Analysis of Algorithms

Skip Lists

Andres Mendez-Vazquez

October 23, 2020

Outline

Outline

Dictionaries

Definition

A dictionary is a collection of elements; each of which has a unique search key.

- Uniqueness criteria may be relaxed (multi-set).
- Do not force uniqueness.

Dictionaries

Definition

A dictionary is a collection of elements; each of which has a unique search key.

- Uniqueness criteria may be relaxed (multi-set).
- Do not force uniqueness.

Purpose

Dictionaries keep track of current members, with periodic insertions and deletions into the set (similar to a database).

- Course records
- Symbol table (with duplicates)
- Language dictionary (Webster, RAE, Oxford)

Dictionaries

Definition

A dictionary is a collection of elements; each of which has a unique search key.

- Uniqueness criteria may be relaxed (multi-set).
- Do not force uniqueness.

Purpose

Dictionaries keep track of current members, with periodic insertions and deletions into the set (similar to a database).

Examples

- Membership in a club.
- Course records.
- Symbol table (with duplicates).
- Language dictionary (Webster, RAE, Oxford).

Dictionaries

Definition

A dictionary is a collection of elements; each of which has a unique search key.

- Uniqueness criteria may be relaxed (multi-set).
- Do not force uniqueness.

Purpose

Dictionaries keep track of current members, with periodic insertions and deletions into the set (similar to a database).

Examples

- Membership in a club.
- Course records.
- Symbol table (with duplicates).
- Language dictionary (Webster, RAE, Oxford).

Dictionaries

Definition

A dictionary is a collection of elements; each of which has a unique search key.

- Uniqueness criteria may be relaxed (multi-set).
- Do not force uniqueness.

Purpose

Dictionaries keep track of current members, with periodic insertions and deletions into the set (similar to a database).

Examples

- Membership in a club.
- Course records.
- Symbol table (with duplicates).
- Language dictionary (Webster, RAE, Oxford).

Dictionaries

Definition

A dictionary is a collection of elements; each of which has a unique search key.

- Uniqueness criteria may be relaxed (multi-set).
- Do not force uniqueness.

Purpose

Dictionaries keep track of current members, with periodic insertions and deletions into the set (similar to a database).

Examples

- Membership in a club.
- Course records.
- Symbol table (with duplicates).
- Language dictionary (Webster, RAE, Oxford).

Dictionaries

Definition

A dictionary is a collection of elements; each of which has a unique search key.

- Uniqueness criteria may be relaxed (multi-set).
- Do not force uniqueness.

Purpose

Dictionaries keep track of current members, with periodic insertions and deletions into the set (similar to a database).

Examples

- Membership in a club.
- Course records.
- Symbol table (with duplicates).
- Language dictionary (Webster, RAE, Oxford).

Dictionaries

Definition

A dictionary is a collection of elements; each of which has a unique search key.

- Uniqueness criteria may be relaxed (multi-set).
- Do not force uniqueness.

Purpose

Dictionaries keep track of current members, with periodic insertions and deletions into the set (similar to a database).

Examples

- Membership in a club.
- Course records.
- Symbol table (with duplicates).
- Language dictionary (Webster, RAE, Oxford).

Example: Course records

Dictionary with member records

key ID	Student Name	HW1	
123	Stan Smith	49	$\cdots$
125	Sue Margolin	45	$\cdots$
128	Billie King	24	$\cdots$
	$\vdots$		
	$\vdots$		
190	Roy Miller	36	$\cdots$

Outline

The dictionary ADT operations

Some operations on dictionaries

- size(): Returns the size of the dictionary.
- empty(): Returns TRUE if the dictionary is empty.
- findItem(key): Locates the item with the specified key.
- findAllItems(key): Locates all items with the specified key.
- removeItem(key): Removes the item with the specified key.
- removeAllItems(key): Removes all items with the specified key.
- insertItem(key,element): Inserts a new key-element pair.

The dictionary ADT operations

Some operations on dictionaries

- size(): Returns the size of the dictionary.
- empty(): Returns TRUE if the dictionary is empty.
- findItem(key): Locates the item with the specified key.
- findAllItems(key): Locates all items with the specified key.
- removeItem(key): Removes the item with the specified key.
- removeAllItems(key): Removes all items with the specified key.
- insertItem(key,element): Inserts a new key-element pair.

The dictionary ADT operations

Some operations on dictionaries

- size(): Returns the size of the dictionary.
- empty(): Returns TRUE if the dictionary is empty.
- findItem(key): Locates the item with the specified key.
- findAllItems(key): Locates all items with the specified key.
- removeItem(key): Removes the item with the specified key.
- removeAllItems(key): Removes all items with the specified key.
- insertItem(key,element): Inserts a new key-element pair.

The dictionary ADT operations

Some operations on dictionaries

- size(): Returns the size of the dictionary.
- empty(): Returns TRUE if the dictionary is empty.
- findItem(key): Locates the item with the specified key.
- findAllItems(key): Locates all items with the specified key.
- removeItem(key): Removes the item with the specified key.
- removeAllItems(key): Removes all items with the specified key.
- insertItem(key,element): Inserts a new key-element pair.

The dictionary ADT operations

Some operations on dictionaries

- size(): Returns the size of the dictionary.
- empty(): Returns TRUE if the dictionary is empty.
- findItem(key): Locates the item with the specified key.
- findAllItems(key): Locates all items with the specified key.
- removeItem(key): Removes the item with the specified key.
- removeAllItems(key): Removes all items with the specified key.
- insertItem(key,element): Inserts a new key-element pair.

The dictionary ADT operations

Some operations on dictionaries

- size(): Returns the size of the dictionary.
- empty(): Returns TRUE if the dictionary is empty.
- findItem(key): Locates the item with the specified key.
- findAllItems(key): Locates all items with the specified key.
- removeItem(key): Removes the item with the specified key.
- removeAllItems(key): Removes all items with the specified key.
- insertItem(key,element): Inserts a new key-element pair.

The dictionary ADT operations

Some operations on dictionaries

- size(): Returns the size of the dictionary.
- empty(): Returns TRUE if the dictionary is empty.
- findItem(key): Locates the item with the specified key.
- findAllItems(key): Locates all items with the specified key.
- removeItem(key): Removes the item with the specified key.
- removeAllItems(key): Removes all items with the specified key.
- insertItem(key,element): Inserts a new key-element pair.

Example of unordered dictionary

Example

Consider an empty unordered dictionary, we have then...

Operation	Dictionary	Output
InsertItem$(5, A)$	$\{(5, A)\}$	
InsertItem$(7, B)$	$\{(5, A), (7, B)\}$	
findItem(7)	$\{(5, A), (7, B)\}$	B
findItem(4)	$\{(5, A), (7, B)\}$	No Such Key
size$()$	$\{(5, A), (7, B)\}$	2
removeItem(5)	$\{(7, B)\}$	A
findItem(4)	$\{(7, B)\}$	No Such Key

Outline

How to implement a dictionary?

There are many ways of implementing a dictionary

- Sequences / Arrays
  - Ordered
  - Unordered
- Binary search trees
- Skip lists
- Hash tables

How to implement a dictionary?

There are many ways of implementing a dictionary

- Sequences / Arrays
  - Ordered
  - Unordered
- Binary search trees
- Skip lists
- Hash tables

How to implement a dictionary?

There are many ways of implementing a dictionary

- Sequences / Arrays
  - Ordered
  - Unordered
- Binary search trees
- Skip lists
- Hash tables

How to implement a dictionary?

There are many ways of implementing a dictionary

- Sequences / Arrays
  - Ordered
  - Unordered
- Binary search trees
- Skip lists
- Hash tables

Recall Arrays...

Unordered array

34	14	12	22	18

Recall Arrays...

Unordered array

34	14	12	22	18

Complexity

- Searching and removing takes $O(n)$.
- Inserting takes $O(1)$.

Recall Arrays...

Unordered array

34	14	12	22	18

Complexity

- Searching and removing takes $O(n)$.
- Inserting takes $O(1)$.

Applications

This approach is good for log files where insertions are frequent but searches and removals are rare.

Recall Arrays...

Unordered array

34	14	12	22	18

Complexity

- Searching and removing takes $O(n)$.
- Inserting takes $O(1)$.

Applications

This approach is good for log files where insertions are frequent but searches and removals are rare.

More Arrays

Ordered array

12	14	18	22	34

More Arrays

Ordered array

12	14	18	22	34

Complexity

- Searching takes $O(\log n)$ time (binary search).
- Insert and removing takes O(n) time.

More Arrays

Ordered array

12	14	18	22	34

Complexity

- Searching takes $O(\log n)$ time (binary search).
- Insert and removing takes O(n) time.

Applications

This aproach is good for look-up tables where searches are frequent but insertions and removals are rare.

More Arrays

Ordered array

12	14	18	22	34

Complexity

- Searching takes $O(\log n)$ time (binary search).
- Insert and removing takes O(n) time.

Applications

This aproach is good for look-up tables where searches are frequent but insertions and removals are rare.

Binary searches

Features

- Narrow down the search range in stages
- "High-low" game.

Binary searches

Example find Element(22)

2	4	5	7	8	9	12	**14**	17	19	22	25	27	28	33
$\uparrow$ LOW							$\uparrow$ MID							$\uparrow$ $HIGH$

2	4	5	7	8	9	12	14	17	19	22	**25**	27	28	33
								$\uparrow$ LOW			$\uparrow$ MID			$\uparrow$ $HIGH$

2	4	5	7	8	9	12	14	17	**19**	22	25	27	28	33
								$\uparrow$ LOW	$\uparrow$ MID	$\uparrow$ $HIGH$				

2	4	5	7	8	9	12	14	17	19	**22**	25	27	28	33
										$\uparrow$ $LOW=MID=HIGH$				

Binary searches

Example find Element(22)

2	4	5	7	8	9	12	**14**	17	19	22	25	27	28	33

$\uparrow$ LOW (at 2) $\uparrow$ MID (at 14) $\uparrow$ $HIGH$ (at 33)

2	4	5	7	8	9	12	14	17	19	22	**25**	27	28	33

$\uparrow$ LOW (at 17) $\uparrow$ MID (at 25) $\uparrow$ $HIGH$ (at 33)

2	4	5	7	8	9	12	14	17	**19**	22	25	27	28	33

$\uparrow$ LOW $\uparrow$ MID $\uparrow$ $HIGH$

2	4	5	7	8	9	12	14	17	19	**22**	25	27	28	33

$\uparrow$ $LOW = MID = HIGH$

Binary searches

Example find Element(22)

2	4	5	7	8	9	12	**14**	17	19	22	25	27	28	33
$\uparrow$ LOW							$\uparrow$ MID							$\uparrow$ $HIGH$

2	4	5	7	8	9	12	14	17	19	22	**25**	27	28	33
								$\uparrow$ LOW			$\uparrow$ MID			$\uparrow$ $HIGH$

2	4	5	7	8	9	12	14	17	**19**	22	25	27	28	33
								$\uparrow$ LOW	$\uparrow$ MID	$\uparrow$ $HIGH$				

2	4	5	7	8	9	12	14	17	19	**22**	25	27	28	33
										$\uparrow$ $LOW=MID=HIGH$				

Binary searches

Example find Element(22)

2	4	5	7	8	9	12	**14**	17	19	22	25	27	28	33
↑ LOW							↑ MID							↑ $HIGH$

2	4	5	7	8	9	12	14	17	19	22	**25**	27	28	33
								↑ LOW			↑ MID			↑ $HIGH$

2	4	5	7	8	9	12	14	17	**19**	22	25	27	28	33
								↑ LOW	↑ MID	↑ $HIGH$				

2	4	5	7	8	9	12	14	17	19	**22**	25	27	28	33
										↑ $LOW{=}MID{=}HIGH$				

Recall binary search trees

Implement a dictionary with a BST

A binary search tree is a binary tree T such that:

- Each internal node stores an item (k, e) of a dictionary.
- Keys stored at nodes in the left subtree of v are less than or equal to k.
- Keys stored at nodes in the right subtree of v are greater than or equal to k.

