TDA Conjunto Mutaciones

V0

Generado por Doxygen 1.8.11

Índice

1. Documentación Práctica

Versión

v0

Autor

Carlos Cano y Juan F. Huete

1.1. Introducción

En la práctica anterior hemos creado el TDA Mutación y TDA Enfermedad. El objetivo de esta práctica es crear un TDA contenedor para almacenar y gestionar un conjunto de mutaciones.

1.2. Conjunto como TDA contenedor de información

Nuestro conjunto será un contenedor que permite almacenar la información de la base de datos de mutaciones. Para un mejor acceso, los elementos deben estar ordenados según chr/posición, en orden creciente. Como TDA, lo vamos a dotar de un conjunto restringido de métodos (inserción de elementos, consulta de un elemento por chr/pos o por ID, etc.). Este conjunto "simulará" un set de la stl, con algunas claras diferencias pues, entre otros, no estará dotado de la capacidad de iterar (recorrer) a través de sus elementos, que se hará en las siguientes prácticas.

Asociado al conjunto, tendremos los tipos

```
conjunto::value_type // tipo de dato almacenado en el conjunto
conjunto::size_type  // numero de elementos del conjunto
conjunto::iterator  // iterador sobre los elementos del conjunto
conjunto::const_iterator //Iterador constante
```

que permiten hacer referencia a los elementos almacenados en cada una de las posiciones y el número de elementos del conjunto, respectivamente. Es requisito que el tipo conjunto::value_type tenga definidos los operadores operator< y operator= .

1.3. Representación

El alumno deberá realizar una implementación utilizando como base el TDA vector de la STL. En particular, la representación que se utiliza es un VECTOR ORDENADO de entradas, teniendo en cuenta el valor de los atributos chr/pos, tal y como se especificó al definir el operator< en el TDA Enfermedad.

1.3.1. Función de Abstracción :

Función de Abstracción: AF: Rep => Abs

```
dado C =(vector<mutaciones> vm ) ==> Conjunto BD;
```

Un objeto abstracto, BD, representando una colección ORDENADA de mutaciones según chr/pos, se instancia en la clase conjunto como un vector ordenado de mutaciones.

1.3.2. Invariante de la Representación:

Propiedades que debe cumplir cualquier objeto

```
BD.size() == C.vm.size();

Para todo i, 0 <= i < C.vm.size() se cumple
    C.vm[i].chr está en ("1", "2", "3", "4", "5", "6", "7", "8", "9", "10", "11", "12", "13", "14", "15", "
      16", "17", "18", "19", "20", "21", "22", "X", "Y", "MT")
  C.vm[i].pos > 0;
Para todo i, 0 <= i < C.vm.size()-1 se cumple:
  a) si C.vm[i].chr == C.vm[i+1].chr, entonces: C.vm[i].pos < C.vm[i+1].pos
  b) si C.vm[i].chr != C.vm[i+1].chr, entonces  C.vm[i].chr < C.vm[i+1].chr
  (donde el orden para el número de cromosoma se rige por "1"<"2"<"3"<...<"22"<"X"<"Y"<"MT")
```

1.4. "Se Entrega / Se Pide"

1.4.1. Se entrega

- conjunto.h Plantilla con la especificación del TDA conjunto.
- Función de abstracción e Invariante de respresentación del TDA conjunto.
- principal.cpp Plantilla del fichero con el main del programa. Este programa debe tomar como entrada el fichero de datos "clinvar_20160831.vcf", cargar las mutaciones en un conjunto de mutaciones y exhibir la funcionalidad del TDA Conjunto.

1.4.2. Se Pide

- conjunto.hxx Implementación del TDA conjunto.
- principal.cpp Completar su implementación.
- Analizar la eficiencia teórica y empírica de las operaciones de inserción, búsqueda y borrado en el conjunto.

1.5. "Fecha Límite de Entrega"

La fecha límite de entrega será el 6 de Noviembre a las 23:50 hrs.

