School of Computer Science and Technology
Zhejiang University

Skip List

FIRST LAST
FIRST LAST
FIRST LAST

2020-05-20

Contents

1. Introduction

Skip list is a data structure that allows $\mathcal{O}(\log n)$ search complexity as well as $\mathcal{O}(\log n)$ insertion complexity within an ordered sequence of n elements. Thus it can get the best features of an array while maintaining a linked list-like structure that allows insertion, which is not possible in an array.

Fast search is made possible by maintaining a linked hierarchy of subsequences, with each successive subsequence skipping over fewer elements than the previous one.

Searching starts in the sparsest subsequence until two consecutive elements have been found, one smaller and one larger than or equal to the element searched for. Via the linked hierarchy, these two elements link to elements of the next sparsest subsequence, where searching is continued until finally we are searching in the full sequence.

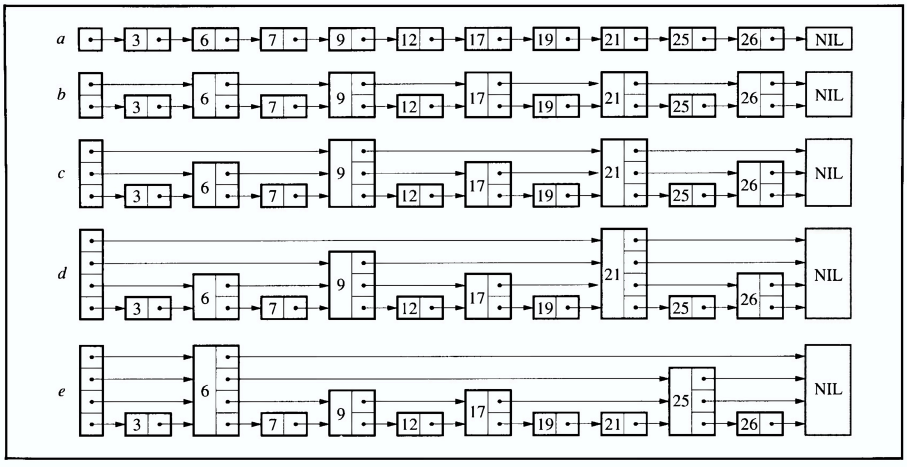

Figure 1.1: Linked Lists with Additional Pointers

2. Algorithm Specification

2.1 Data Structures

2.1.1 `class Node`

2.1.2 `class SkipList`

2.2 Main Operations

2.2.1 Search

We search for an element by traversing forward pointers that do not overshoot the node containing the element being searched for. When no more progress can be made at the current level of forward pointers, the search moves down to the next level. When we can make no more progress at level 1, we must be immediately in front of the node that contains the desired element(it it is in the list).

Algorithm 1 Search(list, searchKey)

Input: Slip list, search key
Output: required value

```
1: x:=list→header
2: –loop invariant: x→key
3: for i:=list→level downto 1 do
4:     while x→forward[i]→key < searchKey do
5:         i:=list→forward[i]
6:     end while
7: end for
8: x:=x→forward[1]
9: if x→key = searchKey then
10:    return x→value
11: end if
12: return failure
```

2.2.2 Insert

2.2.2.1 Generate Random Level

Initially, we discussed a probability distribution where half of the nodes that have level i pointers also have level $i+1$ pointers. To get away from magic constants, we say that a fraction p of the nodes with level i pointers also have level $i+1$ pointers. Levels are generated randomly by an algorithm, without reeference to the number of elements in the list.

Algorithm 2 RandomLevel()

Input:
Output:

```
1: newLevel:=1
2: –random()returns a random value in [0, 1)
3: while random() < p do
4:     newLevel:=newLevel + 1
5: end while
6: return min(newLevel, MaxLevel)
```

2.2.2.2 Insert Value

2.2.3 Delete

2.3 Main Function

Algorithm 3 Insert(list, searchKey, newValue)

Input:
Output:

```
1: local update[1...MaxLevel]
2: for  i:=list→level downto 1 do
3:     while x→forward[i]→key < searchKey do
4:         x:=x→forward[i]
5:         –x→key < searchKey ≤ x→forward[1]→key
6:     end while
7:     update[i]:=x
8: end for
9: x:=x→forward[1]
10: if  x→key = searchKey  then
11:     x→value:=newValue
12: else
13:     newLevel:=RandomLevel()
14:     if newLevel > list→level  then
15:         for i:=list→level+1 to newLevel do
16:             update[i]:=list→header
17:         end for
18:         list→level:=newLevel
19:     end if
20:     x:=makeNode(newLevel, searchKey, value)
21:     for i:=1 to newLevel do
22:         x→forward[i]:=update[i]→forward[i]
23:         update[i]→forward[i]:=x
24:     end for
25: end if
```