Recall binary search trees

Implement a dictionary with a BST

A binary search tree is a binary tree T such that:

- Each internal node stores an item (k, e) of a dictionary.
- Keys stored at nodes in the left subtree of v are less than or equal to k.
- Keys stored at nodes in the right subtree of v are greater than or equal to k.

Recall binary search trees

Implement a dictionary with a BST

A binary search tree is a binary tree T such that:

- Each internal node stores an item (k, e) of a dictionary.
- Keys stored at nodes in the left subtree of v are less than or equal to k.
- Keys stored at nodes in the right subtree of v are greater than or equal to k.

Recall binary search trees

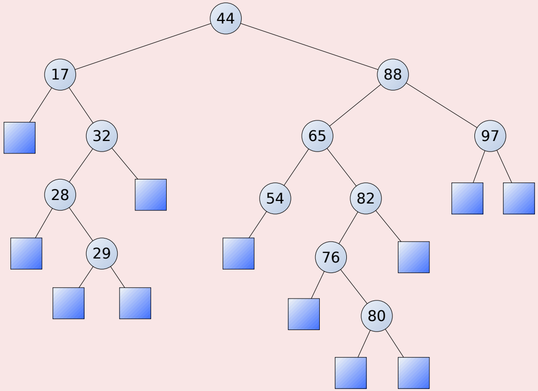
Implement a dictionary with a BST

A binary search tree is a binary tree T such that:

- Each internal node stores an item (k, e) of a dictionary.
- Keys stored at nodes in the left subtree of v are less than or equal to k.
- Keys stored at nodes in the right subtree of v are greater than or equal to k.

Binary searches Trees

Problem!!! Keeping a Well Balanced Binary Search Tree can be difficult!!!



Not only that...

Binary Search Trees

- They are not so well suited for parallel environments.
  - Unless a heavy modifications are done

Not only that...

Binary Search Trees

- They are not so well suited for parallel environments.
  - Unless a heavy modifications are done

In addition

We want to have a

- Compact Data Structure.
- Using as little memory as possible

Not only that...

Binary Search Trees

- They are not so well suited for parallel environments.
  - Unless a heavy modifications are done

In addition

We want to have a

- Compact Data Structure.
- Using as little memory as possible

Not only that...

Binary Search Trees

- They are not so well suited for parallel environments.
  - Unless a heavy modifications are done

In addition

We want to have a

- Compact Data Structure.
- Using as little memory as possible

Thus, we have the following possibilities

Unordered array complexities

Insertion: $O(1)$

Search: $O(n)$

Ordered array complexities

Insertion: $O(n)$

Search: $O(\log n)$

Well balanced binary trees complexities

Insertion: $O(\log n)$

Search: $O(\log n)$

Big Drawback - Complex parallel Implementation and waste of memory.

Thus, we have the following possibilities

Unordered array complexities

Insertion: $O(1)$

Search: $O(n)$

Ordered array complexities

Insertion: $O(n)$

Search: $O(\log n)$

Well balanced binary trees complexities

Insertion: $O(\log n)$

Search: $O(\log n)$

Big Drawback - Complex parallel Implementation and waste of memory.

Thus, we have the following possibilities

Unordered array complexities

Insertion: $O(1)$

Search: $O(n)$

Ordered array complexities

Insertion: $O(n)$

Search: $O(\log n)$

Well balanced binary trees complexities

Insertion: $O(\log n)$

Search: $O(\log n)$

Big Drawback - Complex parallel Implementation and waste of memory.

We want something better!!!

For this

We will present a probabilistic data structure known as Skip List!!!

Outline

Starting from Scratch

First

- Imagine that you only require to have searches.
- A first approximation to it is the use of a link list for it ($\Theta(n)$ search complexity).
- Then, using this How do we speed up searches?

Starting from Scratch

First

- Imagine that you only require to have searches.
- A first approximation to it is the use of a link list for it ($\Theta(n)$ search complexity).
- Then, using this How do we speed up searches?

Something Notable

- Use two link list, one a subsequence of the other.

- The Top is the high way system,

Starting from Scratch

First

- Imagine that you only require to have searches.
- A first approximation to it is the use of a link list for it ($\Theta(n)$ search complexity).
- Then, using this How do we speed up searches?

Something Notable

- Use two link list, one a subsequence of the other.

Imagine the two lists as a road system

1. The Bottom is the normal road system, L_2.
2. The Top is the high way system, L_1.

Starting from Scratch

First

- Imagine that you only require to have searches.
- A first approximation to it is the use of a link list for it ($\Theta(n)$ search complexity).
- Then, using this How do we speed up searches?

Something Notable

- Use two link list, one a subsequence of the other.

Imagine the two lists as a road system

1. The Bottom is the normal road system, L_2.
2. The Top is the high way system, L_1.

Starting from Scratch

First

- Imagine that you only require to have searches.
- A first approximation to it is the use of a link list for it ($\Theta(n)$ search complexity).
- Then, using this How do we speed up searches?

Something Notable

- Use two link list, one a subsequence of the other.

Imagine the two lists as a road system

1. The Bottom is the normal road system, L_2.
2. The Top is the high way system, L_1.

Starting from Scratch

First

- Imagine that you only require to have searches.
- A first approximation to it is the use of a link list for it ($\Theta(n)$ search complexity).
- Then, using this How do we speed up searches?

Something Notable

- Use two link list, one a subsequence of the other.

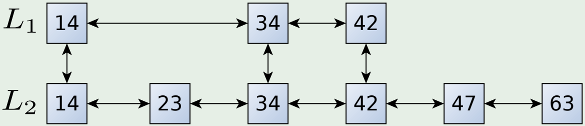
Imagine the two lists as a road system

1. The Bottom is the normal road system, L_2.
2. The Top is the high way system, L_1.

Outline

Example

High-Bottom Way System



Thus, we have...

The following rule

To Search first search in the top one (L_1) as far as possible, then go down and search in the bottom one (L_2).

We can use a little bit of optimization

We have the following worst cost

$$\text{Search Cost High-Bottom Way System} = \text{Cost Searching Top} + \ldots$$

$$\text{Cost Search Bottom}$$

Or

$$\text{Search Cost} = length\left(L_1\right) + \text{Cost Search Bottom}$$

The interesting part is "Cost Search Bottom"

This can be calculated by the following quotient:

$$\frac{length\left(L_2\right)}{length\left(L_1\right)}$$

We can use a little bit of optimization

We have the following worst cost

$$\text{Search Cost High-Bottom Way System} = \text{Cost Searching Top} + ...$$

$$\text{Cost Search Bottom}$$

Or

$$\text{Search Cost} = length\left(L_1\right) + \text{Cost Search Bottom}$$

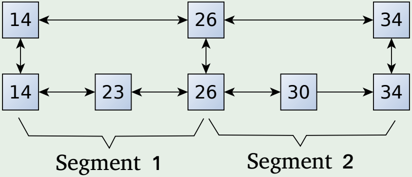
The interesting part is "Cost Search Bottom"

This can be calculated by the following quotient:

$$\frac{length\left(L_2\right)}{length\left(L_1\right)}$$

Why?

If we think we are jumping



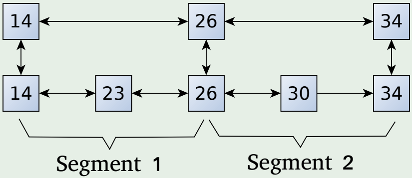
Then cost of searching each of the bottom segments = 2

Thus the ratio is a "decent" approximation to the worst case search

$$\frac{length\left(L_2\right)}{length\left(L_1\right)} = \frac{5}{3} = 1.66$$

Why?

If we think we are jumping



Then cost of searching each of the bottom segments $= 2$

Thus the ratio is a "decent" approximation to the worst case search

$$\frac{length\,(L_2)}{length\,(L_1)} = \frac{5}{3} = 1.66$$

Outline

Thus, we have...

Then, the cost for a search (when $length\left(L_2\right) = n$)

$$\text{Search Cost } = length\left(L_1\right) + \frac{length\left(L_2\right)}{length\left(L_1\right)} = length\left(L_1\right) + \frac{n}{length\left(L_1\right)} \tag{1}$$

Taking the derivative with respect to $length\left(L_1\right)$ and making the result equal 0

$$\frac{d\text{Search Cost}}{dlength\left(L_1\right)} = 1 - \frac{n}{length^2\left(L_1\right)} = 0$$

Thus, we have...

Then, the cost for a search (when $length\,(L_2) = n$)

$$\text{Search Cost } = length\,(L_1) + \frac{length\,(L_2)}{length\,(L_1)} = length\,(L_1) + \frac{n}{length\,(L_1)} \quad (1)$$

Taking the derivative with respect to $length\,(L_1)$ and making the result equal 0

$$\frac{d\text{Search Cost}}{dlength\,(L_1)} = 1 - \frac{n}{length^2\,(L_1)} = 0$$

Final Cost

We have that the optimal length for L_1

$$length\,(L_1) = \sqrt{n}$$

Plugging back in (Eq. 1)

$$\text{Search Cost } = \sqrt{n} + \frac{n}{\sqrt{n}} = \sqrt{n} + \sqrt{n} = 2 \times \sqrt{n}$$

Final Cost

We have that the optimal length for L_1

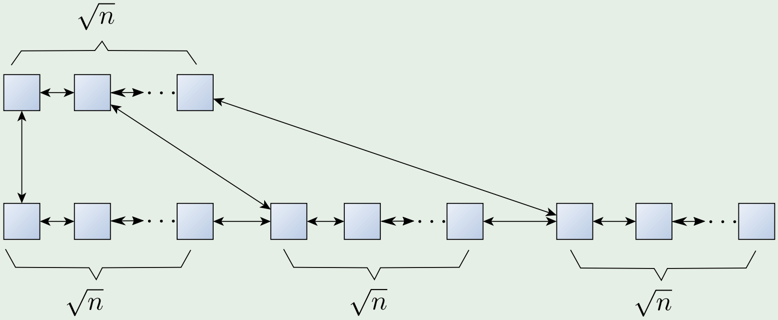
$$length\,(L_1) = \sqrt{n}$$

Plugging back in (Eq. 1)

$$\text{Search Cost } = \sqrt{n} + \frac{n}{\sqrt{n}} = \sqrt{n} + \sqrt{n} = 2 \times \sqrt{n}$$

