Rendimiento de TDA

José Alberto Pejuán Martínez

11411073

Resumen

Por medio del uso del método *clock* perteneciente al archivo de cabecera <time.h> se puede conocer cuánto tarda un proceso cualquiera. Lo anterior es útil si se desea determinar qué proceso es más eficaz que otro. Esta investigación pretende averiguar que TDALista contiene los métodos más eficaces y por ende evaluar la mejor opción a utilizar.

Cada estructura es distinta, alguna puede ser más sencilla de programar pero dolorosa y tardía al ejecutarse, o por el contrario, dolorosa de programar y rápida al ejecutarse, es por eso que se necesita una prueba concreta.

Introducción

Al utilizar una computadora y más específicamente un programa, muchas veces se le da prioridad a la velocidad con la cual se realizan las tareas. Claro está que cada tarea es distinta y requieren más o menos tiempo según lo demanden. En la clase de *Estructura de Datos* se estudian los tipos de datos abstractos (listas, colas, pilas, árboles, etc.), cada uno de estos datos abstractos son diferentes, pero tienen en común varios métodos que todo TDA debe de tener. Los métodos en común son: init, insert, remove, clear, indexOf, first, last y get. Uno podría pensar que dado que estos TDA son muy parecidos, no importaría cual decidiéramos utilizar si es que no necesitamos métodos específicos de un solo TDA. La diferencia radica en el rendimiento y velocidad de cada uno de los métodos de estos tipos de dato abstracto. Un programador debe ser preocupado por el rendimiento de su software, más aún si pretende comercializarlo o de igual forma si simplemente alguien además de él lo utilizará. Mediante la utilización de la función clock() del archivo de cabecera <time.h>, se medirá el tiempo que tarda cada método en común de una a cinco mil veces y para tres distintos tipos de TDALista. Los tipos de datos abstractos que se utilizarán son el Fixed Size ArrayList, Double Cursor y Simple Linked List. Se busca encontrar las fortalezas y debilidades de cada uno de estos TDA para poder explotar su capacidad en un futuro.

Marco teórico

Un tipo de dato abstracto o TDA, es una estructura definida por el programador de modo que se pueda manipular de igual forma que a los tipos de dato incluidas en un sistema (Universidad de Jaen, 2009). Los TDA contienen métodos tales como los mencionados en la introducción que son indispensables al momento de programar. Interesa saber de qué manera funcionan los distintos TDALista que se utilizará en esta investigación y hacer un análisis previo a tomar los tiempos de cada método.

Definición de lista: Es una serie finita de cero o más elementos enlazados entre ellos. (Señas, 2006)

FSArrayList:

Las siglas FS, en el nombre de este tipo de dato abstracto, significan “Fixed Size”, que al español se traduce como “Tamaño Fijo”. Este TDA es una lista con una capacidad máxima la cual es decidida por el usuario, puede tener uno o dos punteros. La FSArrayList a utilizar es una lista doblemente enlazada con tamaño fijo. Las listas doblemente enlazadas contienen dos punteros, uno que hace referencia al nodo siguiente y otro al anterior, haciendo posible recorrer la lista en ambos sentidos (Con Clase, 2009). El primer nodo de la lista en este TDA tiene su puntero de anterior apuntando a NULL y el último nodo tiene su puntero de siguiente también en NULL.

```
#include "tdalist.h"
#include "object.h"

class fsarrayList : public TDAList
{
 private:
    unsigned int capacity;
    Object** data;
 public:
    fsarrayList(int);
    virtual ~fsarrayList();
    virtual bool insert(Object*,int);
    virtual Object* remove(unsigned int);
    virtual Object* first() const;
    virtual Object* last() const;
    virtual int getCapacity() const;
    virtual bool isFull() const;
    virtual void clear();
    virtual Object* get(unsigned int) const;
    virtual void print()const;
    virtual int indexOf(Object* E) const ;
    virtual int prev(int) const;
    virtual int next(int) const;

};
```

La lista contiene un atributo de tipo unsigned int llamado capacity que es el maximo numero de nodos que la lista puede soportar. Sus métodos semi-exclusivos son el getCapacity() y los metodos de recorrido prev() y next().