2. Lista de tareas pendientes

Miembro main (int argc, char ∗argv[])

Analiza la eficiencia teórica y empírica de las operaciones find, insert y erase

3. Índice de clases

3.1. Lista de clases

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

4. Indice de archivos

4.1. Lista de archivos

Lista de todos los archivos con descripciones breves:

5. Documentación de las clases

5.1. Referencia de la Clase conjunto

Clase conjunto.

```
#include <conjunto.h>
```

Tipos públicos

- typedef mutacion value_type
- typedef unsigned int size_type
- typedef vector< mutacion >::iterator iterator
- typedef vector< mutacion >::const_iterator const_iterator

Métodos públicos

- conjunto ()

  constructor primitivo.
- conjunto (const conjunto &d)

  constructor de copia
- pair< conjunto::value_type, bool > find (const string &chr, const unsigned int &pos) const

  busca una entrada en el conjunto
- pair< conjunto::value_type, bool > find (const string &ID) const
- pair< conjunto::value_type, bool > find (const conjunto::value_type &e) const
- conjunto::size_type count (const string &chr, const unsigned int &pos) const

  cuenta cuantas entradas coinciden con los parámetros dados.
- conjunto::size_type count (const string &ID) const
- conjunto::size_type count (const conjunto::value_type &e) const
- bool insert (const conjunto::value_type &e)

  Inserta una entrada en el conjunto.
- bool erase (const string &chr, const unsigned int &pos)

  Borra una entrada en el conjunto . Busca la entrada con chr/pos o id en el conjunto (utiliza e.getID() en el tercer caso) y si la encuentra la borra.
- bool erase (const string &ID)
- bool erase (const conjunto::value_type &e)
- void clear ()

  Borra todas las entradas del conjunto, dejandolo vacio.
- conjunto::size_type size () const

  numero de entradas en el conjunto
- bool empty () const

  Chequea si el conjunto esta vacio (size()==0)
- conjunto & operator= (const conjunto &org)

  operador de asignación
- conjunto::iterator begin ()

  begin del conjunto
- conjunto::iterator end ()

  end del conjunto
- conjunto::const_iterator cbegin () const

  begin del conjunto
- conjunto::const_iterator cend () const

  end del conjunto
- conjunto::const_iterator lower_bound (const string &chr, const unsigned int &pos) const

  busca primer elemento por debajo ('antes', '<') de los parámetros dados.
- conjunto::const_iterator lower_bound (const conjunto::value_type &e) const
- conjunto::const_iterator upper_bound (const string &chr, const unsigned int &pos) const

  busca primer elemento por encima ('después', '>') de los parámetros dados.
- conjunto::const_iterator upper_bound (const conjunto::value_type &e) const

Métodos privados

- bool cheq_rep () const

  Chequea el Invariante de la representacion.

Atributos privados

- vector< mutacion > vm

5.1.1. Descripción detallada

Clase conjunto.

conjunto::conjunto, conjunto::find, conjunto::size, conjunto::count, conjunto::insert, conjunto::erase, conjunto::clear, conjunto::empty, conjunto::operator=, conjunto::begin, conjunto::end, conjunto::cbegin, conjunto::cend, conjunto::lower_bound, conjunto::upper_bound
Tipos conjunto::value_type, conjunto::size_type
Descripción: Un conjunto es un contenedor que permite almacenar en orden creciente un conjunto de elementos no repetidos. En nuestro caso el conjunto va a tener un subconjunto restringido de métodos (inserción de elementos, consulta de un elemento, etc). Este conjunto "simulará" un conjunto de la stl, con algunas claras diferencias pues, entre otros, no estará dotado de la capacidad de iterar (recorrer) a través de sus elementos.

Asociado al conjunto, tendremos el tipo

```
conjunto::value_type
```

que permite hacer referencia al elemento almacenados en cada una de las posiciones del conjunto, en nuestro caso mutaciones (SNPs). Es requisito que el tipo conjunto::value_type tenga definidos los operadores operator< y operator= .

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

Autor

Elena María Gómez Ríos

5.1.2. Documentación de los 'Typedef' miembros de la clase

5.1.2.1. typedef vector<mutacion>::const_iterator conjunto::const_iterator

5.1.2.2. typedef vector<mutacion>::iterator conjunto::iterator

5.1.2.3. typedef unsigned int conjunto::size_type

5.1.2.4. typedef mutacion conjunto::value_type

5.1.3. Documentación del constructor y destructor

5.1.3.1. conjunto::conjunto ( )

constructor primitivo.

fichero de implementacion de la clase conjunto

5.1.3.2. conjunto::conjunto (const conjunto & *d*)

constructor de copia

Parámetros

in	*d*	conjunto a copiar

5.1.4. Documentación de las funciones miembro

5.1.4.1. conjunto::iterator conjunto::begin ( )

begin del conjunto

Devuelve

Devuelve un iterador al primer elemento del conjunto. Si no existe devuelve end

Postcondición

no modifica el conjunto.