Data structure with a Square Root Relation

Something like this



Now

For a three layer link list data structure

We get a search cost of $3 \times \sqrt[3]{n}$

In general for k layers, we have

$$k \times \sqrt[k]{n}$$

Thus, if we make $k = \log_2 n$, we get

$$\begin{aligned}\text{Search Cost} &= \log_2 n \times \sqrt[\log_2 n]{n} \\ &= \log_2 n \times (n)^{1/\log_2 n} \\ &= \log_2 n \times (n)^{\log_n 2} \\ &= \log_2 n \times 2 \\ &= \Theta(\log_2 n)\end{aligned}$$

Now

For a three layer link list data structure

We get a search cost of $3 \times \sqrt[3]{n}$

In general for k layers, we have

$$k \times \sqrt[k]{n}$$

Thus, if we make $k = \log_2 n$, we get

$$\begin{aligned}
\text{Search Cost} &= \log_2 n \times \sqrt[\log_2 n]{n} \\
&= \log_2 n \times (n)^{1/\log_2 n} \\
&= \log_2 n \times (n)^{\log_n 2} \\
&= \log_2 n \times 2 \\
&= \Theta(\log_2 n)
\end{aligned}$$

Now

For a three layer link list data structure

We get a search cost of $3 \times \sqrt[3]{n}$

In general for k layers, we have

$$k \times \sqrt[k]{n}$$

Thus, if we make $k = \log_2 n$, we get

$$\begin{aligned}
\text{Search Cost} =& \log_2 n \times \sqrt[\log_2 n]{n} \\
=& \log_2 n \times (n)^{1/\log_2 n} \\
=& \log_2 n \times (n)^{\log_n 2} \\
=& \log_2 n \times 2 \\
=& \Theta\left(\log_2 n\right)
\end{aligned}$$

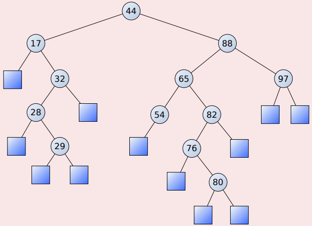
Thus

Something Notable

We get the advantages of the binary search trees with a simpler architecture!!!

Thus

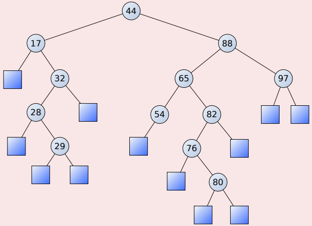
Binary Search Trees



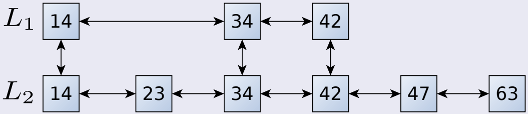
New Architecture

Thus

Binary Search Trees



New Architecture



Problem!!!

If we decided to have a deterministic algorithm

- We need to decide how to do
  - Insertion
  - Deletions

We can simplify them

- By using probabilities

Problem!!!

If we decided to have a deterministic algorithm

- We need to decide how to do
  - Insertion
  - Deletions

We can simplify them

- By using probabilities

Thus

We are ready to give a

Definition for Skip List

Outline

A Little Bit of History

Skip List

They were invented by William Worthington "Bill" Pugh Jr.!!!

A Little Bit of History

Skip List

They were invented by William Worthington "Bill" Pugh Jr.!!!

How is him?

- He is is an American computer scientist who invented the skip list and the Omega test for deciding Presburger arithmetic.
- He was the co-author of the static code analysis tool FindBugs.
- He was highly influential in the development of the current memory model of the Java language together with his PhD student Jeremy Manson.

A Little Bit of History

Skip List

They were invented by William Worthington "Bill" Pugh Jr.!!!

How is him?

- He is is an American computer scientist who invented the skip list and the Omega test for deciding Presburger arithmetic.
- He was the co-author of the static code analysis tool FindBugs.
- He was highly influential in the development of the current memory model of the Java language together with his PhD student Jeremy Manson.

A Little Bit of History

Skip List

They were invented by William Worthington "Bill" Pugh Jr.!!!

How is him?

- He is is an American computer scientist who invented the skip list and the Omega test for deciding Presburger arithmetic.
- He was the co-author of the static code analysis tool FindBugs.
- He was highly influential in the development of the current memory model of the Java language together with his PhD student Jeremy Manson.

Skip List Definition

Definition

A skip list for a set S of distinct (key,element) items is a series of lists $S_0, S_1, ..., S_h$ such that:

- Each list S_i contains the special keys $+\infty$ and $-\infty$
- List S_0 contains the keys of S in nondecreasing order
- Each list is a subsequence of the previous one
  - $S_0 \supseteq S_1 \supseteq S_2 \supseteq ... \supseteq S_h$
- List S_h contains only the two special keys

Skip List Definition

Definition

A skip list for a set S of distinct (key,element) items is a series of lists $S_0, S_1, ..., S_h$ such that:

- Each list S_i contains the special keys $+\infty$ and $-\infty$
- List S_0 contains the keys of S in nondecreasing order
- Each list is a subsequence of the previous one
  - $S_0 \supseteq S_1 \supseteq S_2 \supseteq ... \supseteq S_h$
- List S_h contains only the two special keys

Skip List Definition

Definition

A skip list for a set S of distinct (key,element) items is a series of lists $S_0, S_1, ..., S_h$ such that:

- Each list S_i contains the special keys $+\infty$ and $-\infty$
- List S_0 contains the keys of S in nondecreasing order
- Each list is a subsequence of the previous one
  - $S_0 \supseteq S_1 \supseteq S_2 \supseteq ... \supseteq S_h$
- List S_h contains only the two special keys

Skip List Definition

Definition

A skip list for a set S of distinct (key,element) items is a series of lists $S_0, S_1, ..., S_h$ such that:

- Each list S_i contains the special keys $+\infty$ and $-\infty$
- List S_0 contains the keys of S in nondecreasing order
- Each list is a subsequence of the previous one
  - $S_0 \supseteq S_1 \supseteq S_2 \supseteq ... \supseteq S_h$
- List S_h contains only the two special keys

Skip List Definition

Definition

A skip list for a set S of distinct (key,element) items is a series of lists $S_0, S_1, ..., S_h$ such that:

- Each list S_i contains the special keys $+\infty$ and $-\infty$
- List S_0 contains the keys of S in nondecreasing order
- Each list is a subsequence of the previous one
  - $S_0 \supseteq S_1 \supseteq S_2 \supseteq ... \supseteq S_h$
- List S_h contains only the two special keys

Skip List Definition

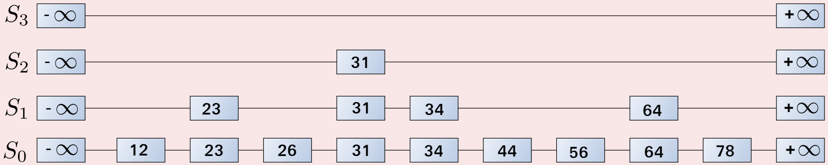
Definition

A skip list for a set S of distinct (key,element) items is a series of lists $S_0, S_1, ..., S_h$ such that:

- Each list S_i contains the special keys $+\infty$ and $-\infty$
- List S_0 contains the keys of S in nondecreasing order
- Each list is a subsequence of the previous one
  - $S_0 \supseteq S_1 \supseteq S_2 \supseteq ... \supseteq S_h$
- List S_h contains only the two special keys

Skip List Definition

Example



Skip list search

We search for a key x in a skip list as follows

- We start at the first position of the top list.
- At the current position p, we compare x with $y == p.next.key$
  - $x == y$: we return $p.next.element$
  - $x > y$: we scan forward
  - $x < y$: we "drop down"
- If we try to drop down past the bottom list, we return $null$.

Skip list search

We search for a key x in a skip list as follows

- We start at the first position of the top list.
- At the current position p, we compare x with $y == p.next.key$
  - $x == y$: we return $p.next.element$
  - $x > y$: we scan forward
  - $x < y$: we "drop down"
- If we try to drop down past the bottom list, we return $null$.

Skip list search

We search for a key x in a skip list as follows

- We start at the first position of the top list.
- At the current position p, we compare x with $y == p.next.key$
  - $x == y$: we return $p.next.element$
  - $x > y$: we scan forward
  - $x < y$: we "drop down"
- If we try to drop down past the bottom list, we return $null$.

Skip list search

We search for a key x in a skip list as follows

- We start at the first position of the top list.
- At the current position p, we compare x with $y == p.next.key$
  - $x == y$: we return $p.next.element$
  - $x > y$: we scan forward
  - $x < y$: we "drop down"
- If we try to drop down past the bottom list, we return $null$.

Skip list search

We search for a key x in a skip list as follows

- We start at the first position of the top list.
- At the current position p, we compare x with $y == p.next.key$
  - $x == y$: we return $p.next.element$
  - $x > y$: we scan forward
  - $x < y$: we "drop down"
- If we try to drop down past the bottom list, we return $null$.

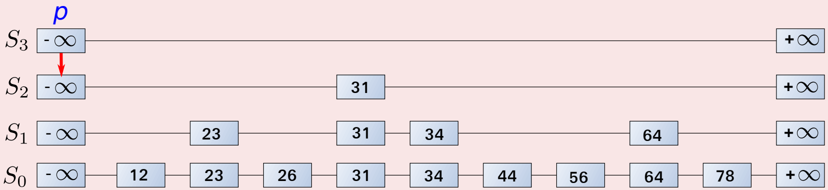
Skip list search

We search for a key x in a skip list as follows

- We start at the first position of the top list.
- At the current position p, we compare x with $y == p.next.key$
  - $x == y$: we return $p.next.element$
  - $x > y$: we scan forward
  - $x < y$: we "drop down"
- If we try to drop down past the bottom list, we return $null$.

