pour insérer vous méme sans utiliser randomly

createSkipListNode()

1//

value:entier

:SkipListNode

level:entier

Role:realiser la structure initiale de Skipkist

2//

createSkipList()

:SkipList

Role: creation d’une skiplist complete.

3//

randomLevel()

:entier

Role:creer les niveaux aleatoirement (flip coin).

4//

Search()

Value:entier

:SkipListNode

Skiplist:SkipList

Role:recherche d’un element dans la liste par sa valeur

5//

findPrevNode()

node:SkipListNode :SkipListNode

Role:recherche de le maillon précédent

6//

Insert()

skipList:SkipList

value:entier

Role:insérer une valeur dans skiplist.

7//

delete()

skipList:SkipList

Value:entier

Role:suprimer une maillon par une valeur .

8//

printSkipList()

skipList:SkipList

Role:l'affichage d'elements de skiplist( tous les niveaux) .

9//

searchByPosition()

Position:entier

:SkipListNode

skipList:SkipList

Role:recherche un element pour une position donnée { l'elem existe ou non} .

pour manipuler plus que 45000 maillon

10//

createNode2()

value:entier

:Node

level:entier

Role:creation de maillon pour la 2ème structure {pour plus que 45000 maillon}

11//

createSkipList2()

: SkipList2

Role:creation de skiplist pour la 2ème structure {pour plus que 45000 maillon}

12//

Insert2()

value:entier

skiplist2:SkipList2

Role:insérer des valeures dans skiplist

13//

Search2()

Value:entier

:Node

Skiplist2:SkipList2

Role: recherche par une valeur dans skiplist

14//

insertAfter()

after:Node

Skiplist2:SkipList

value:entier

Role:une fonction aide a insérer une valeur dans skiplist

15//

Delete2()

Value:entier

skiplist:SkipList

Role:supprimer un element par valeur entrer par l’utilisateur

16//

printSkipList2()

skiplist:SkipList

Role: afficher les elements de skiplist (tous les niveaux)

les fonctions d'affichages

17//

welcome()

Role:draw a box " welcome to our tp "

18//

intro()

Role: introduction to the topic of the tp

19//

Welcome()

Role:fonction d'affichage (de tp) .

20//

welcome2()

Role:fonction d'affichage.

21//

Menu\_rep1()

Role:presentation pour la première partie.

22//

Menu\_rep2()

Role: presentation de la 2ème partie

A small definition of skiplist and its functions:

A Skip List is a data structure that allows for efficient search, insertion, and deletion operations. It consists of a linked list of nodes, where each node contains a key and one or more pointers to other nodes. The pointers in each node are used to create "shortcuts" in the list, allowing for faster traversal of the list.

The Skip List is designed to have multiple layers, with the bottom layer being a standard linked list of all the elements in the list. Each subsequent layer is formed by "skipping" nodes from the lower layer, with the probability of a node being included in the next layer determined by a random process. This creates a structure that resembles a pyramid, with the top layer containing only a single node that acts as a sentinel.

When searching for an element in the Skip List, the search begins at the top layer and moves down to the lower layers. At each layer, the search moves forward in the list until it reaches a node whose key is greater than or equal to the search key, at which point the search moves down to the next layer. This process continues until the element is found or until the bottom layer is reached and the element is not found.

Insertion and deletion operations in the Skip List are similar to search operations, with the search being used to find the appropriate position for the new element or the element to be deleted. Once the position is found, the new element is inserted or the element to be deleted is removed, and the pointers in the affected nodes are updated to maintain the structure of the Skip List.

A o(t) of a skiplist

The time complexity of operations in a Skip List depends on the height of the Skip List and the number of elements in it.

Search: The time complexity of a search operation in a Skip List is O(log n), where n is the number of elements in the Skip List. This is because, at each level of the Skip List, the search operation only needs to consider half of the nodes.

Insertion: The time complexity of an insertion operation in a Skip List is also O(log n). This is because, like in the search operation, the insertion operation only needs to traverse half of the nodes at each level to find the appropriate insertion point.

Deletion: The time complexity of a deletion operation in a Skip List is also O(log n). This is because, similar to the insertion and search operations, the deletion operation only needs to traverse half of the nodes at each level to find the node to be deleted.

The space complexity of a Skip List is O(n), where n is the number of elements in the Skip List, since each node in the Skip List requires space for its key and pointers. However, the height of the Skip List can also affect the space complexity, with a higher height requiring more space.

THE STRUCTRES USED

We have uses 2 stuctures , Both structures provided have their own benefits and advantages,

and the choice of which one to use depends on the specific requirements and constraints of your application.

The first structure, which is a node-based implementation of a skip list, provides the flexibility to store additional

information in each node, such as pointers to adjacent nodes in all directions (up, down, left, right),

and a level number to represent the node's height within the skip list. This level number allows for efficient

searching and insertion of nodes in the skip list, as nodes at higher levels are skipped over during a search,

resulting in a faster search time.

On the other hand, the second structure, which is an array-based implementation of a skip list,

simplifies the node structure by only storing the value and level information for each node.

This makes the structure more memory-efficient and easier to implement, especially when dealing with large numbers of elements.

Therefore, if your application requires the ability to store additional information in each node,

such as pointers to adjacent nodes, or if you need to implement more advanced operations on the skip list,

such as deletion or dynamic resizing, the first structure would be a better choice. However,

if memory efficiency is a primary concern, or if you are dealing with a large number of elements,

the second structure would be a more suitable choice.

Overall, it's important to choose the right data structure based on your specific needs and requirements,

and both structures you provided have their own benefits that can be useful depending on the application.

How to calculate a skiplist size in order to draw the performance graph

The size of a skip list can be determined by considering the number of elements that will be stored in the list, as well as the probability of each element being included in a higher-level sublist.

One way to determine the size of a skip list is to use the formula:

size = n \* (log(1/p) / log(2))

where:

n is the number of elements to be stored in the skip list

p is the probability that any given element will be included in a higher-level sublist

The formula above assumes that the probability p is constant for all elements in the skip list. This formula provides a rough estimate of the size of the skip list needed to store n elements with a given probability of each element being included in a higher-level sublist. However, the actual size of a skip list may vary depending on the implementation and specific use case.

A small analysis of our way of thinking to solve this problem

Presentation1:

It gives you the freedom to made your own skiplist by providing you sush an empty skiplist initializing with 0 using the function createskiplistnode and the function createskiplist to put everything together it’s like we give you all the colors and the necessary mterials and then ask you to made art ,,,so everytime you want to insert a value we help you using the function insert and to get the levels we will use the function randomlevel and to made the delete operation more easy we involved two kinds of search which are the normal search and search by position so whatever we find the node you wanna to delete ,it’s gonna be deleted using the delete function ..

This structure deal with small number of nodes or let me say as much as you could handle (you couls insert as much as nodes you could)…..

Presentation1:

It will provide you a skiplist with random values using the function createnode2 and the function createskiplist2 to get the final shape of the skiplist which have random nodes with random values and levels…

For the deletion and he insertion we just need to use insertafter which is a function that insert the element you entered in the skiplist and you need to enter the element supposed to be before the inserted value ,,,it will automatically use the randomlevel function and use it on this new node .

For the delete which is the second choice ,you just enter the value you want to delete and we will search for ir using search2 function and it’s gonna be deleted from all the levels ..

After anychange on the generated skiplist it gonna be automatically printed in the execution screen according to printskiplist2 function ..

This presentation gonna be used to deal with big number of nodes(more than 50000)