The analysis of self-adjusting data-structures

Results,

Conjectures,

Alternatives

John Iacono

Polytechnic University

Brooklyn

NYC

NY

USA
Outline

- Trees
- Heaps
Binary Search Trees - Prehistory

- AVL Trees
- "Optimal" trees
- Level-linked trees
- Finger trees
AVL Trees

- \( O(\log n) \) Ins/Del/Search
- One bit per node of structural info
- Worst-case asymptotically optimal
- Many alternatives (e.g., left-leaning red-black trees) [Sedgewick 08]
"Optimal trees" [e.g., Knuth 85]

- Given n items each with a weight $w_i$, $\sum w_i = 1$, in an optimal tree the cost to search for item $i$ is $O(\log \frac{1}{w_i})$

- Optimal if searches are drawn independently at random with probability equal to the weights

- Entropy
Optimal Trees

Simple Construction
Optimal Trees

Simple Construction
Optimal Trees

Simple Construction

Diagram of a tree with nodes labeled A through H.
Optimal Trees

Simple construction
Level-linked Trees

- Idea: Searches should be fast if the key being searched is "close" to the previous search.
Level-Linked Trees

- Idea: Searches should be fast if the key being searched is "close" to the previous search.

- Search takes time $O(\log K)$

  $K =$ number of keys from last search to current search.
Level-linked Trees
Level-linked Trees
Level-linked Trees

Add more pointers
Level-linked Trees

Add more painters
Finger Trees

Idea:

1 3 10 12 17 31 54 63 64 65 77 81
Finger Trees

Idea:

1 3 10 12 17 31 54 63 64 65 77 81
Finger Trees

Idea:

$O(\log n)$

1 3 10 12 17 31 54 63 64 65 77 81
Enough Prehistory

Splay Trees

[{}]

Sleator & Tarjan 85
Splay Trees

- No balance information stored
- No structural constraints
- Runtimes are amortized

IDEA: Bring the item you search to the root
Move-to-Root: First Idea
Move-to-Root: First Idea
Move-to-Root: First Idea
Move-to-Root: First Idea
Move-to-Root: First Idea
Move-to-Root: First Idea
Move to Root: First Idea
Move-to-Root: First Idea
Move-to-Root: First Idea
Move-to-Root: First Idea

Problem: $O(n)$ runtime
Move-to-Root: Second Idea
Move-to-Root: Second Idea
Move-to-Root: Second Idea
Move-to-Root: Second Idea
Move-to-Root: Second Idea
Soplay Tree Rules
Splay Trees

- EZ to understand
  how they work

- Hard to understand
  why they work
Amortized Analysis - Potential

- Potential of a node is \log \text{of subtree size}
Amortized Analysis - Potential

- Potential of a node is log of subtree size

- To splay a node at depth $h$ costs $h$

- Potential change is $O(\log n) - h$

- Amortized cost = $O(h \log n)$
The Big Conjecture

Splay Trees Are The BEST BST

(within constant factors)
The Big Conjecture

Splay Trees Are The BEST BST

(within constant factors)

What is a BST?
What is a BST? [Wilber 89]

- You know the structure
- Unit cost operations, one pointer
  - Go Left/Right/Up
  - Rotate Left/Right
- To search, bring the pointer to the item being searched.
What is an optimal BST

Given a search sequence

\[ X = x_1, x_2, \ldots, x_m \]

\[ \text{OPT}(X) = \text{Fastest any BST can run } X \]
What is an optimal BST?

Given a search sequence

\[ X = x_1, x_2, \ldots, x_m \]

\( \text{OPT}(X) = \text{Fastest any BST can run } X \)

Can be computed in time

\[ O(5^{\log_2 n}) \]
BST Model - Another View
Dynamic Optimality

Conjecture

$\text{SPLAY}(x) = O(\text{DET}(x))$
Dynamic Optimality

Conjecture

$SPLAY(x) = O(OET(x))$

Online

Offline
BST Model

- Crucial. With a more powerful model, the comparison with the best offline algorithm is not so interesting, if space is unlimited.
Open Problem

- Come up with any result on dynamic optimality in the RAM model.
Open Problem

- Come up with any result on dynamic optimality in the ram model.
- E.g. A structure

\[ \forall U \in RDS(x) = o\left( \sqrt{\frac{\log n}{\log \log n}} \cdot \text{RAMOPT}(x) \right) \]
So we can't prove it. What next...