SLinkedList:

En el caso de la SLinkedList, cuya letra 'S' al principio de su nombre significa "simple", solamente contiene un puntero el cual hace referencia al nodo next o siguiente. Este tipo de lista nada más puede recorrerse en un sentido y al eliminarse la lista es posible que tome menos tiempo ya que solo debe anular un puntero. En este TDA si el nodo es el último de la lista, el puntero siguiente toma valor NULL (Con Clase, 2009).

Esta clase tiene como atributo propio un nodo* llamado head. Head es el primer nodo de la lista y al incializar la clase este es NULL. Aunque la clase contiene un metodo prev y next estos no son los que enlazan y no deben ser confundidos con los métodos getNext y getPrev de la clase nodo que utiliza FSArrayList para enlazarse. SLinkedList utiliza un nodo propio el cual como se dijo anteriormente solo tiene un puntero apuntando al siguiente.

```
class SLinkedList : public TDAList {
protected:
    SLNode* head;
public:
    SLinkedList();
    virtual ~SLinkedList();
    virtual bool insert(Object*,int);
    virtual int indexOf(Object*)const;
    virtual Object* get(unsigned)const;
    virtual Object* remove(unsigned);
    virtual int prev(int)const;
    virtual int next(int)const;
    virtual void clear();
    virtual Object* first()const;
    virtual Object* last()const;
    virtual void print()const;
    virtual bool isEmpty()const;
    virtual bool isFull()const;
};
```

Cursor:

```
class cursor : public TDAList
{
    struct Registry
    {
        int prev, next;
        Object* datum;
    };

    Registry* map;
    int head;
    int capacity;
    int findNextSlot() const;

public:
    cursor(int=25);
    ~cursor();
    virtual bool insert(Object*, int);
    virtual int indexOf(Object*) const;
    virtual Object* get(unsigned) const;
    virtual Object* remove(unsigned);
    virtual int prev(int) const;
    virtual int next(int) const;
    virtual void clear();
    virtual Object* first() const;
    virtual Object* last() const;
    virtual void print() const;
    virtual bool isFull() const;
};
```

Un cursor es utilizado más que nada en lenguajes de programación que no contienen punteros. Lenguajes como Java y C++ pueden dispensar de los cursores ya que C++ contiene punteros y Java tiene referencias. En esta implementación de cursor se tiene un método exclusivo de la clase cursor llamado findNextSlot. Un cursor simula una lista doblemente enlazada en este caso y la diferencia radica en que en lugar de punteros utiliza un entero que determina su posición y métodos que recorren la lista usando la posición y el next, el cual también es un entero, de cada elemento de la lista.

Metodología

1) Implementar los tres distintos TDALista que se utilizaran: FSArrayList, SLinkedList y Cursor.
2) En el main del proyecto se debe incluir los archivos de cabecera <stdio.h> y <time.h>.
3) Hacer un ciclo ***for*** que crezca de 0 a 5000.
4) Usando la función clock() medir el tiempo de los métodos: insert, indexOf, clear, remove, get, first y last para las tres distintas TDALista. Se deben hacer 5000 checkpoints de clock para cada método e imprimir en pantalla los clicks que realizó el sistema al ejecutar el programa.
5) Introducir los datos a tablas de Excel y comparar el tiempo de cada método y para cada TDALista.

Precauciones a tomar:

- Asegurarse de hacer la impresión de los tiempos en una pantalla que permita visualizar los 5000 checkpoints.
- Cerrar todo programa abierto que no sea del software que se creó.
- Para mejores resultados no realizar la prueba en un sistema virtualizado.
- Deshabilitar el internet de la máquina.
- Realizar los llamados de los métodos de la misma forma en el main solamente cambiando el tipo de TDALista al cual se le realiza dicho método.
- Solo tomar el tiempo del método que se quiere medir y no de sus preparaciones.