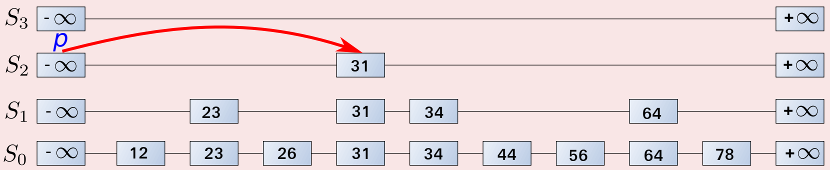
Example search for 78

$x < p.next.key$: "drop down"



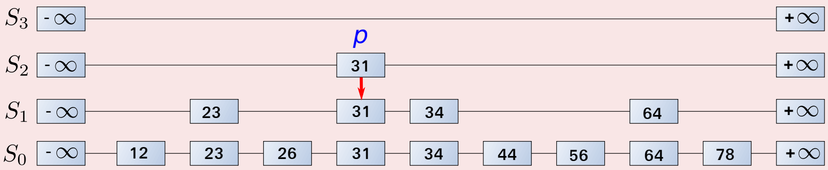
Example search for 78

$x > p.next.key$: "scan forward"



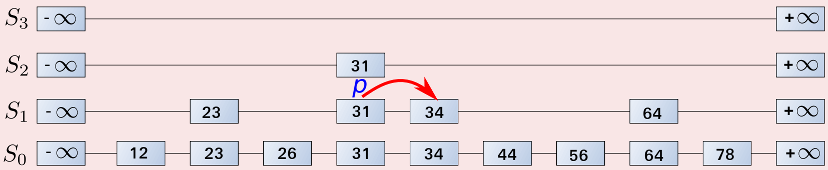
Example search for 78

$x < p.next.key$: "drop down"



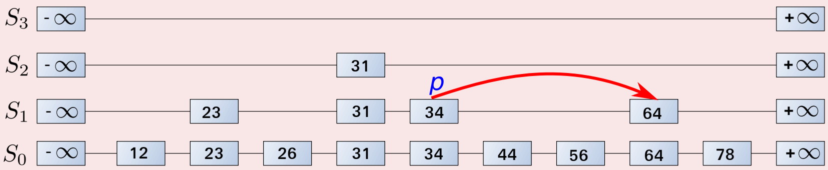
Example search for 78

$x > p.next.key$: "scan forward"



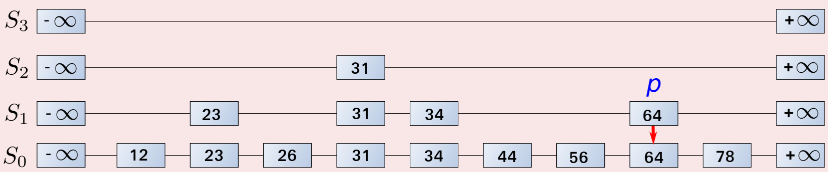
Example search for 78

$x > p.next.key$: "scan forward"



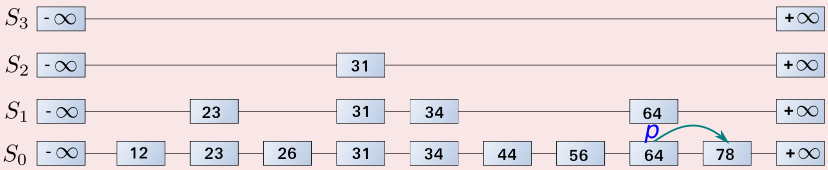
Example search for 78

$x < p.next.key$: "drop down"



Example search for 78

$x == y$: we return $p.next.element$



Outline

How do we implement this data structure?

We can implement a skip list with quad-nodes

A quad-node stores:

1. Entry Value
2. Link to the previous node
3. Link to the next node
4. Link to the above node
5. Link to the below node

Also we define special keys PLUS_INF and MINUS_INF, and we modify the key comparator to handle them.

How do we implement this data structure?

We can implement a skip list with quad-nodes

A quad-node stores:

- Entry Value
- Link to the previous node
- Link to the next node
- Link to the above node
- Link to the below node

Also we define special keys PLUS_INF and MINUS_INF, and we modify the key comparator to handle them.

How do we implement this data structure?

We can implement a skip list with quad-nodes

A quad-node stores:

- Entry Value
- Link to the previous node
- Link to the next node
- Link to the above node
- Link to the below node

Also we define special keys PLUS_INF and MINUS_INF, and we modify the key comparator to handle them.

How do we implement this data structure?

We can implement a skip list with quad-nodes

A quad-node stores:

- Entry Value
- Link to the previous node
- Link to the next node
- Link to the above node
- Link to the below node

Also we define special keys PLUS_INF and MINUS_INF, and we modify the key comparator to handle them.

How do we implement this data structure?

We can implement a skip list with quad-nodes

A quad-node stores:

- Entry Value
- Link to the previous node
- Link to the next node
- Link to the above node
- Link to the below node

Also we define special keys PLUS_INF and MINUS_INF, and we modify the key comparator to handle them.

How do we implement this data structure?

We can implement a skip list with quad-nodes

A quad-node stores:

- Entry Value
- Link to the previous node
- Link to the next node
- Link to the above node
- Link to the below node

Also we define special keys PLUS_INF and MINUS_INF, and we modify the key comparator to handle them.

How do we implement this data structure?

We can implement a skip list with quad-nodes

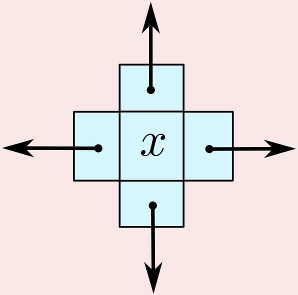
A quad-node stores:

- Entry Value
- Link to the previous node
- Link to the next node
- Link to the above node
- Link to the below node

Also we define special keys PLUS_INF and MINUS_INF, and we modify the key comparator to handle them.

Example

Quad-Node Example



Skip lists uses Randomization

Use of randomization

We use a randomized algorithm to insert items into a skip list.

Running time

We analyze the expected running time of a randomized algorithm under the following assumptions:

- The coins are unbiased.
- The coin tosses are independent.

Skip lists uses Randomization

Use of randomization

We use a randomized algorithm to insert items into a skip list.

Running time

We analyze the expected running time of a randomized algorithm under the following assumptions:

- The coins are unbiased.
- The coin tosses are independent.

Worst case running time

The worst case running time of a randomized algorithm is often large but has very low probability.

- e.g. It occurs when all the coin tosses give **"heads."**

Skip lists uses Randomization

Use of randomization

We use a randomized algorithm to insert items into a skip list.

Running time

We analyze the expected running time of a randomized algorithm under the following assumptions:

- The coins are unbiased.
- The coin tosses are independent.

Worst case running time

The worst case running time of a randomized algorithm is often large but has very low probability.

- e.g. It occurs when all the coin tosses give **"heads."**

Skip lists uses Randomization

Use of randomization

We use a randomized algorithm to insert items into a skip list.

Running time

We analyze the expected running time of a randomized algorithm under the following assumptions:

- The coins are unbiased.
- The coin tosses are independent.

Worst case running time

The worst case running time of a randomized algorithm is often large but has very low probability.

- e.g. It occurs when all the coin tosses give **"heads."**

Skip lists uses Randomization

Use of randomization

We use a randomized algorithm to insert items into a skip list.

Running time

We analyze the expected running time of a randomized algorithm under the following assumptions:

- The coins are unbiased.
- The coin tosses are independent.

Worst case running time

The worst case running time of a randomized algorithm is often large but has very low probability.

- e.g. It occurs when all the coin tosses give **"heads."**

Skip lists uses Randomization

Use of randomization

We use a randomized algorithm to insert items into a skip list.

Running time

We analyze the expected running time of a randomized algorithm under the following assumptions:

- The coins are unbiased.
- The coin tosses are independent.

Worst case running time

The worst case running time of a randomized algorithm is often large but has very low probability.

- e.g. It occurs when all the coin tosses give **"heads."**

Outline

Insertion

To insert

To insert an entry $(key, object)$ into a skip list, we use a randomized algorithm:

1. We repeatedly toss a coin until we get tails:
   - We denote with i the number of times the coin came up heads.

Insertion

To insert

To insert an entry $(key, object)$ into a skip list, we use a randomized algorithm:

- We repeatedly toss a coin until we get tails:
  - We denote with i the number of times the coin came up heads.

Insertion

To insert

To insert an entry $(key, object)$ into a skip list, we use a randomized algorithm:

- We repeatedly toss a coin until we get tails:
  - We denote with i the number of times the coin came up heads.

We have two cases

If $i \geq h$, we add to the skip list new lists $S_{h+1}, ..., S_{i+1}$

- Each containing only the two special keys.
- We search for x in the skip list and find the positions $p_0, p_1, ..., p_i$ of the items with largest key less than x in each lists $S_0, S_1, ..., S_i$.
- For $j \leftarrow 0, ..., i$, we insert item $(key, object)$ into list S_j after position p_j.

We have two cases

If $i \geq h$, we add to the skip list new lists $S_{h+1}, ..., S_{i+1}$

- Each containing only the two special keys.
- We search for x in the skip list and find the positions $p_0, p_1, ..., p_i$ of the items with largest key less than x in each lists $S_0, S_1, ..., S_i$.
- For $j \leftarrow 0, ..., i$, we insert item $(key, object)$ into list S_j after position p_j.

If $i < h$, we do not insert new lists

- We search for x in the skip list and find the positions $p_0, p_1, ..., p_{i-1}$ of the items with largest key less than x in each lists $S_0, S_1, ..., S_{i-1}$.
- For $j \leftarrow 0, ..., i-1$, we insert item $(key, object)$ into list S_j after position p_j.

We have two cases

If $i \geq h$, we add to the skip list new lists $S_{h+1}, ..., S_{i+1}$

- Each containing only the two special keys.
- We search for x in the skip list and find the positions $p_0, p_1, ..., p_i$ of the items with largest key less than x in each lists $S_0, S_1, ..., S_i$.
- For $j \leftarrow 0, ..., i$, we insert item $(key, object)$ into list S_j after position p_j.

If $i < h$, we do not insert new lists

- We search for x in the skip list and find the positions $p_0, p_1, ..., p_{i-1}$ of the items with largest key less than x in each lists $S_0, S_1, ..., S_{i-1}$.
- For $j \leftarrow 0, ..., i-1$, we insert item $(key, object)$ into list S_j after position p_j.

We have two cases

If $i \geq h$, we add to the skip list new lists $S_{h+1}, ..., S_{i+1}$

- Each containing only the two special keys.
- We search for x in the skip list and find the positions $p_0, p_1, ..., p_i$ of the items with largest key less than x in each lists $S_0, S_1, ..., S_i$.
- For $j \leftarrow 0, ..., i$, we insert item $(key, object)$ into list S_j after position p_j.

If $i < h$, we do not insert new lists

- We search for x in the skip list and find the positions $p_0, p_1, ..., p_{i-1}$ of the items with largest key less than x in each lists $S_0, S_1, ..., S_{i-1}$.
- For $j \leftarrow 0, ..., i-1$, we insert item $(key, object)$ into list S_j after position p_j.

We have two cases

If $i \geq h$, we add to the skip list new lists $S_{h+1}, ..., S_{i+1}$

- Each containing only the two special keys.
- We search for x in the skip list and find the positions $p_0, p_1, ..., p_i$ of the items with largest key less than x in each lists $S_0, S_1, ..., S_i$.
- For $j \leftarrow 0, ..., i$, we insert item $(key, object)$ into list S_j after position p_j.

If $i < h$, we do not insert new lists

- We search for x in the skip list and find the positions $p_0, p_1, ..., p_{i-1}$ of the items with largest key less than x in each lists $S_0, S_1, ..., S_{i-1}$.
- For $j \leftarrow 0, ..., i-1$, we insert item $(key, object)$ into list S_j after position p_j.

We have two cases

If $i \geq h$, we add to the skip list new lists $S_{h+1}, ..., S_{i+1}$

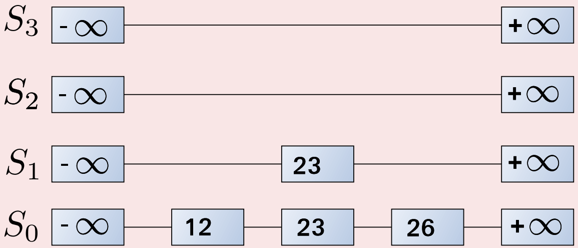
- Each containing only the two special keys.
- We search for x in the skip list and find the positions $p_0, p_1, ..., p_i$ of the items with largest key less than x in each lists $S_0, S_1, ..., S_i$.
- For $j \leftarrow 0, ..., i$, we insert item $(key, object)$ into list S_j after position p_j.