- Find some specific things that trees can do and prove/conj that splay trees can do it too.
Splay trees ∈ Prehistory

- As good as “optimal” trees
  though we don’t know the
  probabilities

- As good as static finger trees
  though we don’t know the finger

[5+1]
Splay trees & Prehistory

- As good as "optimal" trees
  though we don't know the
  probabilities

- As good as static finger trees
  though we don't know the
  finger

Amazing
Side note

- Having the same runtime as "optimal" trees is not so hard even if you don't know the probabilities

- Just rebuild every n time with the observed probabilities
Splay Trees - Working Set

1 2 1 3 4 7 3 1 2
Splay Trees - Working Set
Splay Trees — Working Set

1 2 13 4 7 3 1 2

$O(\log 4)$

- This implies everything on previous page
Splay Trees — Working Set

$E_{+T}$

1 2 1 3 4 2 3 1 2

$O(\log 4)$

- This implies everything on previous page
- Not amazing

All these just plan with weights
Splay Trees Can Do It!

- Dynamic Finger
  - Bitter hard proof
- Sequential Access
  - Combinatorially

- Trees can be dequeues
  - Still no proof
  - Close \( O(\alpha^*(n)) \) [pettie 08]
Are the splay rules special?
Are the splay rules special?
Are the splay rules special?
So What Is Special?

Depth of node after splay
= Before splay
  - $c / \text{Intersection with splay path}$
  + $O(1)$
So What Is Special?

Depth of node after splay

= Before splay

− \lfloor \text{intersection with splay path} \rfloor < 1

+ O(1)

Would be nice to see a proof that used this directly
Other Conjectured Variants

- Turn search path into complete balanced tree

[Diagram of search path transformation]
Other Conjectured Variants

- Turn search path into complete balanced tree

Conj: Not \( O(\log n) \) amortized
Other Conjectured Variants

- Idea: Splay trees are conjectured to be $O(COPT(x))$ and online.

- Is there a poly-time $O(COPT(x))$ BST?

- If you knew the future who would you put closest to the root [comn Munro]
IAN

Next
\[ \text{Conv} : \text{TAN}(x) = O(\text{OET}(x)) \]
- Conj: $\text{IAN}(x) = o(\text{Opt}(x))$

- Conj: IAN has $O(\log n)$ amortized time per search.
If the key values were to be randomly permuted (maintaining equality) then $\text{WORKING-SET}(x) = O(\text{OPT}(x))$. 
If the key values were to be randomly permuted (maintaining equality), then $\text{WORKING-SET}(x) = O(\text{OPT}(x))$.

If rotations are free, and you have lots of time to think, there is an online dynamic structure. [Olum, Chawla, Kalai 02]
So what next?

- The proofs are getting hard
  so let's try to invent some
  things that are simpler
So what next?

- The proofs are getting hard so let's try to invent some things that are simpler

- Unified Trees: A combination

[IoLi Budion, Calé, Vomaine, 07]
So what next?

- The proofs are getting hard so let's try to invent some things that are simpler.

- Unified Trees: A combination of spatial and temporal locality.
So what next?

- What about a competitive ratio result?

\[ M_{\text{DS}}(x) = c \cdot \text{OPT}(x) \]

\[ c = \Omega(\log n) \text{ trivial} \]

\[ c = O(1) \text{ dynamic} \]
So what next?

