

Departamento de Informática y ciencias de la computación.

Universidad de Concepción.

Estructuras de datos:

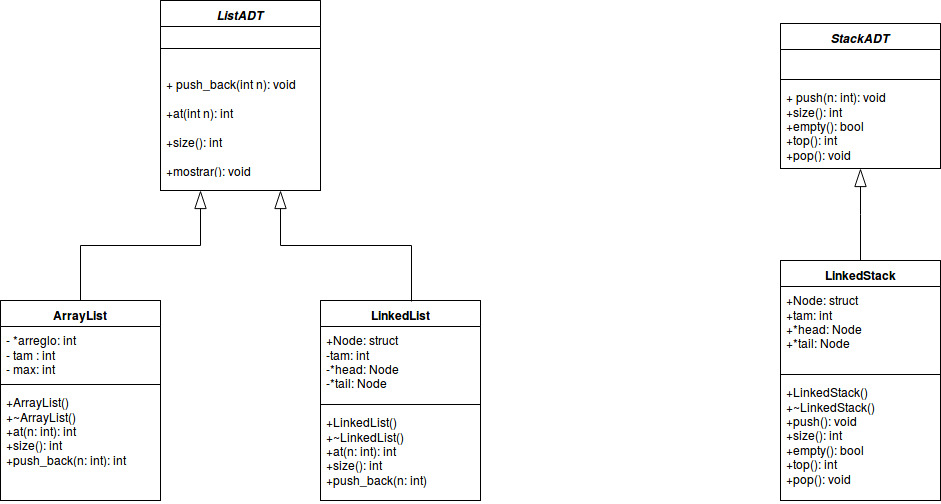
Proyecto I.

Integrantes:

Javier Arriagada I.

Lisette Morales G.

1. [1 punto] Cree una pequeña librería (entendida como conjunto de clases) que contenga los ADTList y Stack junto con algunas implementaciones de las mismas realizadas en los laboratorios (ArrayList y LinkedList para List) o vistas en clase (Stack basado en la LinkedList anterior). Adjunte un fichero test lib.cpp que permita probar la funcionalidad de la librería. Documente bien ésta librería ya que se podrá requerir más adelante en futuros laboratorios y mini-proyectos. Específicamente, se debe incluir un diagrama de clases UML y se debe indicar la complejidad de cada operación en notación asintótica.



ArrayList LinkedList

push\_back(int n) O(1) O(1)

at(int n) O(1) O(n)

size() O(1) O(1)

LinkedStack

Push () O(1)

Empty () O(1)

Size () O(1)

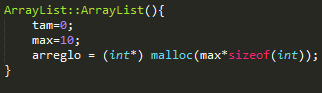
Top () O(1)

Pop () O(n)

Para el ArrayList:

En el constructor se inicializa una variable int tam, que indicará el tamaño actual de la lista.

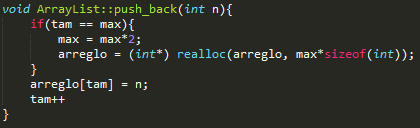
Int max, indicará un valor máximo para así poder pedir memoria con el tamaño de esta variable



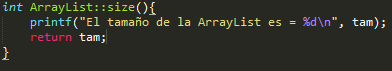
El destructor liberarà toda la lista de la memoria



Método push\_back(int n). Recibe como parámetro una variable entera y la inserta en el arreglo dinámico y aumenta la variable tam en 1. En el caso de que tam sea igual a la variable max previamente definida, esto quiere decir que el arreglo dinámico está lleno, por lo que se pedirá un realloc para aumentar su tamaño al doble



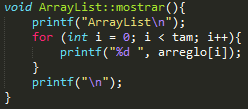
Metodo size(). Devuelve la variable tam



Metodo at(int n). Devuelve el elemento en la posición pedida



Se creó un método adicional para poder imprimir por pantalla los arreglos almacenados