5.1.4.2. conjunto::const_iterator conjunto::cbegin ( ) const

begin del conjunto

Devuelve

Devuelve un iterador constante al primer elemento del conjunto. Si no existe devuelve end

Postcondición

no modifica el conjunto.

5.1.4.3. conjunto::const_iterator conjunto::cend ( ) const

end del conjunto

Devuelve

Devuelve un iterador constante al final del conjunto (posicion siguiente al ultimo.

Postcondición

no modifica el conjunto.

5.1.4.4. bool conjunto::cheq_rep ( ) const [private]

Chequea el Invariante de la representacion.

Invariante

IR: rep ==> bool

- Para todo i, 0 <= i < vm.size() se cumple
  - vm[i].chr está en ("1", "2", "3", "4", "5", "6", "7", "8", "9", "10", "11", "12", "13", "14", "15", "16", "17", "18", "19", "20", "21", "22", "X", "Y", "MT")
  - vm[i].pos > 0;
- Para todo i, 0 <= i < C.vm.size()-1 se cumple: a) si vm[i].chr == vm[i+1].chr, entonces: vm[i].pos < vm[i+1].pos b) si vm[i].chr != vm[i+1].chr, entonces vm[i].chr < vm[i+1].chr (donde el orden para el número de cromosoma se rige por "1"<"2"<"3"<...<"22"<"X"<"Y"<"MT")

  Devuelve

  true si el invariante es correcto, falso en caso contrario

5.1.4.5. void conjunto::clear ( )

Borra todas las entradas del conjunto, dejandolo vacio.

Postcondición

El conjunto se modifica, quedando vacio.

5.1.4.6. conjunto::size_type conjunto::count (const string & *chr,* const unsigned int & *pos*) const

cuenta cuantas entradas coinciden con los parámetros dados.

Parámetros

in	*chr*	de la mutación.
in	*pos*	de la mutación.
in	*ID*	de la mutación.
in	*e*	entrada. Utilizar e.getID() para buscar cuántas mutaciones tienen el mismo ID, el resto de los valores de entrada no son tenidos en cuenta

Devuelve

Como el conjunto de mutaciones no puede tener entradas repetidas, devuelve 1 (si se encuentra la entrada) o 0 (si no se encuentra).

Postcondición

no modifica el conjunto.

5.1.4.7. conjunto::size_type conjunto::count (const string & *ID*) const

5.1.4.8. conjunto::size_type conjunto::count (const conjunto::value_type & *e*) const

5.1.4.9. bool conjunto::empty ( ) const

Chequea si el conjunto esta vacio (size()==0)

Postcondición

No se modifica el conjunto.

5.1.4.10. conjunto::iterator conjunto::end ( )

end del conjunto

Devuelve

Devuelve un iterador al final del conjunto (posicion siguiente al ultimo.

Postcondición

no modifica el conjunto.

5.1.4.11. bool conjunto::erase (const string & *chr,* const unsigned int & *pos*)

Borra una entrada en el conjunto . Busca la entrada con chr/pos o id en el conjunto (utiliza e.getID() en el tercer caso) y si la encuentra la borra.

Parámetros

in	*chr*	de la mutación a borrar.
in	*pos*	de la mutación a borrar.
in	*ID*	de la mutación a borrar.
in	*e*	entrada con e.getID() que geremos borrar, el resto de los valores no son tenidos en cuenta

Devuelve

true si se ha eliminado la entrada.

Postcondición

Si esta en el conjunto su tamaño se decrementa en 1.

5.1.4.12. bool conjunto::erase (const string & *ID*)

5.1.4.13. bool conjunto::erase (const conjunto::value_type & *e*)

5.1.4.14. pair< conjunto::value_type, bool > conjunto::find (const string & *chr,* const unsigned int & *pos*) const

busca una entrada en el conjunto

Parámetros

in	*chr*	cromosoma de la mutación a buscar.
in	*pos*	posición en el cromosoma de la mutación.
in	*ID*	identificador de la mutación a buscar
in	*e*	entrada. Utilizar e.getID() o la combinación e.getChr()/e.getPos() para buscar una mutación con igual ID o Chr/Pos, el resto de los valores de entrada pueden ser ignorados.