- What about a competitive ratio result?

\[ \text{MYDS}(x) = c \cdot \text{OPT}(x) \]

\[ c = \Theta(\log n) \quad \text{trivial} \]

\[ c = \Theta(1) \quad \text{dynamic} \quad \text{[Demaine, Harvey, I, Perreson 07]} \]

\[ TANGO(x) = \Theta(\log \log n) \cdot \text{OPT}(x) \]
Competitive Trees

- Competitive ratio stuff needs a decent lower bound

\( \text{query}(X) \geq \text{sometunnel}(X) \)

- We are lucky, more than luckily, there are two:

  \([\text{Wilber 89}]\)
Defaut

- Thinking about trees, rotations, etc., hurts my head

- There must be a better way
$X = 1, 4, 5, 7, 6, 2, 3$
$X = \{1, 4, 5, 7, 6, 2, 3\}$

$X \times X$
$X = 1, 4, 5, 7, 6, 2, 3$
\[ X = 1, 4, 5, 7, 6, 2, 3 \]
\[X = 1, 4, 5, 7, 6, 2, 3\]
\[ X = 1, 4, 5, 7, 6, 2, 3 \]
X = 1, 4, 5, 7, 6, 2, 3

Black X
= Item Searched

Red X
= Item touched

Total # of X's
= Total cost of all searches

Key Value

Time

{ x x x
   x x x
   x x x
   x x x
   x x x
   x x x
   x x x
   x x x
}
$X = 1, 4, 5, 7, 6, 2, 3$

Can you find this

Key Value

Empty

X
\[ X = 1, 4, 5, 7, 6, 2, 3 \]

Can you find this?

Key Value

Empty

No!
Box Model

- No empty boxes = "Orthogonally satisfied"
Box Model

- No empty boxes = "Orthogonally satisfied"

- All BST \rightarrow orthogonally satisfied

- Orthogonally satisfied \rightarrow BST
Minimal OS Superset = DynOPT
One Idea: Greedy
One Idea: Greedy
One Idea: Greedy
One Idea: Greedy
One Idea: Greedy
One Idea: Greedy
One Idea: Greedy
One Idea: Greedy
One Idea: Greedy
One Idea: Greedy

Observation
This algorithm is JAN
One Idea: Greedy

\[ \text{Observation} \]
This algorithm is IAN

\[ \text{Observation 2} \]
We can make an online PST with runtime \( \Theta(\text{IAN}(x)) \)
Where were we?

- Lower bounds
  - W₁
  - W₁Ⅰ
$WI(x) = \# \text{ of circles}$

$\text{Oct}(x) = \sum WI(x)$
$\text{WI}(x) = \# \text{ of circles}$

$\text{OCT}(x) = \Omega(\text{WI}(x))$

$\text{TANGO}(x) = O \left( \log \log n \cdot \text{WI}(x) \right)$
\[ \text{WII} \]

\[ \text{WII}(X) = \text{pt of circles} \]

\[ \text{OPT}(X) = \bigcap \{ \text{WII}(X) \} \]
$WII(X) = \forall$ of circles

$OPT(X) = \exists (WII(X))$

Don't know which is better, WI or WII.

Conj: WII is better and is tight
NISIAN [Hormans thesis]
NISLAN
NIS LAN
NISLAIN
NISLAN
NISIAN
NISIAN
NISIAN

Not 05
NISIAN

\[ \Omega(w_1(x)) = NISIAN(x) \leq \Omega(\text{OPT}(x)) \]

Not OS

But it is a LB!!

And no worse than

\[ w_1, w_2 \]
Conclusion - Trees

A constant-factor minimal OS superset.

