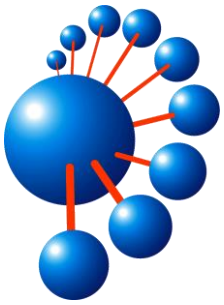


Estructura de datos

“SkipList VS LinkedList”

Estudiante: Roberto Ávila

Profesor: Diego Seco

Ayudante: Alexander Irribarra

Índice

Introducción

Hay situaciones donde necesitamos tener las herramientas correctas para resolver problemas de una forma rápida ,eficiente y sólida , en estructura de datos esto se traduce en conocer bien las estructuras para saber en qué momento usarlas , es por esa motivación que nos dedicaremos a analizar y comparar resultados entre dos estructuras que almacenan enteros , ambas entregan al cliente que las ocupa 3 operaciones básicas , insert(insertar entero) , remove (eliminar entero) , search(buscar entero) , ambas en esencia hacen lo mismo para el cliente, sin embargo el funcionamiento interno de cada una es distinta y en eso es en que los concentraremos , dando detalles de la implementación de cada una , poniéndolas a prueba experimentalmente como también analizando su comportamiento teóricamente , luego de revisar cada una tendremos una visión más amplia de que es lo que hace cada estructura y podremos anticipar en el momento de utilizar cual será la más adecuada a nuestras necesidades.

MiniSet

Es un tipo abstracto de datos representado por un conjunto de enteros, además contiene las operaciones básicas que heredan las estructuras que se analizarán, las cuales son insert(int), remove(int) y search(int).

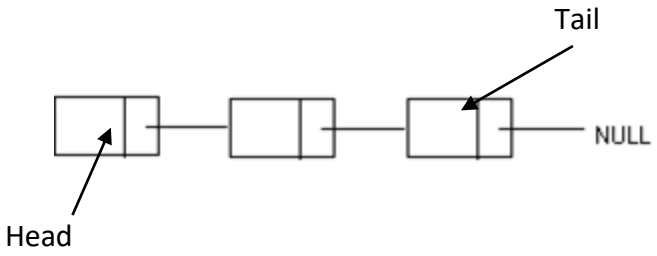
Insert: inserta un entero en el conjunto de datos, y sí ya está insertado no inserta el elemento en la estructura.

Remove: Elimina el elemento sí está insertado previamente en el conjunto

Search: Borra el elemento si el númeroaa existe en el conjunto

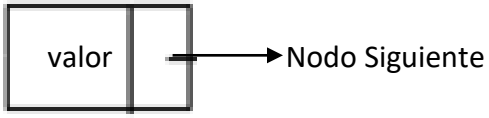
LinkedList

LinkedList es una de las estructuras de datos que será implementada heredando lo métodos de MiniSet , consiste en una secuencia de nodos, en la que se almacenan datos de forma dinámica , cada nodo contiene un valor además de un puntero que contiene la dirección del siguiente nodo , al tener un puntero esta estructura pasa a ser llamada lista simplemente enlazada, además hay dos punteros a los extremos de la lista uno que tiene la referencia al primer nodo insertado, el cual es llamado Head (indica el inicio de la lista) y el otro referencia al último nodo del lista llamado Tail.



Implementación

para implementar la lista se ocupa como pilar principal una estructura llamada node que contiene una variable (int) para guardar el valor del nodo y un apuntador que referencia al nodo siguiente



Los Métodos heredados de MiniSet se implementarán de la siguiente manera.

Search: Busca el número que es ingresado como argumento de entrada, la búsqueda es lineal esto significa que busca nodo por nodo y si lo encuentra entre la lista retorna verdadero de lo contrario retorna falso.

Insert: Inserta el número que es ingresado como argumento de entrada , La inserción de números en esta lista será de forma desordenada, se decide hacer de esta manera porque se ocupa la función search para saber si el numero ya está insertado, lo cual es equivalente a ir comparando nodo a nodo para saber la posición de inserción e insertar en orden .

Por consiguiente se ejecuta search si la búsqueda retorna falso (el número no se encuentra en la lista) , precede a insertar el numero al final de la lista

Remove: Elimina el entero que es ingresado como argumento de entrada , este es ayudado por search que busca el numero en la lista , si el numero es encontrado por search , precede a eliminar el nodo que contiene el valor.

Detalles de implementación

Hay 3 archivos fundamentales para la estructura ,está MiniSet.h que es el ADT(tipo abstracto de dato) de donde hereda los métodos insert() , remove() y search , también tenemos la cabecera del la estructura LinkedList.h en donde se declara la estructura node y La clase LinkedList con los contructores y los métodos heredados , además se declaran los punteros globales de tipo node head (indica el inicio de la lista), tail(indica el final de la lista) , select(recorre la lista) , aux(acompaña a select para saber el nodo anterior a el) ,todos ellos fueron declarados globales para que luego de ejecutar search se pueda acceder a la posición donde encontró el nodo con el valor buscado.