If $i < h$, we do not insert new lists

- We search for x in the skip list and find the positions $p_0, p_1, ..., p_{i-1}$ of the items with largest key less than x in each lists $S_0, S_1, ..., S_{i-1}$.
- For $j \leftarrow 0, ..., i-1$, we insert item $(key, object)$ into list S_j after position p_j.

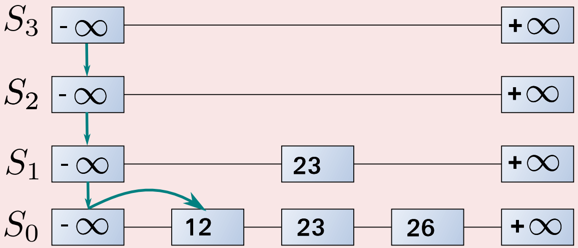
Example: Insertion of 15 in the skip list

First, we use $i = 2$ to insert S_3 into the skip list



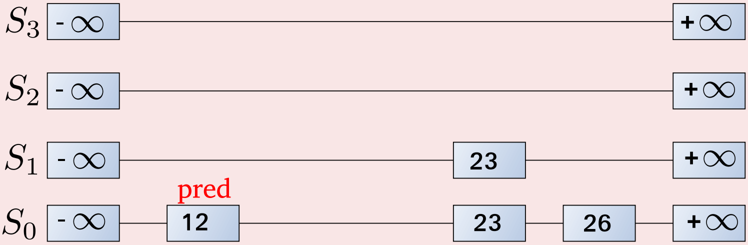
Example: Insertion of 15 in the skip list

Clearly, you first search for the predecessor key!!!



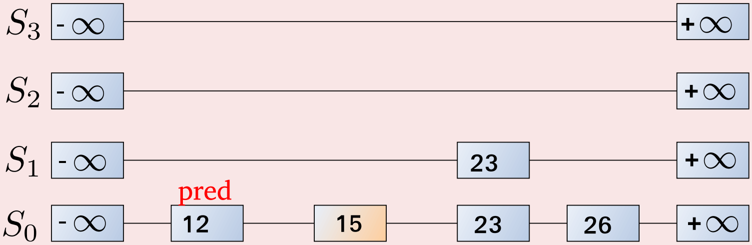
Example: Insertion of 15 in the skip list

Insert the necessary Quad-Nodes and necessary information



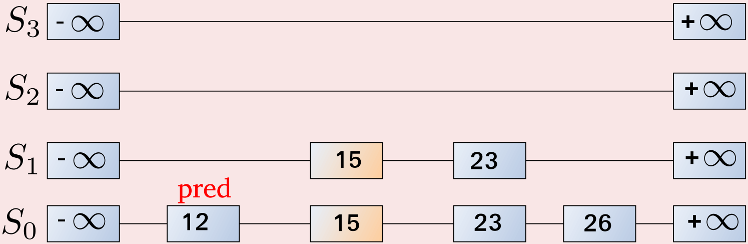
Example: Insertion of 15 in the skip list

Insert the necessary Quad-Nodes and necessary information



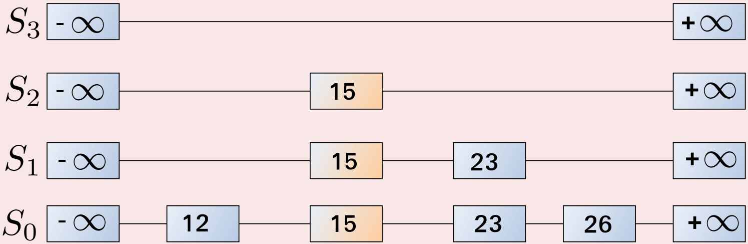
Example: Insertion of 15 in the skip list

Insert the necessary Quad-Nodes and necessary information



Example: Insertion of 15 in the skip list

Finally!!!



Outline

Deletion

To remove an entry with key x from a skip list, we proceed as follows

- We search for x in the skip list and find the positions $p_0, p_1, ..., p_i$ of the items with key x, where position p_j is in list S_j.
- We remove positions $p_0, p_1, ..., p_i$ from the lists $S_0, S_1, ..., S_i$.
- We remove all but one list containing only the two special keys

Deletion

To remove an entry with key x from a skip list, we proceed as follows

- We search for x in the skip list and find the positions $p_0, p_1, ..., p_i$ of the items with key x, where position p_j is in list S_j.
- We remove positions $p_0, p_1, ..., p_i$ from the lists $S_0, S_1, ..., S_i$.
- We remove all but one list containing only the two special keys

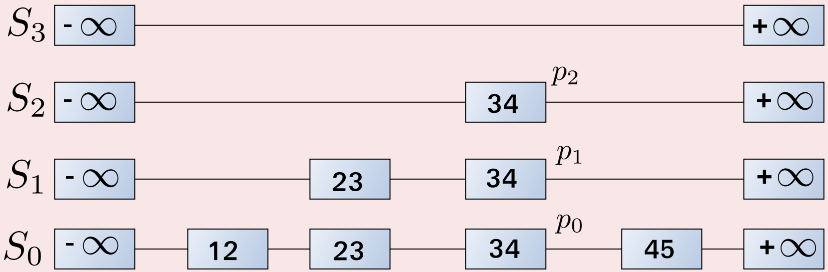
Deletion

To remove an entry with key x from a skip list, we proceed as follows

- We search for x in the skip list and find the positions $p_0, p_1, ..., p_i$ of the items with key x, where position p_j is in list S_j.
- We remove positions $p_0, p_1, ..., p_i$ from the lists $S_0, S_1, ..., S_i$.
- We remove all but one list containing only the two special keys

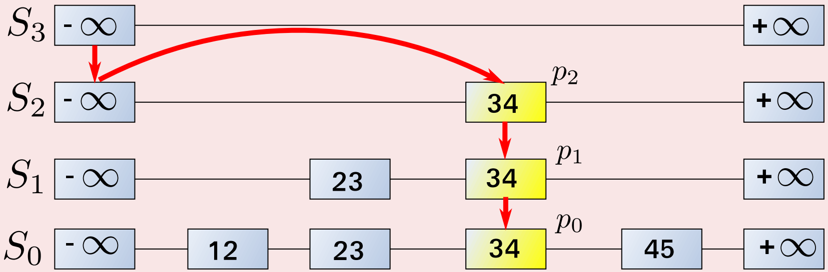
Example: Delete of 34 in the skip list

We search for 34 in the skip list and find the positions $p_0, p_1, ..., p_2$ of the items with key 34



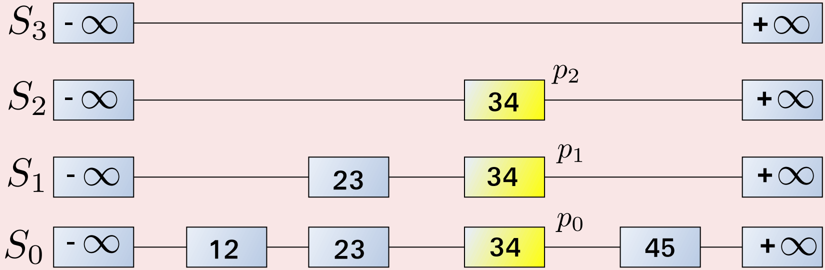
Example: Delete of 34 in the skip list

We search for 34 in the skip list and find the positions $p_0, p_1, ..., p_2$ of the items with key 34



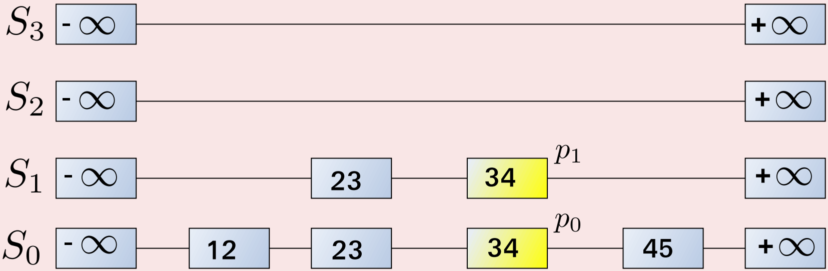
Example: Delete of 34 in the skip list

We start doing the deletion!!!



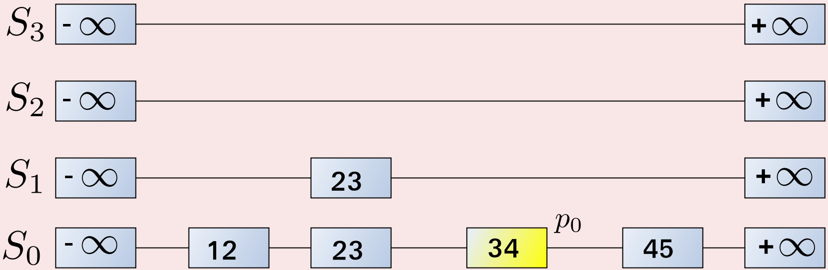
Example: Delete of 34 in the skip list

One Quad-Node after another



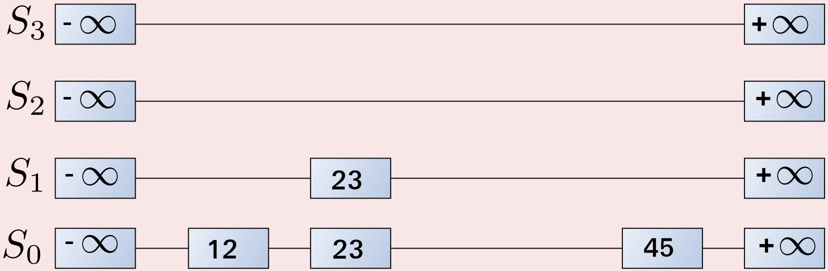
Example: Delete of 34 in the skip list

One Quad-Node after another



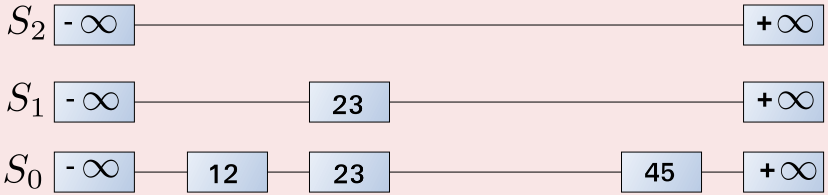
Example: Delete of 34 in the skip list

One Quad-Node after another



Example: Delete of 34 in the skip list

Remove One Level



Outline

Space usage

Space usage

The space used by a skip list depends on the random bits used by each invocation of the insertion algorithm.

Space : $O(n)$

Theorem

The expected space usage of a skip list with n items is $O(n)$.

Space : $O(n)$

Theorem

The expected space usage of a skip list with n items is $O(n)$.