Algorithm 4 Delete(list, searchKey, newValue)

Input:
Output:

```
1: local update[1...MaxLevel]
2: x:=list→header
3: for  i:=list→level downto 1 do
4:     while x→forward[i]→key < searchKey do
5:         x:=x→forward[i]
6:     end while
7:     update[i]:=x
8: end for
9: x:=x→forward[1]
10: if  x→key = searchKey  then
11:     for i:=1 to list→level do
12:         if update[i]→forward[i] ≠ x then
13:             break
14:         end if
15:         update[i]→forward[i]:=x→forward[i]
16:     end for
17:     free(x)
18:     while list→level > 1 and list→header→forward[list→level] = NIL do
19:         list→level:=list→level-1
20:     end while
21: end if
```

3. Design of Experiment

[*Describe the process used to meet the project goal.*]

4. Computational model

[*Describe thoroughly the computational model/s used in the project*]

4.1 Problem geometry and setup

4.2 Mesh generation and description

4.3 Numerical schemes

5. Complexity Analysis

5.1 The Total Result

We give the results of the complexity analysis without proof, and then we will give our analysis. As we can see,

Table 5.1: The Result

Algorithm	**Average**	**Worst Case**
Space	$O(n)$	$O(n \log n)$
Search	$O(\log n)$	$O(n)$
Insert	$O(\log n)$	$O(n)$
Delete	$O(\log n)$	$O(n)$

5.2 Related Definitinos

In order to better analyze the complexity through mathematical means, we will introduce some related concepts in advance, which will help us simplify the analysis

5.2.1 C_m^n

5.3 Space Complexcity

Every time a number is inserted, the program will randomly assign a height for node to storage pointer (less than MaxHeight), so it's Space Complexity is $O(n)$

5.4 Average Time Complexity

5.4.0.1 Definitions

The height of the PSL is expected to be about $log_{\frac{1}{P}} N$. Since, among all elements that made it to a certain level, about every (1/P)th element will make it to the next higher level, one should expect to make 1/p key comparisons per level. Therefore, one should expect about 1/p*log1/p n key comparisons in total, when searching for +. As it will turn out (Theorem 3.3), this is exactly the leading term in the search cost for + in a PSL of n keys.

6. Conclusions

Bibliography

[1] Thomas D Economon, Francisco Palacios, Sean R Copeland, Trent W Lukaczyk, and Juan J Alonso. Su2: An open-source suite for multiphysics simulation and design. *Aiaa Journal*, 54(3):828–846, 2015.

Appendix A: Resources

[*Report the config files of the software used (i.e. SU2 [1] and the mesher). Also attach to this report an archive with the mesh files, solutions and the reference solution data (e.g. data points of a Cp plot ...)*]

Mesh configuration files

SU2 configuration files

Appendix A: Resources

[*Report the config files of the software used (i.e. SU2 [1] and the mesher). Also attach to this report an archive with the mesh files, solutions and the reference solution data (e.g. data points of a Cp plot ...)*]

Mesh configuration files

SU2 configuration files

/24.88/10

/10 arabic pageΔl@page Level

/24.88RandomLevel()
/24.88newLevel:=1
/24.88--random()returns a random value in [0, 1)
/24.88while /24.88random() < p /24.88do
/24.88newLevel:=newLevel + 1
/24.88return /24.88min(newLevel, MaxLevel)

/24.88/10/10Insert Value ubsubsectionmarkInsert Value

/24.88Insert(list, searchKey, newValue)

/24.88/10

/10 arabic
pageΔl@page
/24.88local /24.88update[1...MaxLevel]
/24.88x:=list$\rightarrow$header
/24.88for/24.88 i:=list$\rightarrow$level
/24.88downto /24.881 /24.88do
/24.88while /24.88 x$\rightarrow$forward[i]$\rightarrow$key
$<$ searchKey /24.88do
/24.88x:=x$\rightarrow$forward[i]
/24.88--x$\rightarrow$key $<$ searchKey $\leq$
x$\rightarrow$forward[1]$\rightarrow$key
/24.88update[i]:=x
/24.88x:=x$\rightarrow$forward[1]
/24.88if/24.88 x$\rightarrow$key = searchKey
/24.88then /24.88x$\rightarrow$value:=newValue
/24.88else
/24.88newLevel:=RandomLevel()
/24.88if /24.88newLevel $>$ list$\rightarrow$level
/24.88then

