probabilidad) en el set de mutaciones.

#### 1.5.2. topKMutaciones (int k, string keyword);

Para resolver este problema debemos utilizar las distintas estructuras almacenadas. En primer lugar, debemos determinar cuando una enfermedad está relacionada con el string 'keyword' y, para cada una de las enfermedades obtenidas, determinar las mutaciones asociadas. Estos procesos ya han sido implementados en otros métodos propuestos en la práctica.

Al finalizar el proceso anterior, debemos ordenar las mutaciones por el valor de su probabilidad. Dado que el conjunto completo de mutaciones contiene más de 130.000 elementos, podemos esperar que el conjunto de elementos a ordenar sea grande, del orden de miles. Sin embargo, sólo necesitamos seleccionar los k (por ejemplo, 10) elementos con mayor probabilidad.

Para hacer este proceso eficiente, se debe utilizar una priority_queue donde se insertarán mutaciones de forma que la que tenga menor probabilidad dentro de las k seleccionadas esté en tope de la cola. Esta cola nunca tendrá un tamaño mayor que k.

Así, al construir la priority_queue, debemos insertar las k primeras mutaciones directamente en la cola. Para el resto de elementos, comprobaremos si la probabilidad de la nueva mutación es mayor que la que está en el tope, que será la de menor probabilidad de las k seleccionadas. De ser cierto, se sacará la mutación del tope de la cola y se insertará la nueva mutación. En caso contrario, podemos descartar la mutación, pues tendremos la seguridad de que no está entre las k más probables. En cualquier caso, debemos asegurarnos que en la priority_queue NO HAY elementos repetidos, por lo que dispondremos de un contenedor adicional (unordered_set) con información sobre la mutaciones en la misma (por ejemplo los ID de mutación). Este se actualizará conforme se inserten y borren los elementos en la cola.

## 1.6. Entrega

El alumno debe empaquetar todos los archivos relacionados con el proyecto en un archivo con nombre "ClinVar.↩ zip". Tenga en cuenta que no se incluirán ficheros objeto, ni ejecutables, ni el fichero de datos de mutaciones. 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:

ClinVar.zip

- Makefile
- include – Carpeta con ficheros de cabecera (.h)

- src –Carpeta con código fuente (.cpp)
- doc –Carpeta con Documentación
- obj – Carpeta para código objeto (.o)
- bin – Carpeta para ejecutables
- datos – Carpeta para ficheros de datos

La fecha límite de entrega es el día 27 de diciembre de 2016 a las 23:59.