Proof

We use the following two basic probabilistic facts:

1. *Fact 1:* The probability of getting i consecutive heads when flipping a coin is $\frac{1}{2^i}$.
2. *Fact 2:* If each of n entries is present in a set with probability p, the expected size of the set is np.
    1. How? Remember $X = X_1 + X_2 + \ldots + X_n$ where X_i is an indicator function for event A_i = the i element is present in the set. Thus:

$$E[X] = \underbrace{\sum_{i=1}^{n} E[X_i] = \sum_{i=1}^{n} Pr\{A_i\}}_{\text{Equivalence } E[X_A] \text{ and } Pr\{A\}} = \sum_{i=1}^{n} p = np$$

Space : $O(n)$

Theorem

The expected space usage of a skip list with n items is $O(n)$.

Proof

We use the following two basic probabilistic facts:

1. *Fact 1:* The probability of getting i consecutive heads when flipping a coin is $\frac{1}{2^i}$.
2. *Fact 2:* If each of n entries is present in a set with probability p, the expected size of the set is np.
   1. How? Remember $X = X_1 + X_2 + ... + X_n$ where X_i is an indicator function for event A_i = the i element is present in the set. Thus:

$$E[X] = \underbrace{\sum_{i=1}^{n} E[X_i] = \sum_{i=1}^{n} Pr\{A_i\}}_{\text{Equivalence } E[X_A] \text{ and } Pr\{A\}} = \sum_{i=1}^{n} p = np$$

Space : $O(n)$

Theorem

The expected space usage of a skip list with n items is $O(n)$.

Proof

We use the following two basic probabilistic facts:

1. *Fact 1:* The probability of getting i consecutive heads when flipping a coin is $\frac{1}{2^i}$.
2. *Fact 2:* If each of n entries is present in a set with probability p, the expected size of the set is np.
   1. How? Remember $X = X_1 + X_2 + ... + X_n$ where X_i is an indicator function for event A_i = the i element is present in the set. Thus:

$$E[X] = \underbrace{\sum_{i=1}^{n} E[X_i] = \sum_{i=1}^{n} Pr\{A_i\}}_{\text{Equivalence } E[X_A] \text{ and } Pr\{A\}} = \sum_{i=1}^{n} p = np$$

Space : $O(n)$

Theorem

The expected space usage of a skip list with n items is $O(n)$.

Proof

We use the following two basic probabilistic facts:

1. *Fact 1:* The probability of getting i consecutive heads when flipping a coin is $\frac{1}{2^i}$.
2. *Fact 2:* If each of n entries is present in a set with probability p, the expected size of the set is np.
    1. How? Remember $X = X_1 + X_2 + ... + X_n$ where X_i is an indicator function for event A_i = the i element is present in the set. Thus:

$$E[X] = \underbrace{\sum_{i=1}^{n} E[X_i] = \sum_{i=1}^{n} Pr\{A_i\}}_{\text{Equivalence } E[X_A] \text{ and } Pr\{A\}} = \sum_{i=1}^{n} p = np$$

Space : $O(n)$

Theorem

The expected space usage of a skip list with n items is $O(n)$.

Proof

We use the following two basic probabilistic facts:

1. *Fact 1:* The probability of getting i consecutive heads when flipping a coin is $\frac{1}{2^i}$.
2. *Fact 2:* If each of n entries is present in a set with probability p, the expected size of the set is np.
   1. How? Remember $X = X_1 + X_2 + ... + X_n$ where X_i is an indicator function for event $A_i =$ the i element is present in the set. Thus:

$$E[X] = \underbrace{\sum_{i=1}^{n} E[X_i] = \sum_{i=1}^{n} Pr\{A_i\}}_{\text{Equivalence } E[X_A] \text{ and } Pr\{A\}} = \sum_{i=1}^{n} p = np$$

Proof

Now consider a skip list with n entries

Using Fact 1, an element is inserted in list S_i with a probability of

$$P\left[x \in S_i\right] = \frac{1}{2^i}$$

Now by Fact 2

The expected size of list S_i is

$$E\left[\left|S_i\right|\right] = \frac{n}{2^i}$$

Proof

Now consider a skip list with n entries

Using Fact 1, an element is inserted in list S_i with a probability of

$$P\left[x \in S_i\right] = \frac{1}{2^i}$$

Now by Fact 2

The expected size of list S_i is

$$E\left[|S_i|\right] = \frac{n}{2^i}$$

Proof

The expected number of nodes used by the skip list with height h

$$E\left[\text{Size Skip List}\right] = \sum_{i=0}^{h} \frac{n}{2^i} = n \sum_{i=0}^{h} \frac{1}{2^i}$$

Here, we have a problem!!! What is the value of h?

Height h

First

The running time of the search and insertion algorithms is affected by the height h of the skip list.

Second

We show that with high probability, a skip list with n items has height $O(\log n)$.

Height h

First

The running time of the search and insertion algorithms is affected by the height h of the skip list.

Second

We show that with high probability, a skip list with n items has height $O(\log n)$.

For this, we have the following fact!!!

We use the following Fact 3

We can view the level $l\left(x_i\right) = \max\left\{j | \text{where } x_i \in S_j\right\}$ of the elements in the skip list as the following random variable

$$X_i = l\left(x_i\right)$$

for each element x_i in the skip list.

For this, we have the following fact!!!

We use the following Fact 3

We can view the level $l(x_i) = \max\{j | \text{where } x_i \in S_j\}$ of the elements in the skip list as the following random variable

$$X_i = l(x_i)$$

for each element x_i in the skip list.

And this is a random variable!!!

- Remember the insertions!!! Using an unbiased coin!!
- Thus, all X_i have a geometric distribution.

For this, we have the following fact!!!

We use the following Fact 3

We can view the level $l(x_i) = \max\{j | \text{where } x_i \in S_j\}$ of the elements in the skip list as the following random variable

$$X_i = l(x_i)$$

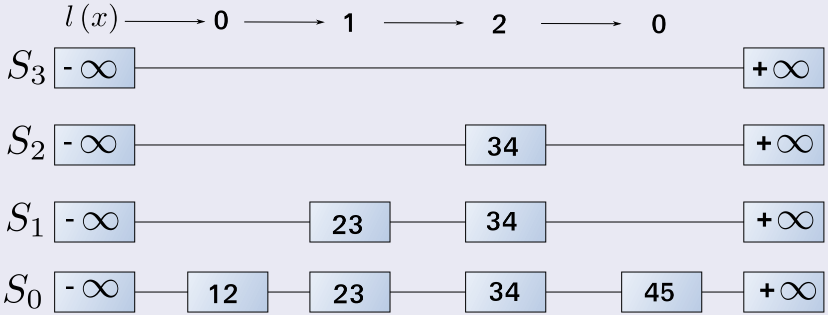
for each element x_i in the skip list.

And this is a random variable!!!

- Remember the insertions!!! Using an unbiased coin!!
- Thus, all X_i have a geometric distribution.

Example for $l(x_i)$

We have



BTW What is the geometric distribution?

k failures where

$$k = \{1, 2, 3, ...\}$$

Probability mass function

$$Pr\left(X = k\right) = \left(1 - p\right)^{k-1} p$$

BTW What is the geometric distribution?

k failures where

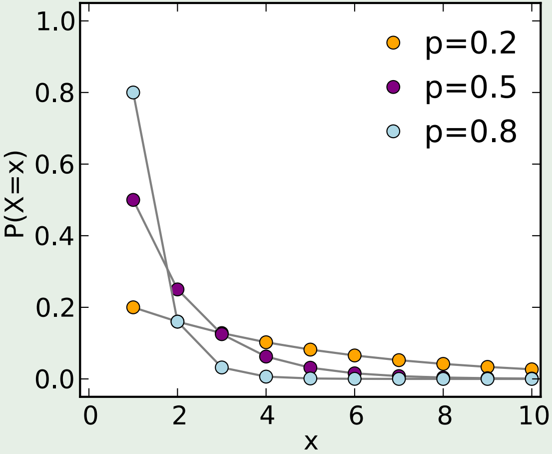
$$k = \{1, 2, 3, ...\}$$

Probability mass function

$$Pr\left(X = k\right) = \left(1 - p\right)^{k-1} p$$

Probability Mass Function

For Different Probabilities



Then

We have the following inequality for the geometric variables

$$Pr\left[X_i > t\right] \leq (1-p)^t \;\; \forall i = 1, 2, ..., n$$

- If we assume we have a fair coin $p = \frac{1}{2}$

This is because

$$F(t) = P\left[X_i \leq t\right] = \sum_{i=1}^{t} (1-p)^{i-1} p$$

Then

We have the following inequality for the geometric variables

$$Pr\left[X_i > t\right] \leq (1-p)^t \;\; \forall i = 1, 2, ..., n$$

- If we assume we have a fair coin $p = \frac{1}{2}$

This is because

$$F(t) = P\left[X_i \leq t\right] = \sum_{i=1}^{t} (1-p)^{i-1} p$$

Then, we have

Then, we have that

$$Pr\left[X_i > t\right] = \sum_{i=t+1}^{\infty} (1-p)^{i-1} p = 1$$

Then, we have

Then, we have that

$$Pr\left[X_i > t\right] = \sum_{i=t+1}^{\infty} (1-p)^{i-1} p = 1$$

Thus, we have that

$$\sum_{i=t+1}^{\infty} (1-p)^{i-1} p = p \sum_{i=t+1}^{\infty} (1-p)^{i-1}$$

$$= p \sum_{k=1, k=i-t}^{\infty} (1-p)^{k+t-1}$$

$$= p(1-p)^t \sum_{k=1, k=i-t}^{\infty} (1-p)^{k-1}$$

$$= (1-p)^t \frac{p}{1-p} \leq (1-t)^t \text{ Given the fair coin}$$

Then, we have

Then, we have that

$$Pr\left[X_i > t\right] = \sum_{i=t+1}^{\infty} (1-p)^{i-1} p = 1$$

Thus, we have that

$$\begin{aligned}
\sum_{i=t+1}^{\infty} (1-p)^{i-1} p = p \sum_{i=t+1}^{\infty} (1-p)^{i-1} \\
= p \sum_{k=1, k=i-t}^{\infty} (1-p)^{k+t-1} \\
= p (1-p)^{t} \sum_{k=1, k=i-t}^{\infty} (1-p)^{k-1} \\
= (1-p)^{t} \frac{p}{1-p} \leq (1-t)^{t} \text{ Given the fair coin}
\end{aligned}$$

Then, we have

Then, we have that

$$Pr\left[X_i > t\right] = \sum_{i=t+1}^{\infty} (1-p)^{i-1} p = 1$$

Thus, we have that

$$\begin{aligned}
\sum_{i=t+1}^{\infty} (1-p)^{i-1} p &= p \sum_{i=t+1}^{\infty} (1-p)^{i-1} \\
&= p \sum_{k=1, k=i-t}^{\infty} (1-p)^{k+t-1} \\
&= p\,(1-p)^{t} \sum_{k=1, k=i-t}^{\infty} (1-p)^{k-1} \\
&= (1-p)^{t} \frac{p}{1-p} \leq (1-t)^{t} \text{ Given the fair coin}
\end{aligned}$$

Then, we have

Then, we have that

$$Pr\left[X_i > t\right] = \sum_{i=t+1}^{\infty} (1-p)^{i-1} p = 1$$

Thus, we have that

$$\begin{aligned}
\sum_{i=t+1}^{\infty} (1-p)^{i-1} p &= p \sum_{i=t+1}^{\infty} (1-p)^{i-1} \\
&= p \sum_{k=1, k=i-t}^{\infty} (1-p)^{k+t-1} \\
&= p\,(1-p)^{t} \sum_{k=1, k=i-t}^{\infty} (1-p)^{k-1} \\
&= (1-p)^{t} \frac{p}{1-p} \leq (1-t)^{t} \text{ Given the fair coin}
\end{aligned}$$

Then, we have

Using our original formula

$$Pr\left[X_i > t\right] \leq \left(1 - t\right)^t$$

In this way, we have

Then, we have

$$Pr\left\{\max_i X_i > t\right\} \leq n(1-p)^t$$

How?