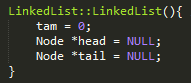


Para LinkedList

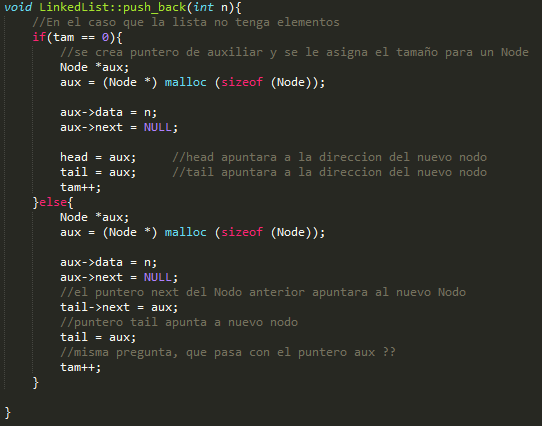
Se define previamente unstructo con las siguientes características. Almacenará un entero y tendrá un puntero \*next que apunta a un Node



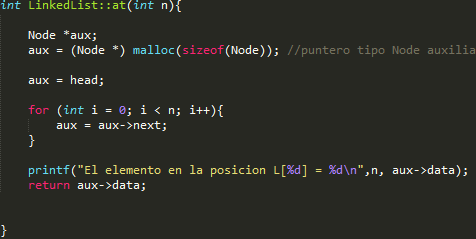
El constructor, inicializa una variable int tam a 0, esto indicará el tamaño de la lista y se inicializan 2 punteros a Node, que se encargaran de apuntar al primer elemento de la lista(head) y al último(tail)



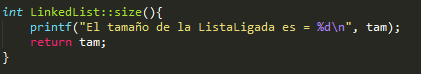
Metodo push\_back(int n). En el caso que la lista esté vacía,se creará un puntero a Node auxiliar, a este Node se le asigna a la variable data el valor ingresado y su puntero next apuntara a NULL. Luego head y tail apuntan a ese nuevo Node. Finalmente se aumenta la variable tam en 1.  
En el caso que no este vacìa pasara lo mismo, pero el punto next del ultimo Node referenciado con el puntero tail, apuntara al Node auxiliar y el puntero tail ahora apuntara a el nuevo Node. Finalmente se aumenta la variable tam en 1.



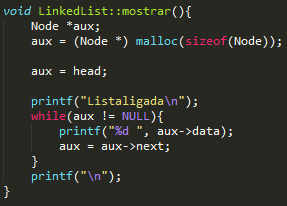
Metodo at(int n). Se crea un puntero auxiliar de tipo Node, Luego se recorre la lista ligada n veces y se devolverá el valor del Node buscado.



Metodo size(). Devolvera la variable tam



Se creó un método adicional para poder imprimir por pantalla los arreglos almacenados

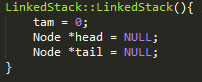


LinkedStack

Se define previamente unstructo con las siguientes características. Almacenará un entero y tendrá un puntero \*next que apunta a un Node

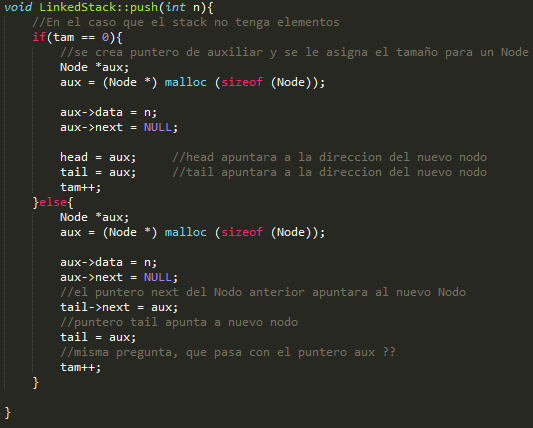


El constructor, inicializa una variable int tam a 0, esto indicará el tamaño de la lista y se inicializan 2 punteros a Node, que se encargaran de apuntar al primer elemento de la lista(head) y al último(tail)

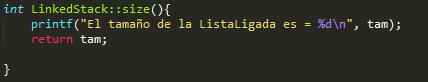


Metodo push(int n). En el caso que la lista esté vacía,se creará un puntero a Node auxiliar, a este Node se le asigna a la variable data el valor ingresado y su puntero next apuntara a NULL. Luego head y tail apuntan a ese nuevo Node. Finalmente se aumenta la variable tam en 1.

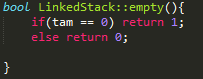
En el caso que no este vacìa pasara lo mismo, pero el punto next del ultimo Node referenciado con el puntero tail, apuntara al Node auxiliar y el puntero tail ahora apuntara a el nuevo Node. Finalmente se aumenta la variable tam en 1.