En el archivo Linkedlist.cpp se hace efectiva la implementación de la estructura con los siguientes métodos.

LinkedList(): Es el constructor de la lista ,en él se inicializa el puntero Head con NULL, lo que indica que la lista se crea vacía.

~LinkedList(): es el destructor de la lista y lo que hace es comenzar a borrar todos los nodos desde la cabeza hasta el final de la lista.

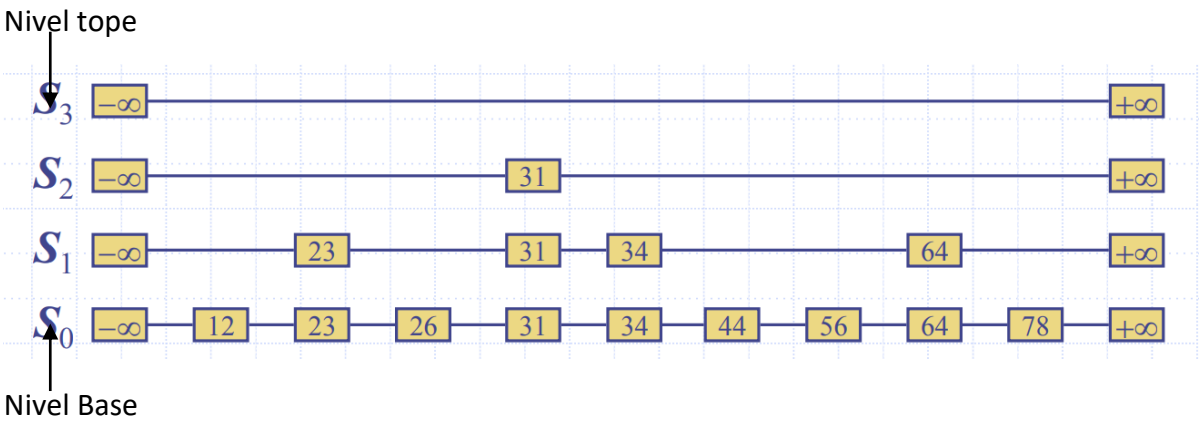
Search(int num):Contiene un ciclo que permite recorrer la lista desde Head hasta Tail , el puntero select comienza apuntando el inicio de la lista , luego dentro del ciclo tenemos dos condiciones, select puede ser igual a NULL lo que significa que la lista está vacia o llegó al final de la lista por consiguiente retorna falso. si el valor del nodo al cual apunta select es igual al valor buscado retorna verdadero. Si no se cumple ninguna condición se pasa al nodo siguiente en la lista y se guarda el nodo anterior en la variable aux.

Insert(int num) : Si la lista está vacia se inserta el nodo con el valor entregado por num , de lo contrario ejecuta a search(int num) si retorna falso inserta el valor num al final , la cola(tail) pasa a ser el nodo insertado.

Remove(int num) : Se hace una búsqueda del valor num con search y si es verdadero , procede a borrar el nodo apuntado por select , une el nodo apuntado por aux con el siguiente de select para conservar la conexión de la lista.

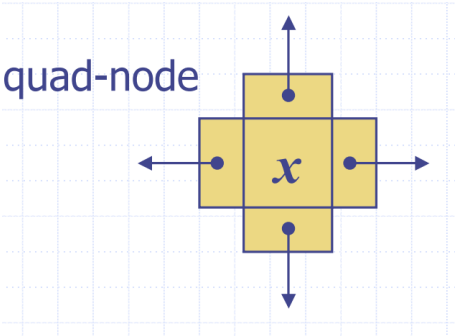
SkipList

Es una estructura basada en listas enlazadas paralelas, contiene una lista base en donde se guardan todos los números enteros insertados en la estructura, luego sobre la base están los niveles que contienen subconjuntos de números ella, generando una vía rápida para llegar a los valores ingresados en la skiplist,la cantidad de niveles es la altura de la estructura. sobre los niveles hay una lista vacía que indica el fin de ellos a, todas las listas ingresadas en la estructura contienen dos extremos infinitos para saber el inicio y el final de ellas.



Implementación

La estructura elemental utilizada para los niveles que están compuestos por listas enlazadas es un nodo con cuatro punteros(quad-node) que indican los elementos que están después, antes, sobre y bajo el nodo. Lo que facilita el recorrido por la skiplist.