We have that

$$Pr\left\{\max_i X_i > t\right\} = Pr\left\{\max\left\{X_1, X_2, ..., X_n\right\} > t\right\}$$
$$= \sum_{i=1}^{n} Pr\left\{X_i > t \text{ and } X_i = \max\left\{X_1, X_2, ..., X_n\right\}\right\}$$

How?

- That one of the elements becomes the maximum in height and a height greater than t

How?

We have that

$$
\begin{aligned}
Pr\left\{\max_i X_i > t\right\} &= Pr\left\{\max\left\{X_1, X_2, ..., X_n\right\} > t\right\} \\
&= \sum_{i=1}^{n} Pr\left\{X_i > t \text{ and } X_i = \max\left\{X_1, X_2, ..., X_n\right\}\right\}
\end{aligned}
$$

How?

- That one of the elements becomes the maximum in height and a height greater than t

Why?

Because the height of an element depends on independent event

- Each toss coin until tails is independent of the others!!!

Example

When having two lists

$$\{\max(X_1, X_2) > t\} = \{X_1 > t \text{ and } X_1 > X_2 \text{ or exclusive } X_2 > t \text{ and } X_2 > X_1\}$$

- Yes, you need to remember that the max is a single element not both...

Therefore

Then

$$
\begin{aligned}
Pr\left\{X_1 > t \text{ and } X_1 > X_2 \text{ or exclusive } X_2 > t\right\} = & Pr\left\{X_1 > t \text{ and } X_1 > X_2\right\} + \ldots \\
& Pr\left\{X_2 > t \text{ and } X_2 > X_1\right\}
\end{aligned}
$$

Assuming exclusivity between phenomena $X_i > X_j$ and $X_i > t$

$$
\begin{aligned}
Pr\left\{X_1 > t \text{ and } X_1 > X_2 \text{ or exclusive } X_2 > t\right\} = & P\left(X_1 > t\right) P\left(X_1 > X_2\right) + \ldots \\
& P\left(X_2 > t\right) P\left(X_2 > X_1\right) \\
\leq & P\left(X_1 > t\right) + P\left(X_2 > t\right)
\end{aligned}
$$

Therefore

Then

$$
\begin{aligned}
Pr\left\{X_1 > t \text{ and } X_1 > X_2 \text{ or exclusive } X_2 > t\right\} =& Pr\left\{X_1 > t \text{ and } X_1 > X_2\right\} + ...\\
& Pr\left\{X_2 > t \text{ and } X_2 > X_1\right\}
\end{aligned}
$$

Assuming exclusivity between phenomena $X_i > X_j$ and $X_i > t$

$$
\begin{aligned}
Pr\left\{X_1 > t \text{ and } X_1 > X_2 \text{ or exclusive } X_2 > t\right\} =& P\left(X_1 > t\right) P\left(X_1 > X_2\right) + ...\\
& P\left(X_2 > t\right) P\left(X_2 > X_1\right)\\
\leq & P\left(X_1 > t\right) + P\left(X_2 > t\right)
\end{aligned}
$$

This gives us something

We have that

$$Pr\left\{X_i > t \text{ and } X_i = \max\left\{X_i\right\}_{i=1}^{n}\right\} = Pr\left\{X_i > t\right\} P\left\{X_i = \max\left\{X_i\right\}_{i=1}^{n}\right\}$$

Then, we can say that

$$Pr\left\{X_i > t \text{ and } X_i = \max\left\{X_i\right\}_{i=1}^{n}\right\} \leq Pr\left\{X_i > t\right\}$$

This gives us something

We have that

$$Pr\left\{X_i > t \text{ and } X_i = \max\left\{X_i\right\}_{i=1}^{n}\right\} = Pr\left\{X_i > t\right\} P\left\{X_i = \max\left\{X_i\right\}_{i=1}^{n}\right\}$$

Then, we can say that

$$Pr\left\{X_i > t \text{ and } X_i = \max\left\{X_i\right\}_{i=1}^{n}\right\} \leq Pr\left\{X_i > t\right\}$$

Finally, using this fact

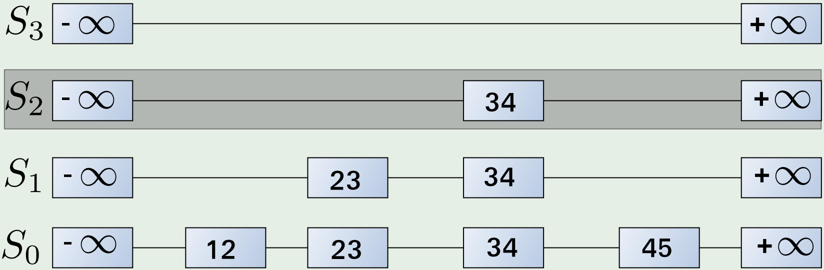
We have when summing over all events X_i

$$\sum_{i=1}^{n} Pr\left\{X_i > t\right\} \leq \sum_{i=1}^{n} \left(1-p\right)^t = n\left(1-p\right)^t$$

An Observation

The $\max_i X_i$

It represents the list with the one entry apart from the special keys.



Another One

Also REMEMBER!!!

We are talking about a fair coin, thus $p = \frac{1}{2}$.

Outline

Height: $3\log_2 n$ with probability at least $1 - \frac{1}{n^2}$

Theorem

A skip list with n entries has height at most $3\log_2 n$ with probability at least $1 - \frac{1}{n^2}$

Proof

Consider a skip list with n entires

By Fact 3, the probability that list S_t has at least one item (The $\max_i X_i > t$) is at most $\frac{n}{2^t}$.

$$P\left(|S_t| \geq 1\right) = P\left(\max_i X_i > t\right) \leq \frac{n}{2^t}.$$

By picking $t = 3\log n$

We have that the probability that $S_{3\log_2 n}$ has at least one entry is at most:

$$\frac{n}{2^{3\log_2 n}} = \frac{n}{n^3} = \frac{1}{n^2}.$$

Proof

Consider a skip list with n entires

By Fact 3, the probability that list S_t has at least one item (The $\max_i X_i > t$) is at most $\frac{n}{2^t}$.

$$P\left(|S_t| \geq 1\right) = P\left(\max_i X_i > t\right) \leq \frac{n}{2^t}.$$

By picking $t = 3\log n$

We have that the probability that $S_{3\log_2 n}$ has at least one entry is at most:

$$\frac{n}{2^{3\log_2 n}} = \frac{n}{n^3} = \frac{1}{n^2}.$$

Look at we want to model

We want to model

- The height of the Skip List is at most $t = 3\log_2 n$
- Equivalent to the negation of having list $S_{3\log_2 n}$

Look at we want to model

We want to model

- The height of the Skip List is at most $t = 3\log_2 n$
- Equivalent to the negation of having list $S_{3\log_2 n}$

Then, the probability that the height $h = 3\log_2 n$ of the skip list is

$$P\left(\text{Skip List height } 3\log_2 n\right) = 1 - \frac{1}{n^2}$$

Look at we want to model

We want to model

- The height of the Skip List is at most $t = 3\log_2 n$
- Equivalent to the negation of having list $S_{3\log_2 n}$

Then, the probability that the height $h = 3\log_2 n$ of the skip list is

$$P\left(\text{Skip List height } 3\log_2 n\right) = 1 - \frac{1}{n^2}$$

Finally

The expected number of nodes used by the skip list with height h

- Given that $h = 3\log_2 n$

$$\sum_{i=0}^{3\log_2 n} \frac{n}{2^i} = n \sum_{i=0}^{3\log_2 n} \frac{1}{2^i}$$

Given the geometric sum

$$S_m = \sum_{k=0}^{m} r^k = \frac{1 - r^{m+1}}{1 - r}$$

Finally

The expected number of nodes used by the skip list with height h

- Given that $h = 3\log_2 n$

$$\sum_{i=0}^{3\log_2 n} \frac{n}{2^i} = n \sum_{i=0}^{3\log_2 n} \frac{1}{2^i}$$

Given the geometric sum

$$S_m = \sum_{k=0}^{m} r^k = \frac{1 - r^{m+1}}{1 - r}$$

We have finally

The Upper Bound on the number of nodes

$$n\sum_{i=0}^{3\log_2 n}\frac{1}{2^i} = n\left(\frac{1-\left(\frac{1}{2}\right)^{3\log_2 n+1}}{1-1/2}\right)$$

$$= n\left(\frac{1-\frac{1}{2^{3\log_2 n+1}}}{1/2}\right)$$

$$= n\left(\frac{1-\frac{1}{\left(2^{\log_2 n}\right)^3 2}}{1/2}\right)$$

$$= n\left(\frac{1-\frac{1}{2n^3}}{1/2}\right) = n\left(\frac{2\left[2n^3-1\right]}{2n^3}\right)$$

We have finally

The Upper Bound on the number of nodes

$$
\begin{aligned}
n\sum_{i=0}^{3\log_2 n}\frac{1}{2^i} &= n\left(\frac{1-\left(\frac{1}{2}\right)^{3\log_2 n+1}}{1-{}^1\!/_2}\right)\\
&= n\left(\frac{1-\frac{1}{2^{3\log_2 n+1}}}{{}^1\!/_2}\right)\\
&= n\left(\frac{1-\frac{1}{\left(2^{\log_2 n}\right)^3 2}}{{}^1\!/_2}\right)\\
&= n\left(\frac{1-\frac{1}{2n^3}}{{}^1\!/_2}\right) = n\left(\frac{2\left[2n^3-1\right]}{2n^3}\right)
\end{aligned}
$$

We have finally

The Upper Bound on the number of nodes

$$n\sum_{i=0}^{3\log_2 n}\frac{1}{2^i} = n\left(\frac{1-\left(\frac{1}{2}\right)^{3\log_2 n+1}}{1-{}^1\!/\!_2}\right)$$

$$= n\left(\frac{1-\frac{1}{2^{3\log_2 n+1}}}{{}^1\!/\!_2}\right)$$

$$= n\left(\frac{1-\frac{1}{\left(2^{\log_2 n}\right)^3 2}}{{}^1\!/\!_2}\right)$$

$$= n\left(\frac{1-\frac{1}{2n^3}}{{}^1\!/\!_2}\right) = n\left(\frac{2\left[2n^3-1\right]}{2n^3}\right)$$

We have finally

The Upper Bound on the number of nodes

$$
\begin{aligned}
n\sum_{i=0}^{3\log_2 n}\frac{1}{2^i} =& n\left(\frac{1-\left(\frac{1}{2}\right)^{3\log_2 n+1}}{1-{}^1\!/_2}\right)\\
=& n\left(\frac{1-\frac{1}{2^{3\log_2 n+1}}}{{}^1\!/_2}\right)\\
=& n\left(\frac{1-\frac{1}{\left(2^{\log_2 n}\right)^3 2}}{{}^1\!/_2}\right)\\
=& n\left(\frac{1-\frac{1}{2n^3}}{{}^1\!/_2}\right) = n\left(\frac{2\left[2n^3-1\right]}{2n^3}\right)
\end{aligned}
$$

Finally

We have

$$\left(\frac{2n^3 - 1}{n^2}\right) = 2n - \frac{1}{n^2} \leq 2n$$

The Upper Bound with probability $1 - \frac{1}{n^2}$

$$2n - \frac{1}{n^2} \leq 2n = O(n)$$

Finally

We have

$$\left(\frac{2n^3-1}{n^2}\right)=2n-\frac{1}{n^2}\leq 2n$$

The Upper Bound with probability $1-\frac{1}{n^2}$

$$2n-\frac{1}{n^2}\leq 2n=O\left(n\right)$$

Outline

Search and Insertion Times

Fact 4

The expected number of coin tosses required in order to get tails is 2:

$$\text{Given that } x \sim G\left(\frac{1}{2}\right) \Longrightarrow E\left[x\right] = \frac{1}{p} = 2 \text{ (Fair Coin Assumption)}$$

We use this

To prove that a search in a skip list takes $O(\log n)$ expected time.