Devuelve

Si existe una mutación en el conjunto con ese chr/pos o ID, respectivamente, devuelve un par con una copia de la mutación en el conjunto y con el segundo valor a true. Si no se encuentra, devuelve la mutación con la definicion por defecto y false

Postcondición

no modifica el conjunto.

```
Uso 1:
if (C.find("1", 6433456).second ==true) cout << "Found." ;
else cout << "Not found.";

Uso 2:
if (C.findID("rs12345").second ==true) cout << "Found." ;
else cout << "Not found.";
```

5.1.4.15. pair< conjunto::value_type, bool > conjunto::find (const string & *ID*) const

5.1.4.16. pair< conjunto::value_type, bool > conjunto::find (const conjunto::value_type & *e*) const

5.1.4.17. bool conjunto::insert (const conjunto::value_type & *e*)

Inserta una entrada en el conjunto.

Parámetros

e	entrada a insertar

Devuelve

true si la entrada se ha podido insertar con éxito, esto es, no existe una mutación con igual par chr/pos ni igual ID en el conjunto. False en caso contrario.

Postcondición

Si e no esta en el conjunto, el size() sera incrementado en 1.

5.1.4.18. conjunto::const_iterator conjunto::lower_bound (const string & *chr,* const unsigned int & *pos*) const

busca primer elemento por debajo ('antes', '<') de los parámetros dados.

Parámetros

in	*chr*	de la mutación.
in	*pos*	de la mutación.
in	*e*	entrada.

Devuelve

Devuelve un iterador al primer elemento que cumple que "elemento<e" es falso, esto es, el primer elemento que es mayor o igual que e

Si no existe devuelve end

Postcondición

no modifica el conjunto.

5.1.4.19. conjunto::const_iterator conjunto::lower_bound (const conjunto::value_type & *e*) const

5.1.4.20. conjunto & conjunto::operator= (const conjunto & *org*)

operador de asignación

Parámetros

in	*org*	conjunto a copiar.

Devuelve

Crea y devuelve un conjunto duplicado exacto de org.

5.1.4.21. conjunto::size_type conjunto::size ( ) const

numero de entradas en el conjunto

Postcondición

No se modifica el conjunto.

Devuelve

numero de entradas en el conjunto

5.1.4.22. conjunto::const_iterator conjunto::upper_bound (const string & *chr,* const unsigned int & *pos*) const

busca primer elemento por encima ('después', '>') de los parámetros dados.

Parámetros

in	*chr*	de la mutación.
in	*pos*	de la mutación.
in	*e*	entrada. Devuelve un iterador al primer elemento que cumple que "elemento>e", esto es, el primer elemento ESTRICTAMENTE mayor que e

Si no existe devuelve end

Postcondición

no modifica el conjunto.

5.1.4.23. conjunto::const_iterator conjunto::upper_bound (const conjunto::value_type & *e*) const

5.1.5. Documentación de los datos miembro

5.1.5.1. vector<mutacion> conjunto::vm [private]

Vector que almacena los elementos del conjunto

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

- include/conjunto.h
- include/conjunto.hxx

6. Documentación de archivos

6.1. Referencia del Archivo documentacion.dox

6.2. Referencia del Archivo include/conjunto.h

```
#include <string>
#include <vector>
#include <iostream>
#include "mutacion.h"
#include "conjunto.hxx"
```

Clases

- class conjunto
  Clase conjunto.

Funciones

- ostream & operator<< (ostream &sal, const conjunto &C)
  imprime todas las entradas del conjunto

6.2.1. Documentación de las funciones

6.2.1.1. ostream& operator<< (ostream & *sal,* const conjunto & *C*)

imprime todas las entradas del conjunto

Postcondición

No se modifica el conjunto. Implementar tambien esta funcion

6.3. Referencia del Archivo include/conjunto.hxx

Funciones

- ostream & operator<< (ostream &sal, const conjunto &C)
  imprime todas las entradas del conjunto

6.3.1. Documentación de las funciones

6.3.1.1. ostream& operator<< (ostream & *sal,* const conjunto & *C*)

imprime todas las entradas del conjunto

Postcondición

No se modifica el conjunto. Implementar tambien esta funcion

6.4. Referencia del Archivo src/principal.cpp

```
#include "mutacion.h"
#include "enfermedad.h"
#include "conjunto.h"
#include <iostream>
#include <fstream>
#include <vector>
#include <algorithm>
#include <random>
#include <chrono>
```

Funciones

- bool load (conjunto &cm, const string &s)

  lee un fichero de mutaciones, linea a linea
- int main (int argc, char ∗argv[ ])

6.4.1. Documentación de las funciones

6.4.1.1. bool load (conjunto & *cm,* const string & *s*)

lee un fichero de mutaciones, linea a linea

Parámetros

in	*s*	nombre del fichero
in,out	*cm*	objeto tipo conjunto sobre el que se almacenan las mutaciones