Se hereda los métodos de Miniset, los cuales son insert , remove y search ,serán implementados de la siguiente manera.

Search: Busca el elemento comenzando por el tope de la skiplist, revisando cada nivel nodo a nodo, si en el nivel de búsqueda hay un número menor o igual al buscado, continua la búsqueda bajo ese nodo en el nivel inferior, hasta llegar a la base, si en la base no está el numero retorna falso, de lo contrario retorna verdadero.

Insert: Inserta un elemento a la skiplist desde la base hasta los niveles que determina aleatoriamente, los niveles que inserta son decididos "lanzando una moneda" esto depende de cuantas caras o sellos salgan repetidas de forma sucesiva, continua la inserción buscando si el número a insertar ya está en la base, de ser negativa la búsqueda comienza la inserción desde la base hasta los niveles decretados en el lanzamiento de la moneda.

Remove: Ejecuta la búsqueda del número a eliminar, sí está en la estructura, suprime desde la base hasta el último nivel los nodos que contienen el elemento.

Detalles de implementación

Hay 3 archivos fundamentales para la estructura, está MiniSet.h que es el ADT (tipo abstracto de dato) de donde hereda los métodos insert() , remove() y search , también tenemos la cabecera de la estructura SkipList.h en donde se declara la estructura nodo de 4 punteros y La clase LinkedList con los contructores y los métodos heredados , hay un entero llamado level que contiene la cantidad de niveles de la skiplist ,además se declaran los punteros globales de tipo nodo:

Lmin:indica el infinitito negativo en el tope .

Lmax: apunta al límite superior del tope .

Select: recorre la skiplist .

Aux: acompaña a select a recorrer los niveles horizontalmente para saber el nodo siguiente a él

Downprev: puntero que es utilizado para recorrer los niveles de la skiplist , acompaña al puntero selección apuntando al nodo que está en el nivel inferior.

Tambien hay dos métodos internos uno agrega niveles (addLevel) y el otro nodo en una posición (addnodo)

En el archivo Skiplist.cpp se hace efectiva la implementación de la estructura con los siguientes métodos.

SkipList():Es el constructor de la skiplist crea la lista base con los extremos apuntando abajo a NULL, esa acción es útil para saber el fin skiplist verticalmente , luego enlaza el tope con la base , cada lista tiene sus extremos inicializados en infinito positivo y negativo, Los punteros Lmax y Lmin apuntan a los extremos del tope , también inicializa la cantidad de niveles en 0, al finalizar el constructor a creado una skipList vacía .

Search(int num):Este método se encarga de buscar el entero num en la base, la busqueda comienza en el límite inferior del tope con selección apuntando a Lmin , en un ciclo while la busqueda tiene su punto de partida. Primero examina si en el nodo que está ubicado selección es un valor menor al buscado y al mismo tiempo el número que está siguiente a él sea mayor al buscado si no se cumplé , selección pasa apuntar el siguiente nodo en el nivel y aux pasa a ser el siguiente de selección , se pregunta nuevamente lo anterior si lo anterior es verdad , verifica que selección esté en la base(selección está en base si el puntero que tiene el valor del nodo inferior es NULL) , si no está en la base baja un nivel , nuevamente se verifica que donde está selección sea menor al número buscado y aux (que es siempre el nodo siguiente de selección) sea mayor al número buscado.

-Si selección <num y aux>num y además selección está en la base el número no se encuentra , retorna false

- Si selección = num y selección está en la base, el número pertenece a la skiplist , retorna true.

Search deja siempre al puntero selección en la base ya sea que haya encontrado el numero o no.

Addnodo(int num): es un método privado de la skiplist y permite la inserción de nodos con valor num en la posicion entre selección y aux.

Addleve(int cant): es un método privado de la skiplist agrega niveles después del tope , dentro hay un ciclo for que agrega la cantidad de niveles ingresados como argumento de entrada(cant) , empareja los extremos del tope con los niveles agregados y el tope pasa a ser el último nivel agregado.

Insert(int num): en este método se ingresan los números a la skiplist , primero se calculan los niveles a “ingresar lanzando una moneda “ esto se hace con números y se calcula el módulo 2 , la cantidad de números pares consecutivos son los niveles a insertar. Se busca si el número a insertar ya está en la lista si es así se termina la inserción.

Si el número no está previamente en la skiplist,se analiza la diferencia de niveles entre los que ya están insertados en la lista con los niveles a insertar, si la diferencia es positiva, se insertan los niveles faltantes con addlevel(cant)(donde cant es la diferencia), como search()

deja al puntero selección en la lista base desde ahí se comienza la inserción hacia los niveles superiores .