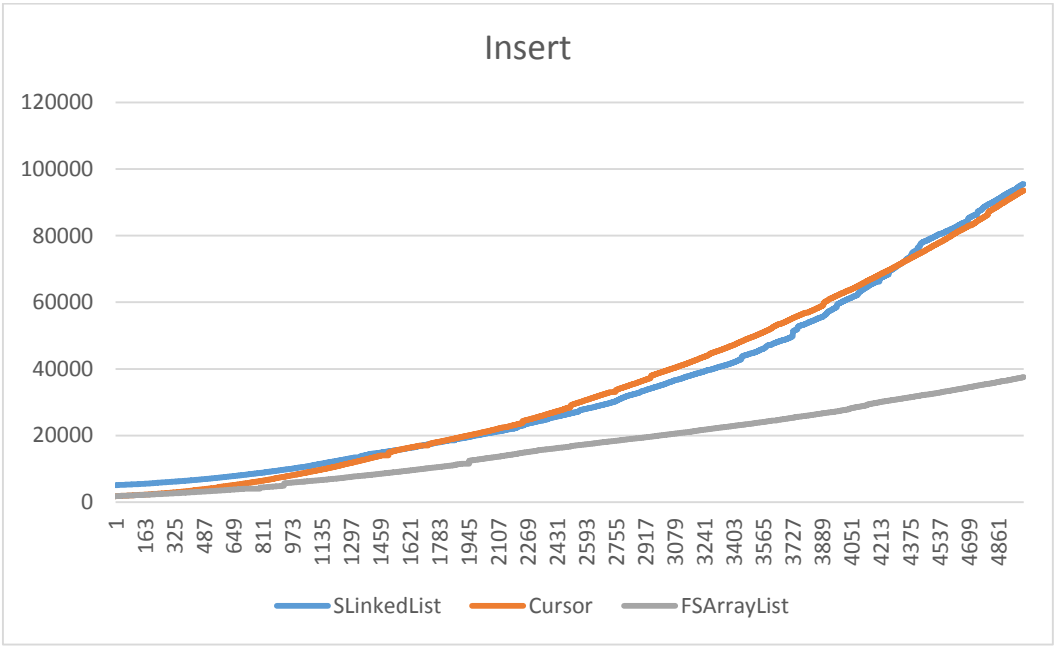
Resultados

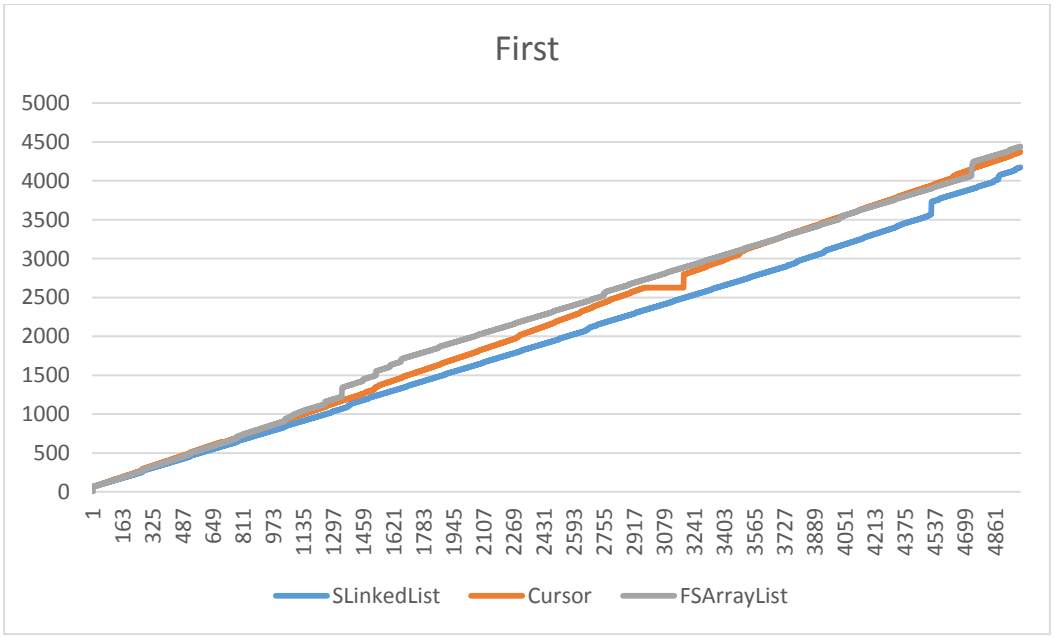
A continuación se presentan las gráficas de crecimiento de los click del sistema mientras la lista realizaba sus métodos n cantidad de veces. Cabe resaltar que se tomó el tiempo con el siguiente código base habiendo hecho los requerimientos que necesita cada proceso antes de ingresar al ciclo:

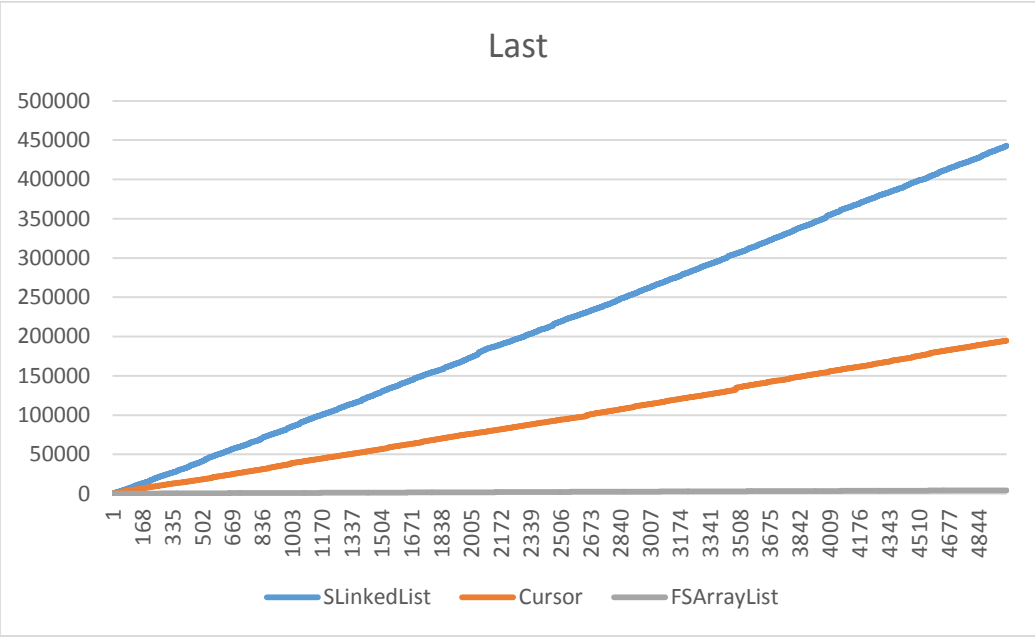
```
Clock_t initial = clock();

Clock_t final;

for(int i=0; i < 5000; i++){

        lista->método();

        final=clock()-initial;

        cout<<final<<" ";

}
```