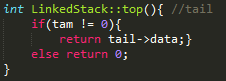
Metodo size(). Devuelve la variable tam



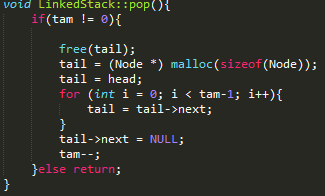
Metodo empty(). Devuelve 1 si esta vacío els tack y 0 si es que no lo esta



Metodo top(). Devuleve el valor de la variable data del último node referenciado por el puntero tail



Metodo pop(). Si el stack no está vacío se libera la memoria del último elemento. Luego se recorre el stack tam-1 veces para llegar al nuevo último Node, este node se apuntara con un puntero tail y el puntero next de este nodo apuntara a NULL. Luego la variable tam se le resta 1. Si esta vacio el método no hace nada



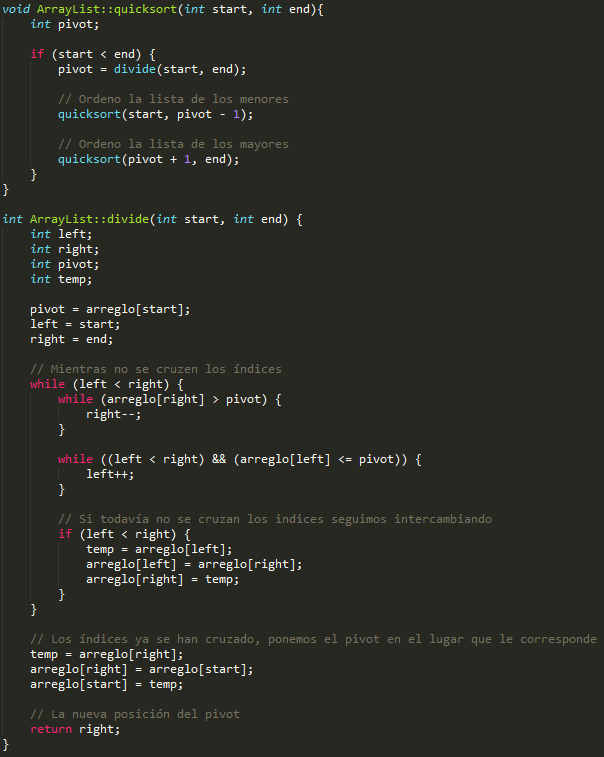
2. Además de los algoritmos de búsqueda lineal y búsqueda binaria vistos en el laboratorio , existen otros algoritmos de búsqueda como la búsqueda exponencial (también conocida como doblada o galopante). Puede encontrar la descripción de la misma en https://en.wikipedia.org/wiki/Exponential\_search. En este ejercicio se le pide implementar dicho algoritmo y compararlo con los desarrollados en el laboratorio 2. Todos los algoritmos se deben adaptar para trabajar sobre el ArrayList implementado en el ejercicio anterior. La comparación se hará experimentalmente y deberá entregar los siguientes gráficos de tiempo de ejecución de los algoritmos para diferentes tamaños de entrada.

a) Se tomarán tiempos de búsqueda de elementos aleatorios.

b) Los elementos buscados siempre se encuentran entre los primeros (10 %).c) Los elementos buscados siempre son mayores que el mayor de los elementos.

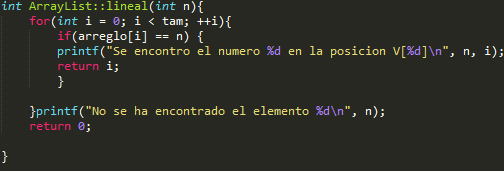
Comente acerca de los resultados obtenidos.

Para este ejercicio se agregaron metodos de busqueda, de los cuales algunos funcionan solo si la lista está ordenada de forma ascendente. Se crearon 2 métodos adicionales para asi poder ordenar las listas. Estos métodos están basados en el algoritmo quicksort



A continuación se adjuntan los métodos de búsqueda

Método lineal(int n). Este método recorre la lista completa de principio a fin hasta encontrar el elemento solicitado

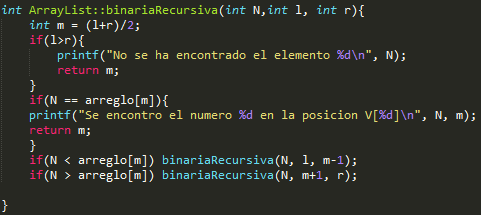


Metodo binariaRecursiva(). Este método recibe 3 parámetros, el valor a buscar, el límite izquierdo y el límite derecho.

