Bertrand Simon

part of a joint work with:
Bender, Berry, Johnson, Kroeger, McCauley, Phillips, Singh, Zage

ENS Lyon

Jan. 2018
Cache-efficient skip lists

Bertrand Simon

part of a joint work with:
Bender, Berry, Johnson, Kroeger, McCauley, Phillips, Singh, Zage

ENS Lyon

Jan. 2018
Outline

1. Skip lists
2. External Memory
3. External-memory skip list
The problem we want to solve

Dictionary problem on $\mathbb{N}$

- Insert $i$
- Delete $i$
- Search $i$
- Range Query ($i,k$ elements)

Example

Insert 26; Insert 8; Insert 4; Insert 17; Insert 42; Insert 1664; Delete 4; Search 26; Delete 26; Insert 58; Insert 2; Search 26; $RQ(8, 4) \rightarrow [8; 17; 42; 58]$;

Performance we seek ($n$ elements in the set)

- Insert, Delete, Search:
- Range Query:
The problem we want to solve

**Dictionary problem on** $\mathbb{N}$

- Insert $i$
- Delete $i$
- Search $i$
- Range Query ($i, k$ elements)

**Example**

- Insert 26; Insert 8; Insert 4;
- Insert 17; Insert 42; Insert 1664;
- Delete 4; Search 26; Delete 26;
- Insert 58; Insert 2; Search 26;
- $RQ(8, 4) \rightarrow [8; 17; 42; 58]$;

**Performance we seek** ($n$ elements in the set)

- Insert, Delete, Search: $O(\log n)$
- Range Query: $O(k + \log n)$
The problem we want to solve

**Dictionary problem on** \( \mathbb{N} \)

- Insert \( i \)
- Delete \( i \)
- Search \( i \)
- Range Query (\( i, k \) elements)

**Example**

- Insert 26; Insert 8; Insert 4;
- Insert 17; Insert 42; Insert 1664;
- Delete 4; Search 26; Delete 26;
- Insert 58; Insert 2; Search 26;
- \( RQ(8, 4) \rightarrow [8; 17; 42; 58] \);

**Performance we seek** (\( n \) elements in the set)

- Insert, Delete, Search: \( \mathcal{O}(\log n) \)
- Range Query: \( \mathcal{O}(k + \log n) \)

**Famous data structures solve this**

- Self-balancing binary search trees (AVL, Red-black tree...)

---

**Skip lists External Memory External-memory skip list**

---

**Bertrand Simon**

**Cache-efficient skip lists**

---

**4 / 20**
What’s the use of skip lists?

Red-black trees also solve this problem but...

- Red-Black tree invented in 1972 [Bayer]
- Who can implement right now a red-black tree?

“Skip lists are simpler, faster and use less space” – W. Pugh, 1989.
What’s the use of skip lists?

Red-black trees also solve this problem but...

- Red-Black tree invented in 1972 [Bayer]
- Who can implement right now a red-black tree?

"Skip lists are simpler, faster and use less space"

What’s the use of skip lists?

Red-black trees also solve this problem but...

- Red-Black tree invented in 1972 [Bayer]
- Who can implement right now a red-black tree?

“Skip lists are simpler, faster and use less space”


Advantage: history independence

- Reveals nothing on the past: deletes, searches, order of operations...
What’s the use of skip lists?

Red-black trees also solve this problem but...

- Who can implement right now a red-black tree?

"Skip lists are simpler, faster and use less space"

Advantage: history independence

- Reveals nothing on the past: deletes, searches, order of operations...

More

- Easy concurrency
- fun, elegant, teaches probabilities...
From a simple list to skip lists

Properties

- Maintain a sorted list of the elements
- Support operations in $O(\log n)$ in expectation and with high probability ($\approx$ worst-case analysis)
From a simple list to skip lists

**Properties**

- Maintain a sorted list of the elements
- Support operations in $O(\log n)$ in expectation and with high probability ($\approx$ worst-case analysis)

**Definition of $O(\log n)$ with high probability**

- $\forall c$ large, with proba $1 - n^{-\Omega(c)}$, all operations cost $< c \log n$
- Ex: $n = 1000$, $1 - 10^{-9} < 3 \log n$
From a simple list to skip lists

Properties

- Maintain a sorted list of the elements
- Support operations in $O(\log n)$ in expectation and with high probability ($\approx$ worst-case analysis)

Definition of $O(\log n)$ with high probability

- $\forall c$ large, with proba $1 - n^{-\Omega(c)}$, all operations cost $< c \log n$
- Ex: $n = 1000$, $1 - 10^{-9} < 3 \log n$

Description of ideal skip lists without updates

On the board
Searching in \( \lg n \) linked lists

**Example:** \texttt{Search(72)}
Updating a skip list

Updating ideal skip lists: expensive
Now rely on probabilities...
Updating a skip list

Updating ideal skip lists: expensive

Now rely on probabilities...

Delete $i$

- Search $i$, delete $i$ from all lists
Updating a skip list

Updating ideal skip lists: expensive

Now rely on probabilities…

Delete $i$

- Search $i$, delete $i$ from all lists

Insert $i$

- Search $i$, insert $i$ at the bottom list
- Toss a coin: Head $\rightarrow$ Return() — Tail $\rightarrow$ insert $i$ one level higher
Updating a skip list

Updating ideal skip lists: expensive

Now rely on probabilities...