/24.88/10

/10 arabic
pageΔl@page
/24.88for /24.88i:=list→level+1
/24.88to /24.88newLevel /24.88do
/24.88update[i]:=list→header
/24.88list→level:=newLevel
/24.88x:=makeNode(newLevel,
searchKey, value)
/24.88for /24.88i:=1 /24.88to
/24.88newLevel /24.88do
/24.88x→forward[i]:=update[i]→forwarc
/24.88update[i]→forward[i]:=x

subsectionΔl@subsection/24.88/12/12hesu
ubsectionmarkDelete

/24.88Delete(list, searchKey,
newValue)
/24.88local /24.88update[1...MaxLevel]

/24.88/10

/10 arabic
pageΔl@page
/24.88x:=list→header
/24.88for/24.88 i:=list→level
/24.88downto /24.881 /24.88do
/24.88while /24.88 x→forward[i]→key
< searchKey /24.88do
/24.88x:=x→forward[i]
/24.88update[i]:=x
/24.88x:=x→forward[1]
/24.88if /24.88x→key = searchKey
/24.88then
/24.88for /24.88i:=1 /24.88to
/24.88list→level /24.88do
/24.88if /24.88update[i]→forward[i]
≠ x /24.88then break
/24.88update[i]→forward[i]:=x→forwarc
/24.88free(x)
/24.88while /24.88list→level

/24.88/10

/10 arabic

pageΔl@page

> 1 /24.88and

/24.88list→header→forward[list→level

= NIL /24.88do

/24.88list→level:=list→level-1

sectionΔl@section/24.88/14.4/14.4hesect
Function

ectionmarkMain Function

```
s@plain/24.88/10

   /10                        arabic
   pageΔl@page
   chapterΔl@chapter/24.88/20.74/20.74Chap


   0

   /24.88/24.88Design of
   Experiment

   [/24.88Describe the process
   used to meet the project goal.]
```

s@plain/24.88/10

/10 arabic
pageΔl@page
chapterΔl@chapter/24.88/20.74/20.74Cha

0

/24.88/24.88Computational
model

[/24.88Describe thoroughly the
computational model/s used in
the project]

sectionΔl@section/24.88/14.4/14.4hesec
geometry and setup
ectionmarkProblem geometry and
setup

/24.88/10

/10 arabic

pageΔl@page

sectionΔl@section/24.88/14.4/14.4hesec
generation and description

ectionmarkMesh generation and

description

sectionΔl@section/24.88/14.4/14.4hesec
schemes

ectionmarkNumerical schemes

s@plain/24.88/10

/10 arabic
pageΔl@page
chapterΔl@chapter/24.88/20.74/20.74Cha

0

/24.88/24.88Complexity
Analysis

sectionΔl@section/24.88/14.4/14.4hesec
Total Result
ectionmarkThe Total Result

We give the results of the complexity
analysis without proof, and
then we will give our analysis.

As we can see, able[h]
abularccc /24.88Algorithm

/24.88/10

/10 arabic

pageΔl@page

/24.88Average /24.88Worst Case

Space	$O(n)$	$O(n \log n)$
Search	$O(\log n)$	$O(n)$
Insert	$O(\log n)$	$O(n)$
Delete	$O(\log n)$	$O(n)$

ndtabular The Result ndtable

sectionΔl@section/24.88/14.4/14.4hesec
Definitinos

ectionmarkRelated Definitinos

In order to better analyze the complexity through mathematical means, we will introduce some related concepts in advance, which will help us simplify the analysis

/24.88/10

/10 arabic
pageΔl@page

subsectionΔl@subsection/24.88/12/12hesu
ubsectionmark@$24.88v$@$normalC_m^n$

sectionΔl@section/24.88/14.4/14.4hesec
Complexcity
ectionmarkSpace Complexcity

Every time a number is inserted,
the program will randomly assign
a height for node to storage
pointer (less than MaxHeight),
so its Space Complexity is $O(n)$

sectionΔl@section/24.88/14.4/14.4hesec
Time Complexity

/24.88/10

/10 arabic
pageΔl@page
ectionmarkAverage Time Complexity

/24.88/10/10Definitions
ubsubsectionmarkDefinitions

The height of the PSL is expected to be about $log_{\frac{1}{P}} N$. Since, among all elements that made it to a certain level, about every (1/P)th element will make it to the next higher level, one should expect to make 1/p key comparisons per level. Therefore, one should expect about 1/p*log1/p n key comparisons in total, when searching for +. As it will turn out (Theorem 3.3),

/24.88/10

/10 arabic pageΔl@page
this is exactly the leading term in the search cost for + in a PSL of n keys.

```
s@plain/24.88/10

   /10                          arabic
   pageΔl@page
   chapterΔl@chapter/24.88/20.74/20.74Cha


   0

   /24.88/24.88Conclusions
```

```
/24.88/10
   /10                          arabic
   page∆l@page
```

```
/24.88/10
   /10                          arabic
   page∆l@page
```