Primero se crea una variable m que vendría siendo la mitad entre los límites.

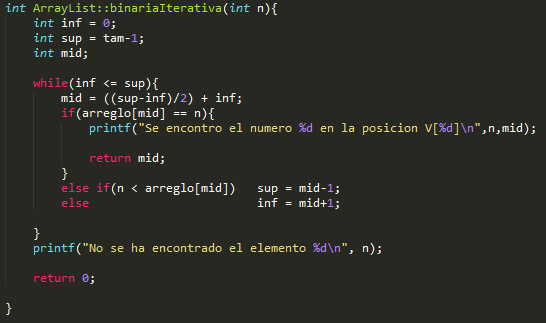
Esta recursividad tiene 2 casos base. Primero si el límite izquierdo es mayor que el derecho, el elemento no se encontró. Tambien si el valor buscado se encuentra en la posición m, devuelve la posición

En caso contrario la función se llamara a si misma cambiando los limites

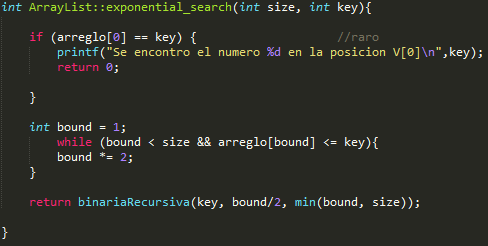


Metodo binariaIterativa(int n). En este método se crean 3 variables, inf que indicará el límite inferior, sup que indicará el límite superior y mid que indicará la mitad de la lista.

Mientras inf sea menor o igual a sup, se vuelve a calcular mid con la nueva mitad y se pregunta si el valor buscado coincide con la posición mid, en caso contrario se vuelven a definir los límites.

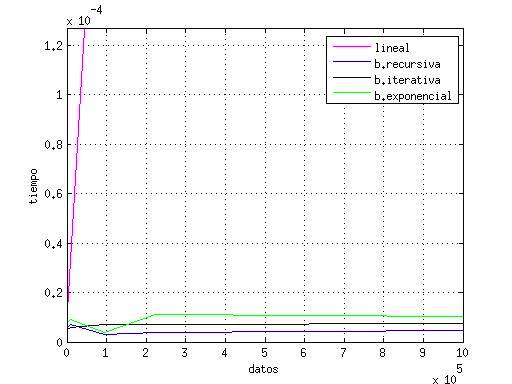


Metodo exponential\_search(int size, int key). Este método primero pregunta si la key buscada está en la primera posición. Si no lo esta crea una variable bound igual a 1. Luego mientras bound sea menor al tamaño de la lista y el elemento en la posición bound de la lista sea menor o igual a la key buscada, bound se multiplicará por 2. La funcion llamara a la funcion bianriaRecursiva con los parametros ( key a buscar, bound/2, min(bound,size)) y retornara lo que esta funcion retorne



Gráficos:

1. búsqueda de elementos aleatorios.



datos Lineal br bi es

883 0.000012 0.000006 0.000005 0.000008

9830 0.000033 0.000007 0.000006 0.000009

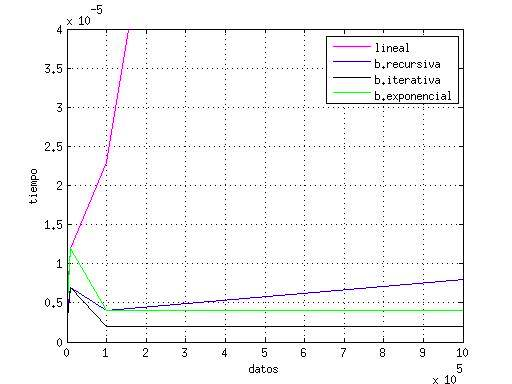
95825 0.000269 0.000003 0.000007 0.000004