Para insertar en los niveles superiores selección busca números que sean menor que el número a insertar y comprueba que este tenga un nodo asociado en el nivel superior e inserta el nodo entre el numero < num y el siguiente, quedando el nodo con valor num en orden en el nivel correspondiente.

Remove(int num): La primera acción que ejecuta es una búsqueda y si está en la base comienza a eliminar el nodo, luego busca en los niveles superiores la existencia del valor también los elimina y enlaza al anterior del nodo con el siguiente , para conservar la integridad de la skiplist

~SkipList(): es el destructor de la SkipList , libera toda la memoria utilizada por la estructura y va eliminando por nivel todos lo nodos que contiene desde el infinito negativo hasta el positivo.

Análisis teórico

Análisis de tiempo

LinkedList

Search: como search busca en cada nodo de la lista para encontrar el valor pedido su complejidad en peor caso es O(n) ya que si no está el nodo tendrá que recorrer toda la lista.

Insert: como usa a search para determinar si está el valor en la lista su complejidad es o(n).

Remove : como usa a search para determinar si está el valor en la lista su complejidad es o(n).

SkipList

Search: search se encarga de buscar nivel a nivel el numero solicitado el peor caso es que los niveles tengan la misma altura que la cantidad de datos ingresados por tanto search se puede demorar 0(n) como por caso

Insert: como usa a search para determinar si está el valor en la Skiplist su complejidad es o(n).

Remove : como usa a search para determinar si está el valor en la Skiplist su complejidad es o(n).

Análisis de espacio

Lista enlazada

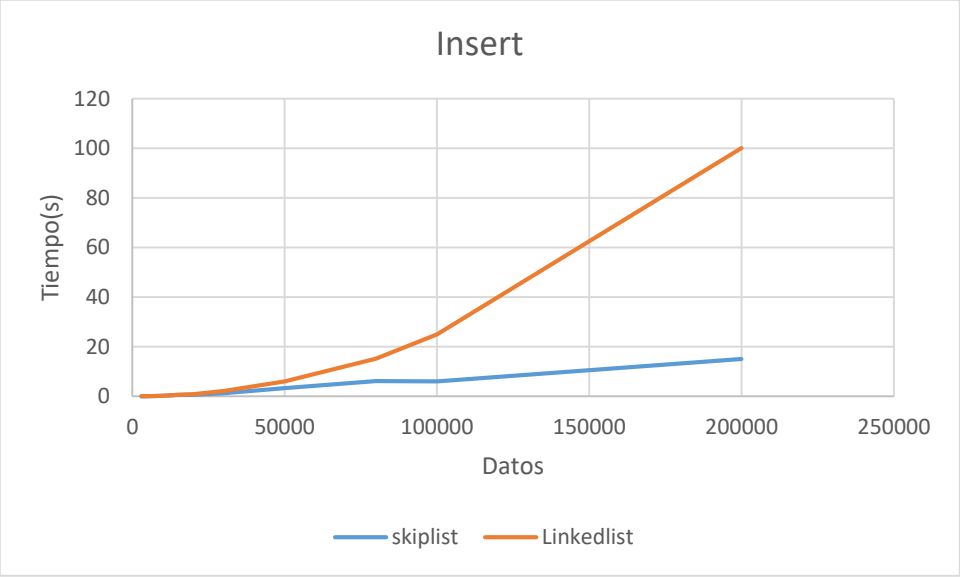
La lista enlazada como peor caso el espacio que ocupa es o(n) porque la cantidad de nodos creados es el tamaño de la lista.