Doesn't seem so hard.
Conclusion - Trees

A constant-factor minimal OS superset doesn't seem so hard

(Exact is NPC for a variant)
Heaps

- Prehistory

- Pairing Heaps

- Skew Heaps

- Connections, alternatives
Prehistarw: Fibonacci Heaps

Fib-Heap

- Extract-Min: $O(\log n)$
- Insert: $O(1)$
- Decrease-Key: $O(1)$
- Meld: $O(1)$

[Frederickson &Thorup 87]
Prehistory: Fibonacci Heaps

<table>
<thead>
<tr>
<th>Operation</th>
<th>Fib-Heap</th>
<th>Best Possible</th>
</tr>
</thead>
<tbody>
<tr>
<td>Extract-Min</td>
<td>$O(\log n)$</td>
<td>Best Possible, so lets invent something else that can't be better and we have no clue how fast they run.</td>
</tr>
<tr>
<td>Insert</td>
<td>$O(1)$</td>
<td></td>
</tr>
<tr>
<td>Decrease-Key</td>
<td>$O(1)$</td>
<td></td>
</tr>
<tr>
<td>Meld</td>
<td>$O(1)$</td>
<td></td>
</tr>
</tbody>
</table>
Prehistam: Fibonacci Heaps

**Fib-Heap**

- **Extract-Min**: $O(\log n)$
- **Insert**: $O(1)$
- **Decrease-Key**: $O(1)$
- **Meld**: $O(1)$

Best possible, so let's invent something else that can't be better and we have no clue how fast they run.

Fib Heaps are
- complicated
- have $\log \log n$ balance bits per node
Pairing Heaps

Primitive: Pair

Fredman
Sedgewick
Sleator
Tarjan
86
Pairing Heap

Insert \((x)\)

\[ X + \triangle \]

\[ M(x) \]

\[ X + Y \]

Decrease Key \((x)\)

\[ \text{Reduction} \]

\[ X + \]
Pairing Heap - Exvarc Min
Pairing Heap - Exact - Min
Pairing Heap - Exact - Min
Pairing Heap - Exact Min
Pairing Heap - Exact - Min
Pairing Heap - Exact-Min

e.g.
Tilt your head and it looks like a splosh.
Tilt your head and it looks like a sploq
Runtime ???
<table>
<thead>
<tr>
<th>Operation</th>
<th>Fib</th>
<th>Pair</th>
</tr>
</thead>
<tbody>
<tr>
<td>Extract-Min</td>
<td>$O(\log n)$</td>
<td>$O(\log n)$</td>
</tr>
<tr>
<td>Insert</td>
<td>$O(1)$</td>
<td>$O(\log n)$</td>
</tr>
<tr>
<td>Decrease-Key</td>
<td>$O(1)$</td>
<td>$O(\log n)$</td>
</tr>
<tr>
<td>Meld</td>
<td>$O(1)$</td>
<td>$O(\log n)$</td>
</tr>
<tr>
<td>Operation</td>
<td>Fib</td>
<td>Pair</td>
</tr>
<tr>
<td>------------------</td>
<td>------------</td>
<td>-------------</td>
</tr>
<tr>
<td>Extract - Min</td>
<td>$O(\log n)$</td>
<td>$O(\log n)$</td>
</tr>
<tr>
<td>Insert</td>
<td>$O(1)$</td>
<td>$O(\log n)$</td>
</tr>
<tr>
<td>Decrease - Key</td>
<td>$O(1)$</td>
<td>$O(\log n)$</td>
</tr>
<tr>
<td>Meld</td>
<td>$O(1)$</td>
<td>$O(\log n)$</td>
</tr>
<tr>
<td>Operation</td>
<td>Fib</td>
<td></td>
</tr>
<tr>
<td>-----------</td>
<td>-----</td>
<td></td>
</tr>
<tr>
<td>Extract-Min</td>
<td>$O(1)$</td>
<td></td>
</tr>
<tr>
<td>Insert</td>
<td>$O(1)$</td>
<td></td>
</tr>
<tr>
<td>Decrease-Key</td>
<td>$O(1)$</td>
<td></td>
</tr>
<tr>
<td>Meld</td>
<td>$O(1)$</td>
<td></td>
</tr>
<tr>
<td>Pair</td>
<td>$O(n)$</td>
<td></td>
</tr>
<tr>
<td>Pair</td>
<td>$O(n)$</td>
<td></td>
</tr>
</tbody>
</table>

+ "Empirical Evidence" of $O(1)$ D-L
[Stasko, Vitter 87]
Lower Bounds [Fredman 99]