Devuelve

true si la lectura ha sido correcta, false en caso contrario

6.4.1.2. int main (int *argc,* char ∗ *argv[ ]*)

Tareas pendientes Analiza la eficiencia teórica y empírica de las operaciones find, insert y erase

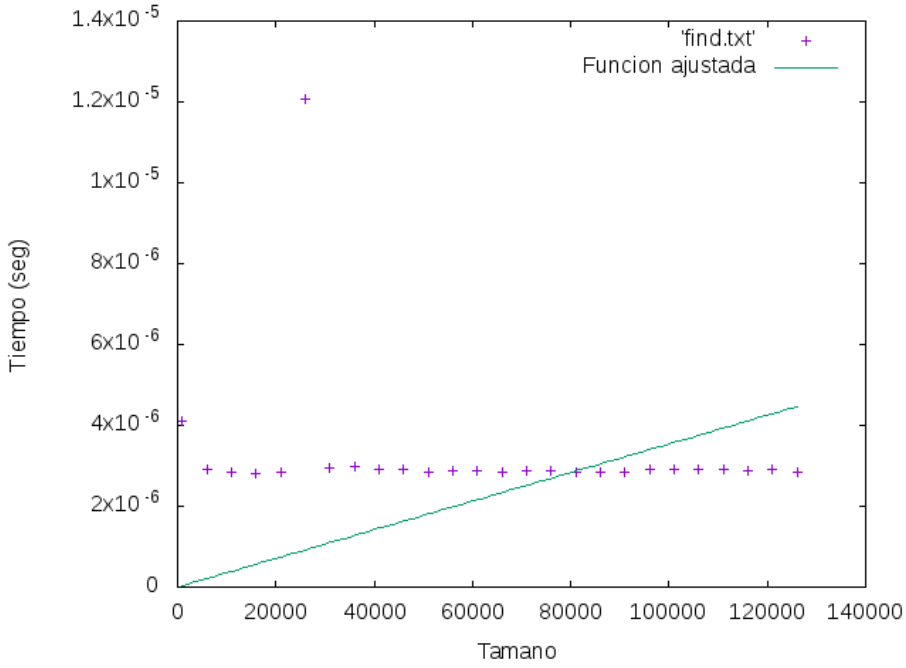
7. Cálculo de eficiencia

7.1. Método find

Vamos a calcular la eficiencia del método `find` de la `clase conjunto`:

```
pair<conjunto::value_type,bool>  conjunto::find (const string & ID) const{
   bool encontrado = false; // O(1)
   mutacion mutfind; // O(1)
   const_iterator it = cbegin(); // O(1)

   while (it != cend() && !encontrado){ // O(n)
      if ( (*it).getID()==ID ){ // O(1)
         mutfind = (*it); // O(1)
         encontrado = true; // O(1)
      }
      it++; // O(1)
   }

   const pair<conjunto::value_type,bool> res (mutfind, encontrado); // O(1)
   return res; // O(1)
}
```

El método `find` tiene eficiencia $O(n)$ como podemos observar en el código anterior.