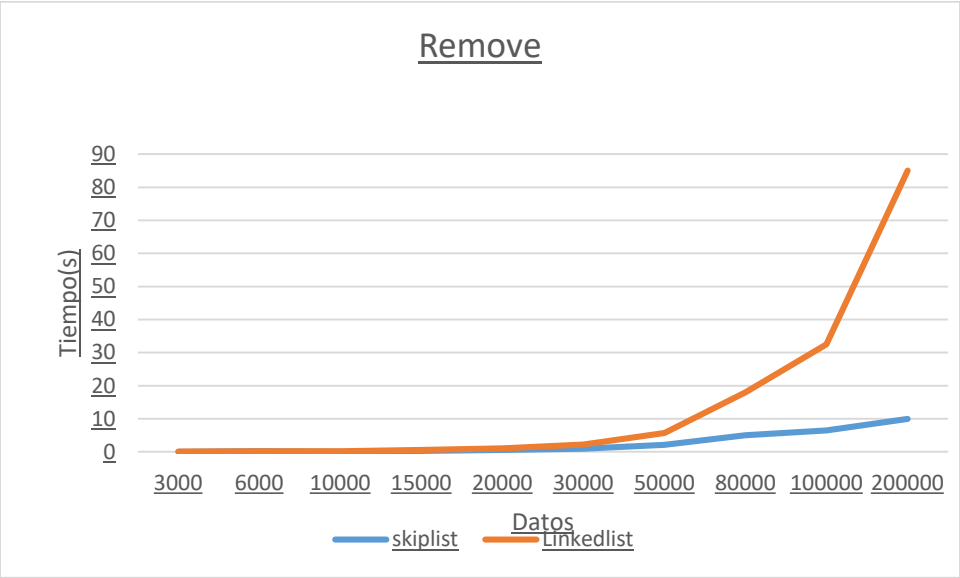
SkipList

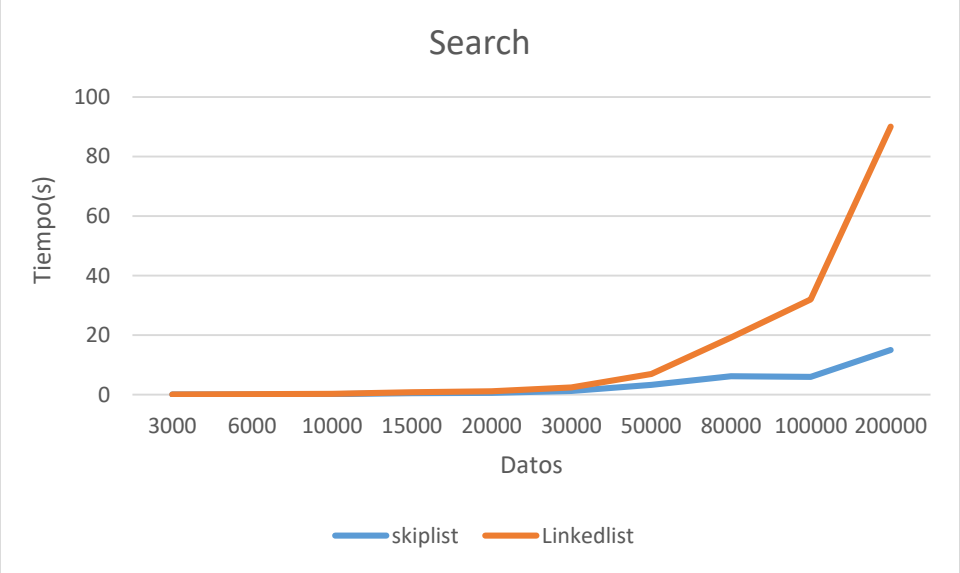
Como peor caso tenemos que los niveles estén completamente llenos, esto significa que tendríamos una cantidad n de elementos multiplicado por la cantidad de niveles O(niveles*n).

Analisis experimental

El análisis experimental consta en la comparación de los tiempos que tardan las estructuras en ejecutar los métodos que están declarados en




Remove
90
80
70
60
50
40
30
20
10
0
Tiempo(s)
3000
6000
10000
15000
20000
30000
50000
80000
100000
200000
Datos
skiplist
Linkedlist


Search
100
80
60
40
20
0
Tiempo(s)
3000
6000
10000
15000
20000
30000
50000
80000
100000
200000
Datos
skiplist
Linkedlist

Conclusión

Comparado las estructuras skiplist y LinkedList pudimos notar que la implementación de la LinkedList es más simple en el sentido que se necesitan menos líneas de código para que funcione la estructura , además fue mostrado el pilar básico de cada uno y a simple vista pudimos notar que la skiplist sería más compleja ya que se necesitan mas conexiones entre nodos , viendo los análisis teóricos de cada una se puede apreciar que cada una de las operaciones básicas tienen como por caso O(n) si solo nos quedamos con esa información podrimos inferir que ambas se demoran los mismo en hacer las operaciones insert,remove y search, pero cuando se mira el análisis experimental lo que se esperaba por la teoría se hiso efectivo solo cuando eran pocos datos ya que cuando aumentaron las cantidades skiplist fue más eficiente demorando menos tiempo que LinkedList en la ejecución de cada operacion , también podemos notar que teóricamente skip list va a ocupar más memoria que LinkedList.

Entonces si queremos almacenar pocos datos no importa cuál de las estructuras ocupemos ya que ambas tendrán un tiempo de ejecución similar sin embargo si queremos almacenar más de 80000 datos lo recomendado sería ocupar skipList ya que la diferencia de tiempos es notoria.

Bibliografía

https://people.ok.ubc.ca/ylucet/DS/SkipList.html