224340 0.001012 0.000004 0.000007 0.000011

1487993 0.002218 0.000005 0.000008 0.000010

6599482 0.029793 0.000008 0.000003 0.000007

b)



datos Lineal br bi es

100 0.000010 0.000014 0.000013 0.000025

1000 0.000006 0.000004 0.000003 0.000005

10000 0.000012 0.000007 0.000007 0.000012

100000 0.000023 0.000004 0.000002 0.000004

1000000 0.000303 0.000008 0.000002 0.000004

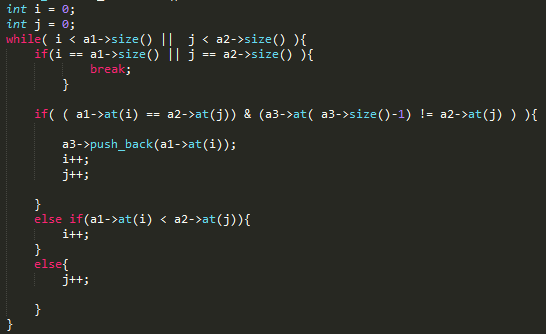
10000000 0.003602 0.000008 0.000005 0.000011

3. Dadas dos listas ordenadas, una operación interesante es aquella que computa la intersección de las mismas y la retorna en una nueva lista también ordenada. Por ejemplo, dadas las listas A =3− > 7− > 10 y B = 1− > 7− > 10− > 12− > 20; la lista resultado serı́a C = 7− > 10.

Nota: Para ambos códigos se agrego una condición que discrimine los números repetidos de las listas. (si por ejemplo ambas listas tienen dos veces 5, ese 5 solo se copiará 1 vez en la lista resultado)

Para este ejercicio se implementaron 2 códigos.

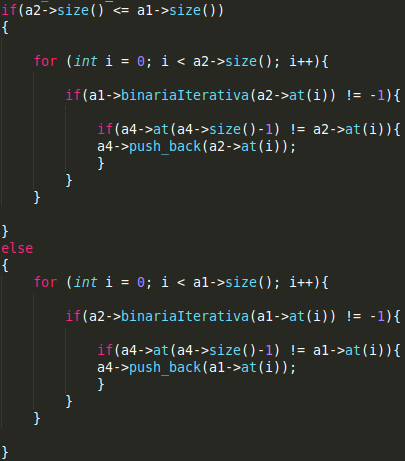
Versión 1: se irán recorriendo las dos listas de forma sincronizada, copiando el menor de los elementos a la lista resultado y avanzando al siguiente elemento de la lista donde se encuentra dicho elemento (mientras que en la otra lista no se avanza)



Se recorren ambas listas simultáneamente y se comparan los valores, Si son iguales, se copia el elemento en una lista auxiliar y se avanza en ambas listas, si son distintos solo avanza la lista que tiene al elemento menor

Complejidad O(n\*m). Siendo n el largo del arreglo más corto(n), y m el del mas largo(m)

Versión 2: cada elemento de la lista más corta se buscará, mediante el algoritmo de búsqueda binaria, en la lista más larga. En caso de encontrarse, se añadirá a la lista resultado. En caso contrario, se omitirá y se pasará al siguiente elemento. (Si las dos listas tienen igual longitud, se tomará cualquiera de ellas como la menor).



Primero se identifica cual es la lista más corta, luego cada elemento de esta lista se busca en la segunda lista por Búsqueda binaria. Si se encuentra, este elemento es copiado en una lista auxiliar

Complejidad O(n) \* O(log m). Porque se recorrera el arreglo completo de la lista mam corta(n) y por cada n se hara una busqueda binaria iterativa en el arreglo mas largo(m).

Evaluación experimental

En el caso que n = m

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | n=m=100 | n=m=1000 | n=m=10000 | n=m=100000 |
| Version 1 | 0.000005 | 0.000053 | 0.000364 | 0.004431 |
| Version 2 | 0.000007 | 0.000042 | 0.000385 | 0.003634 |

En el caso que n != m

n = 100000

|  |  |  |
| --- | --- | --- |
|  | n/m = 1/2 | n/m = 1/4 |
| Version 1 | 0.007610 | 0.010964 |
| Version 2 | 0.003363 | 0.003299 |

c)

datos lineal br bi es

100 0.000005 0.000004 0.000003 0.000005

1000 0.000012 0.000004 0.000004 0.000006

10000 0.000299 0.000003 0.000005 0.000003

100000 0.000302 0.000003 0.000007 0.000008

1000000 0.004575 0.000004 0.000007 0.000010

10000000 0.045086 0.000005 0.000006 0.000010