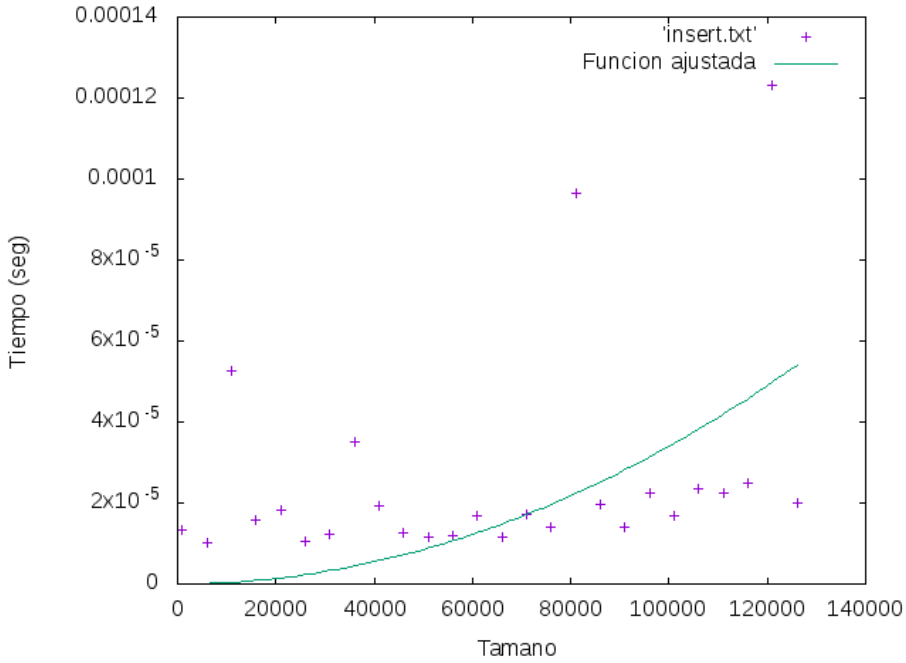
7.2. Método insert

Vamos a calcular la eficiencia del método `insert` de la `clase conjunto`:

```
bool conjunto::insert( const conjunto::value_type & e){
   bool encontrado = false; // O(1)

   if (empty()){
      vm.push_back(e); // O(1) en tiempo amortizado
   }else{
      encontrado = find(e).second; // O(n)

      if (!encontrado){
         vm.insert(lower_bound(e), e); //O(n^2)
      }
   }

   return !encontrado;
}


conjunto::const_iterator conjunto::lower_bound (const conjunto::value_type & e) const{
   const_iterator it = cbegin(); // O(1)
   bool encontrado = false; // O(1)

   while(it != cend() && !encontrado){ // O(n)
      if ((*it)<e )
         it++; // O(1)
      else
         encontrado = true; // O(1)
   }
```

```
   return it;
}
```

El método `insert` tiene eficiencia $O(n^2)$ como podemos observar en el código anterior.

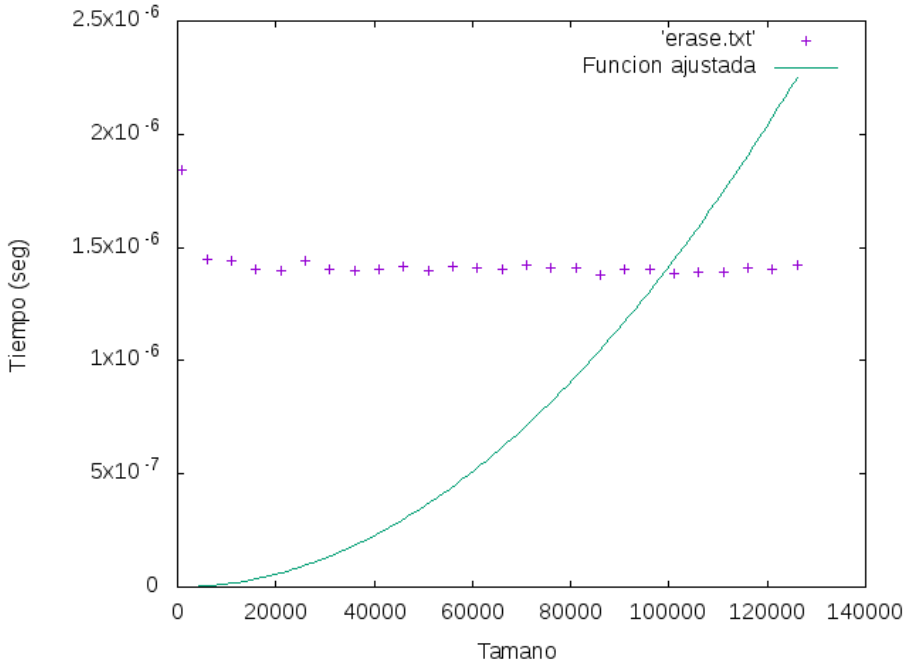


7.3. Método erase

Vamos a calcular la eficiencia del método `find` de la `clase conjunto`:

```
bool conjunto::erase(const string & ID){
   bool encontrado = false; // O(1)
   iterator it = begin(); // O(1)

   while (it != end() && !encontrado){ // O(n^2)
      if ( (*it).getID()==ID ){ // O(n)
         vm.erase(it); // O(n)
         encontrado = true; // O(1)
      }
      it++; // O(1)
   }

   return encontrado; // O(1)
}
```

El método `find` tiene eficiencia $O(n^2)$ como podemos observar en el código anterior.


2.5x10 -6
2x10 -6
1.5x10 -6
1x10 -6
5x10 -7
0
Tiempo (seg)
0
20000
40000
60000
80000
100000
120000
140000
Tamano
'erase.txt'
Funcion ajustada

Índice alfabético