Delete $i$

- Search $i$, delete $i$ from all lists

Insert $i$

- Search $i$, insert $i$ at the bottom list
- Toss a coin: Head $\rightarrow$ Return() $\rightarrow$ Tail $\rightarrow$ insert $i$ one level higher
- Toss a coin: Head $\rightarrow$ Return() $\rightarrow$ Tail $\rightarrow$ insert $i$ one level higher
Updating a skip list

Updating ideal skip lists: \textit{expensive}

Now rely on probabilities...

\textbf{Delete } \textit{i}

- Search \textit{i}, delete \textit{i} from all lists

\textbf{Insert } \textit{i}

- Search \textit{i}, insert \textit{i} at the bottom list
- Toss a coin: Head $\rightarrow$ Return() $\rightarrow$ Tail $\rightarrow$ insert \textit{i} one level higher
- Toss a coin: Head $\rightarrow$ Return() $\rightarrow$ Tail $\rightarrow$ insert \textit{i} one level higher
- Toss a coin: Head $\rightarrow$ Return() $\rightarrow$ Tail $\rightarrow$ insert \textit{i} one level higher
- ...
Updating a skip list

Updating ideal skip lists: expensive

Now rely on probabilities...

Delete $i$
  ▶ Search $i$, delete $i$ from all lists

Insert $i$
  ▶ Search $i$, insert $i$ at the bottom list
  ▶ Toss a coin: Head $\rightarrow$ Return() — Tail $\rightarrow$ insert $i$ one level higher
  ▶ Toss a coin: Head $\rightarrow$ Return() — Tail $\rightarrow$ insert $i$ one level higher
  ▶ Toss a coin: Head $\rightarrow$ Return() — Tail $\rightarrow$ insert $i$ one level higher
  ▶ ... 

Do you see something missing?
Some probabilities

**Theorem**

_A skip list has $O(\log n)$ levels whp._

**Proof.**

\[
P(> c \log n \text{ levels}) \leq n \cdot P(\text{Insert gets } > c \log n \text{ promotions})
\]

\[
\leq n \cdot \left(\frac{1}{2}\right)^{c \log n}
\]

\[
\leq n^{1-c}
\]
Some probabilities

Theorem

A search costs $\mathcal{O}(\log n)$ whp.
Some probabilities

**Theorem**

A search costs $O(\log n)$ whp.

**Proof.**

Analyze it backwards (from bottom to top-left)

- if the node was promoted: go up (proba. 1/2)
- otherwise: go left (proba. 1/2)
- we stop after $< c \log n$ “up” moves

Whp, after how many moves do we stop?
Answer:
Some probabilities

Theorem

A search costs $\Theta(\log n)$ whp.

Lemma

To obtain $c \log n$ Heads, we need $\Theta(\log n)$ coin flips whp.

Proof.

Analyze it backwards (from bottom to top-left)

- if the node was promoted: go up (proba. 1/2)
- otherwise: go left (proba. 1/2)
- we stop after $< c \log n$ “up” moves

Whp, after how many moves do we stop?
Answer:
Some probabilities

**Theorem**

A search costs $O(\log n)$ whp.

**Lemma**

To obtain $c \log n$ Heads, we need $\Theta(\log n)$ coin flips whp.

**Proof.**

Analyze it backwards (from bottom to top-left)

- if the node was promoted: go up (proba. 1/2)
- otherwise: go left (proba. 1/2)
- we stop after $< c \log n$ “up” moves

Whp, after how many moves do we stop?
Answer: $\Theta(\log n)$
Outline

1. Skip lists
2. External Memory
3. External-memory skip list
Forget everything you know

*Classic RAM model used to evaluate algorithm*

- Memory access (read, write)
- Computation (compare, add, multiply...)

\[
\text{cost 1}
\]
Forget everything you know

*Classic RAM model used to evaluate algorithm*

- Memory access (read, write)
- Computation (compare, add, multiply...) \[ \text{cost } 1 \]

*Problem when dealing with large data*

fig/memory.jpg
A new model

Change of view

- *Classic* complexity (RAM model): focus on computations
- Disk-Access Model [Aggarwal’88]: focus on communications
A new model

Change of view

- *Classic* complexity (RAM model): focus on computations
- Disk-Access Model [Aggarwal’88]: focus on communications

Model

- Two layers of memory: a main RAM of size $M$ and an infinite disk
- Data needs to be on RAM to be processed
- Can exchange contiguous blocks of size $B$ for 1 I/O
A new model

Change of view

- *Classic* complexity (RAM model): focus on computations
- Disk-Access Model [Aggarwal’88]: focus on communications

Model

- Two layers of memory: a main RAM of size $M$ and an infinite disk
- Data needs to be on RAM to be processed
- Can exchange contiguous blocks of size $B$ for 1 I/O
- Complexity of an algorithm: worst-case I/O number
Why are I/Os so important?