- After all insertions require searches!!!

Search and Insertion Times

Fact 4

The expected number of coin tosses required in order to get tails is 2:

$$\text{Given that } x \sim G\left(\frac{1}{2}\right) \implies E\left[x\right] = \frac{1}{p} = 2 \text{ (Fair Coin Assumption)}$$

We use this

To prove that a search in a skip list takes $O(\log n)$ expected time.

- After all insertions require searches!!!

Search and Insertions times

Search time

The search time in skip list is proportional to

the number of drop-down steps + the number of scan-forward steps

Drop-down steps

The drop-down steps are bounded by the height of the skip list and thus are $O(\log_2 n)$ with high probability.

Theorem

A search in a skip list takes $O(\log_2 n)$ expected time.

Search and Insertions times

Search time

The search time in skip list is proportional to

the number of drop-down steps + the number of scan-forward steps

Drop-down steps

The drop-down steps are bounded by the height of the skip list and thus are $O(\log_2 n)$ with high probability.

Theorem

A search in a skip list takes $O(\log_2 n)$ expected time.

Search and Insertions times

Search time

The search time in skip list is proportional to

the number of drop-down steps + the number of scan-forward steps

Drop-down steps

The drop-down steps are bounded by the height of the skip list and thus are $O(\log_2 n)$ with high probability.

Theorem

A search in a skip list takes $O(\log_2 n)$ expected time.

Proof

First

- When we scan forward in a list, the destination key does not belong to a higher list.

A scan-forward step is associated with a former coin toss that gave tails

- By Fact 4, in each list the expected number of scan-forward steps is 2.

Proof

First

- When we scan forward in a list, the destination key does not belong to a higher list.

A scan-forward step is associated with a former coin toss that gave tails

- By Fact 4, in each list the expected number of scan-forward steps is 2.

Why?

Given the list S_i

- Then, the scan-forward intervals (Jumps between x_i and x_{i+1}) to the right of S_i are

$$I_1 = [x_1, x_2], I_2 = [x_2, x_3] \ldots I_k = [x_k, +\infty]$$

Then

These interval exist at level i if and only if all $x_1, x_2, \ldots, x_k$ belong to S_i.

Why?

Given the list S_i

- Then, the scan-forward intervals (Jumps between x_i and x_{i+1}) to the right of S_i are

$$I_1 = [x_1, x_2], I_2 = [x_2, x_3] ... I_k = [x_k, +\infty]$$

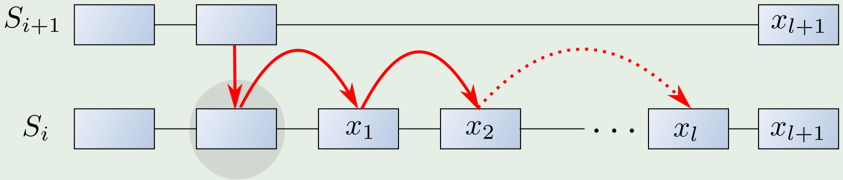
Then

These interval exist at level i if and only if all $x_1, x_2, ..., x_k$ belong to S_i.

We introduce the following concept based on these intervals

Scan-forward siblings

These are element that you find in the search path before finding an element in the upper list.



Now

Given that a search is being done, S_i contains l forward siblings

It must be the case that given $x_1, ..., x_l$ scan-forward siblings, we have that

$$x_1, ..., x_l \notin S_{i+1}$$

and $x_{l+1} \in S_{i+1}$

Thus

We have

Since each element of S_i is independently chosen to be in S_{i+1} with probability $p = \frac{1}{2}$.

We have

The number of scan-forward siblings is bounded by a geometric random variable X_i with parameter $p = \frac{1}{2}$.

- Imagine the fact that you have multiple fails... then $x_1, ..., x_l \notin S_{i+1}$ is modeled by X_i

Thus, we have that

The expected number of scan-forward siblings is bounded by 2!!!

$$\text{Expected \# Scan-Forward Siblings at } i \leq \underbrace{E[X_i] = \frac{1}{1/2}}_{\text{Mean}} = 2$$

Thus

We have

Since each element of S_i is independently chosen to be in S_{i+1} with probability $p = \frac{1}{2}$.

We have

The number of scan-forward siblings is bounded by a geometric random variable X_i with parameter $p = \frac{1}{2}$.

- Imagine the fact that you have multiple fails... then $x_1, ..., x_l \notin S_{i+1}$ is modeled by X_i

Thus, we have that

The expected number of scan-forward siblings is bounded by 2!!!

$$\text{Expected \# Scan-Fordward Siblings at } i \leq E[X_i] = \underbrace{\frac{1}{1/2}}_{\text{Mean}} = 2$$

Thus

We have

Since each element of S_i is independently chosen to be in S_{i+1} with probability $p = \frac{1}{2}$.

We have

The number of scan-forward siblings is bounded by a geometric random variable X_i with parameter $p = \frac{1}{2}$.

- Imagine the fact that you have multiple fails... then $x_1, ..., x_l \notin S_{i+1}$ is modeled by X_i

Thus, we have that

The expected number of scan-forward siblings is bounded by 2!!!

$$\text{Expected } \# \text{ Scan-Fordward Siblings at } i \leq \underbrace{E\left[X_i\right] = \frac{1}{1/2}}_{\text{Mean}} = 2$$

Then

In the worst case scenario

A search is bounded by $O(\log_2 n) + 2\log_2 n = O(\log_2 n)$

An given that a insertion is a (search) + (deletion bounded by the height)

Thus, an insertion is bounded by $2\log_2 n + 3\log_2 n = O(\log_2 n)$

Then

In the worst case scenario

A search is bounded by $O(\log_2 n) + 2\log_2 n = O(\log_2 n)$

An given that a insertion is a (search) + (deletion bounded by the height)

Thus, an insertion is bounded by $2\log_2 n + 3\log_2 n = O(\log_2 n)$

Outline

Applications

We have

- Cyrus IMAP servers offer a "skiplist" backend Data Base implementation.
- Lucene uses skip lists to search delta-encoded posting lists in logarithmic time.
- Redis, an ANSI-C open-source persistent key/value store for Posix systems, uses skip lists in its implementation of ordered sets.
- leveldb, a fast key-value storage library written at Google that provides an ordered mapping from string keys to string values.
- Skip lists are used for efficient statistical computations of running medians.

Applications

We have

- Cyrus IMAP servers offer a "skiplist" backend Data Base implementation.
- Lucene uses skip lists to search delta-encoded posting lists in logarithmic time.
- Redis, an ANSI-C open-source persistent key/value store for Posix systems, uses skip lists in its implementation of ordered sets.
- leveldb, a fast key-value storage library written at Google that provides an ordered mapping from string keys to string values.
- Skip lists are used for efficient statistical computations of running medians.

Applications

We have

- Cyrus IMAP servers offer a "skiplist" backend Data Base implementation.
- Lucene uses skip lists to search delta-encoded posting lists in logarithmic time.
- Redis, an ANSI-C open-source persistent key/value store for Posix systems, uses skip lists in its implementation of ordered sets.
- leveldb, a fast key-value storage library written at Google that provides an ordered mapping from string keys to string values.
- Skip lists are used for efficient statistical computations of running medians.

Applications

We have

- Cyrus IMAP servers offer a "skiplist" backend Data Base implementation.
- Lucene uses skip lists to search delta-encoded posting lists in logarithmic time.
- Redis, an ANSI-C open-source persistent key/value store for Posix systems, uses skip lists in its implementation of ordered sets.
- leveldb, a fast key-value storage library written at Google that provides an ordered mapping from string keys to string values.
- Skip lists are used for efficient statistical computations of running medians.

Applications

We have

- Cyrus IMAP servers offer a "skiplist" backend Data Base implementation.
- Lucene uses skip lists to search delta-encoded posting lists in logarithmic time.
- Redis, an ANSI-C open-source persistent key/value store for Posix systems, uses skip lists in its implementation of ordered sets.
- leveldb, a fast key-value storage library written at Google that provides an ordered mapping from string keys to string values.
- Skip lists are used for efficient statistical computations of running medians.

Outline

Summary

Summary

- A skip list is a data structure for dictionaries that uses a randomized insertion algorithm.
- In a skip list with n entries:
  - The expected space used is $O(n)$
  - The expected search, insertion and deletion time is $O(\log n)$
- Skip list are fast and simple to implement in practice.

Summary

Summary

- A skip list is a data structure for dictionaries that uses a randomized insertion algorithm.
- In a skip list with n entries:
  - The expected space used is $O(n)$
  - The expected search, insertion and deletion time is $O(\log n)$
- Skip list are fast and simple to implement in practice.

Summary

Summary

- A skip list is a data structure for dictionaries that uses a randomized insertion algorithm.
- In a skip list with n entries:
  - The expected space used is $O(n)$
  - The expected search, insertion and deletion time is $O(\log n)$
- Skip list are fast and simple to implement in practice.

Summary

Summary

- A skip list is a data structure for dictionaries that uses a randomized insertion algorithm.
- In a skip list with n entries:
    - The expected space used is $O(n)$
    - The expected search, insertion and deletion time is $O(\log n)$
- Skip list are fast and simple to implement in practice.

Summary

Summary

- A skip list is a data structure for dictionaries that uses a randomized insertion algorithm.
- In a skip list with n entries:
  - The expected space used is $O(n)$
  - The expected search, insertion and deletion time is $O(\log n)$
- Skip list are fast and simple to implement in practice.

Summary

Summary

- A skip list is a data structure for dictionaries that uses a randomized insertion algorithm.
- In a skip list with n entries:
  - The expected space used is $O(n)$
  - The expected search, insertion and deletion time is $O(\log n)$
- Skip list are fast and simple to implement in practice.

Thanks

Questions?