Model of computation:

Generalized from pairing and Fib heaps

Forest of heaps. Combine based on comparisons on the roots and looking at any additional bits attached to the nodes
Lower Bounds

Result:

If Insert is $O(\log \log n)$ and Extract-min is $O(\log n)$ and Decrease-key is $c \log n$, $c < 1$ then the number of augmented bits is $\geq \log \log n$
Lower Bounds

Result:

If Insert is $O(\log n)$ and Extract-min is $O(\log n)$ and Decrease-key is $c \log n, c < 1$ then the number of augmented bits is $\geq \log \log n$.

Pairing heaps have $O$ augmented bits so Decrease-key is $\geq \log n$. 
Open Problems

- What is the real runtime of pairing heaps?
- Is there some self-adjusting structure with the same runtime as Fib heaps? Need to step outside of the model.
Beyond The Worst Case

- Pairing heaps have a working-set property \([\text{[Ion]}]\)

- Conj. Excluding Decrease-Key
  Dynamic Optimality = Working Set in the Heap model

- Queues - not in the model
  \([E. Langerman 05]\)
Variants

How to combine the children of the old root?

Almost anything works except one incremental pass

[[My qualifying exam question from Friedman]]
Variants

(0 + 0) + (0 + 0) + (0 + 0) + (0 - 0)
Variants

\[(0 + \varnothing) + (0 + \varnothing) + (0 + \varnothing) + (0 + \varnothing)\]

\[\left( (0 + \varnothing) + (0 + \varnothing) \right) + \left( (0 + \varnothing) + (0 + \varnothing) \right)\]
Variants

\[(0 + 0) + (0 + 0) + (0 + 0) + (0 + 0)\]

\[\left( (0 + 0) + (0 + 0) \right) + \left( (0 + 0) + (0 + 0) \right) \]
Variants

\[(0 \cdot 0) + (0 \cdot 0) + (0 \cdot 0) + (0 \cdot 0) \]

\[\left( (0 + 0) + (0 + 0) \right) + \left( (0 + 0) + (0 + 0) \right) \]

\[\left( (0 + 0) + (0 + 0) \right) + \left( (0 + 0) + (0 + 0) \right) \]
Skew Heaps [Sleator + Tarjan 86]

- Another self-adjusting heap
- Binary
- Great for teaching amortized analysis.
Skew Heaps - Meld
Skew Heaps - Meld
Skew Heaps - Meld
Skew Heaps - Meld
Analysis

Ins: Extract - min, meld
are $O(\log n)$

Put a coin on any node
that has a larger right
subtree than left

""" All but $\log n$ on a right
path have coins. Flipping to
a left path increases coins by
at most $\log n - \text{path length}$
Analysis

- Not much work has been done on skew heaps.
Analysis

- Not much work has been done on skew heaps.

- Skew heaps are not so good...
My lazy friend

Skew Heap

Lazy friend

1

3

1

3
My lazy friend

Skew Heap

Lazy friend

\[ 1 \quad + \quad 2 \]

\[ 3 \]

Insert 2
My lazy friend

Skew Heap

Lazy friend

Insert ②
My lazy friend

Skew Heap

1
2
3

Lazy friend

1

1.2 1.3

Find-min
My lazy friend

Skew Heap

Lazy friend

Find-min
Lazy Friend

- A heap gives its lazy friend all of the comparisons it does.
- The lazy friend, when it has a forest and needs one tree, does the comparisons in the list among its roots.
Lazy Friend

The number of comparisons performed by a lazy friend is at most the number of the original heap.

(total time may be different as how did our lazy friend find the right comparison)
Skew Heap < Pairing Heap

- The lazy friend of the skew heap is the odd-even pairing heap ([Fredman 99])
Skew Heap ≤ Pairing Heap

1. The lazy friend of the skew heap is the odd-even pairing heap.

2. On any sequence of operations, the skew heap is the same or slower than the pairing heap.
The End