Large data: classic algorithms access frequently to disk

**Access time**

- RAM: 100 ns
- Disk: 10 ms = 10 000 000 ns
Why are I/Os so important?

Large data: classic algorithms access frequently to disk

**Access time**

- **RAM**: 100 ns
- **Disk**: 10 ms = 10 000 000 ns

**Analogy**: \[
\frac{\text{Ram speed}}{\text{Disk speed}} \approx \frac{\text{escape velocity from Earth}}{\text{speed of a turtle}}
\]

**DAM model**: totally forget computations
New bounds

Classic bounds

<table>
<thead>
<tr>
<th></th>
<th>RAM</th>
<th>DAM (I/Os)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scan</td>
<td>$N$</td>
<td></td>
</tr>
<tr>
<td>Search</td>
<td>$\log N$</td>
<td>$N \log N$</td>
</tr>
<tr>
<td>Merge-Sort</td>
<td>$N \log N$</td>
<td></td>
</tr>
</tbody>
</table>
### New bounds

#### Classic bounds

<table>
<thead>
<tr>
<th></th>
<th>RAM</th>
<th>DAM (I/Os)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scan</td>
<td>$N$</td>
<td>$\frac{N}{B}$</td>
</tr>
<tr>
<td>Search</td>
<td>$\log N$</td>
<td></td>
</tr>
<tr>
<td>Merge-Sort</td>
<td>$N \log N$</td>
<td></td>
</tr>
</tbody>
</table>
New bounds

Classic bounds

<table>
<thead>
<tr>
<th></th>
<th>RAM</th>
<th>DAM (I/Os)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scan</td>
<td>$N$</td>
<td>$\frac{N}{B}$</td>
</tr>
<tr>
<td>Search</td>
<td>$\log N$</td>
<td>$\log_B N$</td>
</tr>
<tr>
<td>Merge-Sort</td>
<td>$N \log N$</td>
<td></td>
</tr>
</tbody>
</table>

External memory Search tree: B-tree
# New bounds

## Classic bounds

<table>
<thead>
<tr>
<th>Operation</th>
<th>RAM</th>
<th>DAM (I/Os)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scan</td>
<td>$N$</td>
<td>$\frac{N}{B}$</td>
</tr>
<tr>
<td>Search</td>
<td>$\log N$</td>
<td>$\log_B N$</td>
</tr>
<tr>
<td>Merge-Sort</td>
<td>$N \log N$</td>
<td>$\frac{N}{B} \log_{M/B} \frac{N}{B}$</td>
</tr>
</tbody>
</table>

## External memory Search tree: B-tree
Outline

1. Skip lists

2. External Memory

3. External-memory skip list
Skip lists and external memory

Why it does not work straight away

- RAM Insert: any memory slot
- Each operation requires $\Theta(\log N)$ I/Os
Skip lists and external memory

Why it does not work straight away

- RAM Insert: any memory slot
- Each operation requires $\Theta(\log N)$ I/Os
- We want the same as B-tree:
  $O(\log_B N)$ I/Os — RQ: $O(\log_B N + k/B)$ I/Os

Any idea to improve locality? (& keep history-independence)
Skip lists and external memory

Why it does not work straight away

- RAM Insert: any memory slot
- Each operation requires $\Theta(\log N)$ I/Os
- We want the same as B-tree:
  \[ O(\log_B N) \text{ I/Os} \quad \text{—} \quad \text{RQ: } O(\log_B N + k/B) \text{ I/Os} \]

Any idea to improve locality? (& keep history-independence)

- Block together elements between 2 promoted ones
- Change the promotion probability
What should be the promotion probability?

If \( p > 1/B \)

- Range queries are not efficient

\[ \text{If } p > 1/B \]

\[ \text{Range queries are not efficient} \]
What should be the promotion probability?

If \( p > 1/B \)
- Range queries are not efficient

If \( p < 1/B \)
- Searches have to span several blocks
What should be the promotion probability?

**If** $p > \frac{1}{B}$
- Range queries are not efficient

**If** $p < \frac{1}{B}$
- Searches have to span several blocks

**If** $p = \frac{1}{B}$  [Golovin’2010]
- OK on average
What should be the promotion probability?

**If** $p > 1/B$
- Range queries are not efficient

**If** $p < 1/B$
- Searches have to span several blocks

**If** $p = 1/B$ [Golovin’2010]
- OK on average
- Whp: $\sqrt{N}$ series of $B \log N$ non-promoted elements
- For $> \sqrt{N}$ elements, a search costs $\Omega(\log N)$ I/Os
Towards our skip list

Promotion probability

▶ \( \frac{\log B}{B} < p < B^{-0.5} \) (ex: \( p = B^{-0.7} \)) \( \rightarrow \) searches OK on average
▶ largest series: \( < B \log_B N \) whp \( \rightarrow \) \( O(\log_B N) \) I/Os for searches

Blocking strategy

▶ Block between doubly-promoted elements \( \rightarrow \) Range Queries
▶ Reserve buffers between promoted elements \( \rightarrow \) Updates

More

▶ Some tricks to ensure all bounds whp & history independence
Example of our skip list for $B = 3$ and $p = 1/2$