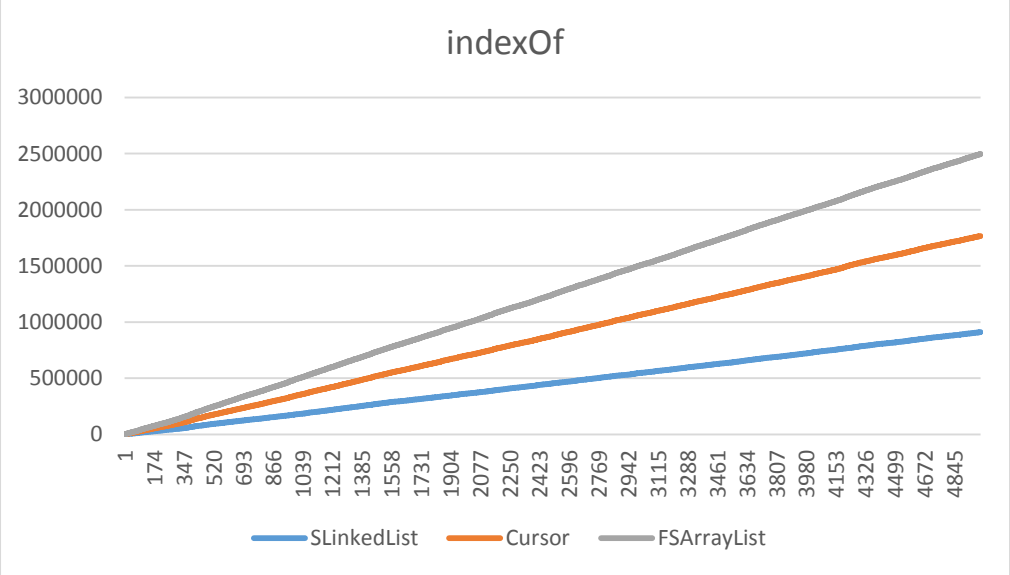
El significado de este código es que solo se toma el tiempo a partir que empieza a realizarse el método que vamos a medir (ultima precaución). El código también implica que cada ciclo se marca el tiempo que tomó realizarse el método número **i** y todos los anteriores.

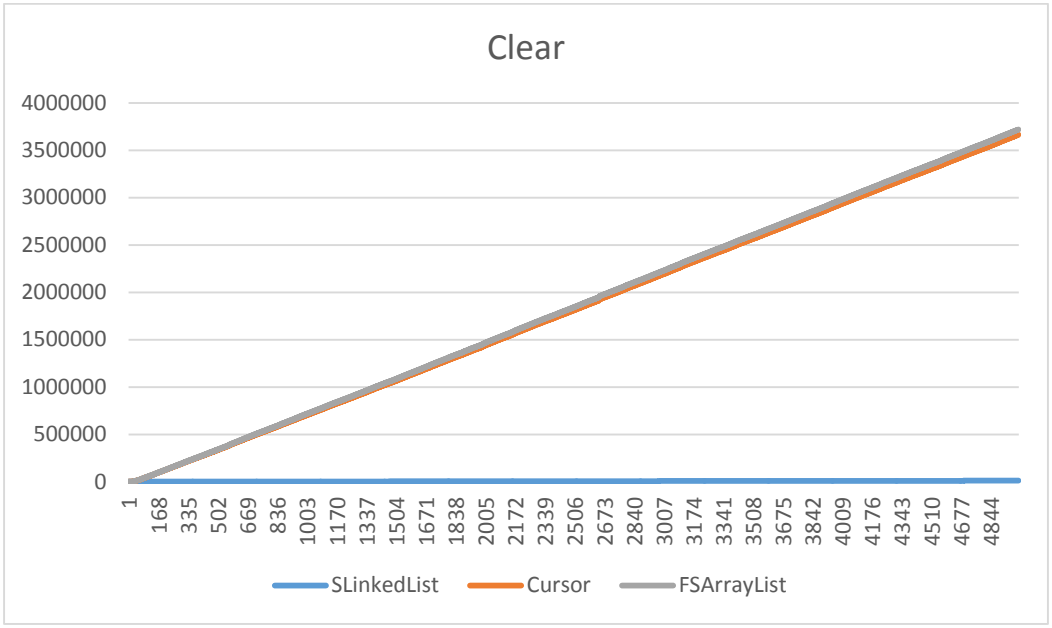
Gráficas:

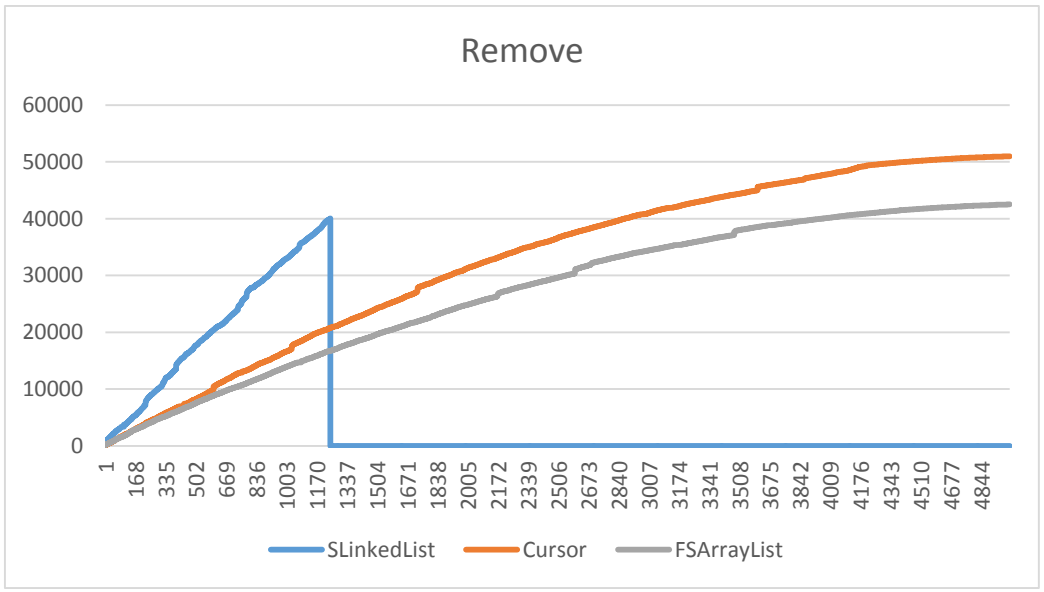

Insert
120000
100000
80000
60000
40000
20000
0
1
163
325
487
649
811
973
1135
1297
1459
1621
1783
1945
2107
2269
2431
2593
2755
2917
3079
3241
3403
3565
3727
3889
4051
4213
4375
4537
4699
4861
SLinkedList
Cursor
FSArrayList

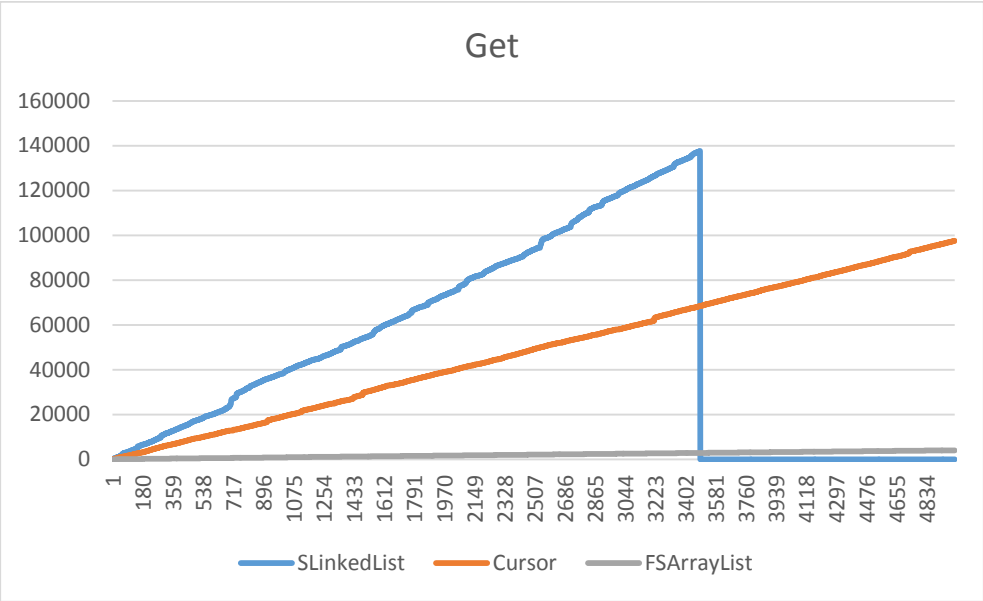

First
5000
4500
4000
3500
3000
2500
2000
1500
1000
500
0
1
163
325
487
649
811
973
1135
1297
1459
1621
1783
1945
2107
2269
2431
2593
2755
2917
3079
3241
3403
3565
3727
3889
4051
4213
4375
4537
4699
4861
SLinkedList
Cursor
FSArrayList


Last
500000
450000
400000
350000
300000
250000
200000
150000
100000
50000
0
1
168
335
502
669
836
1003
1170
1337
1504
1671
1838
2005
2172
2339
2506
2673
2840
3007
3174
3341
3508
3675
3842
4009
4176
4343
4510
4677
4844
SLinkedList
Cursor
FSArrayList


indexOf
3000000
2500000
2000000
1500000
1000000
500000
0
1
174
347
520
693
866
1039
1212
1385
1558
1731
1904
2077
2250
2423
2596
2769
2942
3115
3288
3461
3634
3807
3980
4153
4326
4499
4672
4845
SLinkedList
Cursor
FSArrayList


Clear
4000000
3500000
3000000
2500000
2000000
1500000
1000000
500000
0
1
168
335
502
669
836
1003
1170
1337
1504
1671
1838
2005
2172
2339
2506
2673
2840
3007
3174
3341
3508
3675
3842
4009
4176
4343
4510
4677
4844
SLinkedList
Cursor
FSArrayList


Remove
60000
50000
40000
30000
20000
10000
0
1
168
335
502
669
836
1003
1170
1337
1504
1671
1838
2005
2172
2339
2506
2673
2840
3007
3174
3341
3508
3675
3842
4009
4176
4343
4510
4677
4844
SLinkedList
Cursor
FSArrayList



Conclusión

Para determinar que TDALista hay que analizar cuál de todos fue más eficiente en cada uno de los métodos. En el método de **insert** el TDALista más eficaz fue el FSArrayList. La SLinkedList y el cursor hicieron casi 100000 clicks cada uno por cinco mil inserts. Por el otro lado el FSArrayList realizó los cinco mil insert en menos de 40000 clicks.

En cuanto al método de **first**, las tres gráficas mostraron números similares y es muy difícil decidir que lista es más rápida. Si se toman los resultados de la gráfica que se generó se podría concluir que el TDALista SLinkedList fue el más rápido de los tres aunque por un margen muy pequeño, a diferencia del método **last** en donde FSArrayList supera en eficacia a las demás listas por una diferencia abismal convirtiéndose en la mejor opción y SLinkedList la peor si es que del método **last** es el que se precisa.

El método **indexOf** muestra una diferencia con respecto a los gráficos anteriores ya que la lista más eficaz es SLinkedList y la más lenta FSArrayList, similar al clear en el cual SLinkedList muestra ser la opción a utilizar mientras que las otras dos se encuentran muy parecidas en numero clicks. Las últimas gráficas (remove y get) hacen más sencillo el trabajo analítico porque SLinkedList genera errores cuando llega a cierto número del ciclo mientras que la FSArrayList realiza el **get** en tiempo record y el **remove** un poco más rápido que Cursor (que realizó estos métodos sin pena ni gloria con un segundo lugar). Es evidente entonces que FSArrayList es la mejor alternativa si se desea utilizar una lista y SLinkedList la peor.

Bibliografía

Con Clase. (10 de Diciembre de 2009). *C con clase*. Recuperado el 25 de Octubre de 2014, de http://www.c.conclase.net/edd/

Señas, P. (2006). *Estructura de Datos y Algoritmos Módulo III.* Buenos Aires: Universidad Nacional del Sur.

Universidad de Jaen. (2 de Octubre de 2009). *LDC*. Recuperado el 26 de Octubre de 2014, de Tipos de Datos Abstractos: http://ldc.usb.ve/~gabro/teaching/CI2126/TADPilaLista.pdf