Splay Trees Recap

• Dynamic BST
  - re-arranges tree even after search (via pointer moves and rotations)

• \(\mathcal{O}(\log n)\) amortized cost per operation

• In some cases stronger guarantees (e.g., if search sequence \(R\) structured, biased distribution, locality, ...)

• Conjectured competitiveness ("almost" optimal on every search sequence \(R\))

  - best "offline" dynamic BST execution of \(R\)
  - (knownly \(R\) in advance, optimized for \(R\))

  → Dynamic Optimality Conjecture

  Some partial results, but not even \(\mathcal{O}(\log n)\)-competitiveness is known.

(\(\mathcal{O}(1)\)-comp. conjectured)

Easy claim: Splay is \(\mathcal{O}(\log n)\)-competitive

Proof.
  \(R = r_1, \ldots, r_m\)

  \(m \geq n \quad \text{(}m\ \text{is \# of distinct keys)}\)

  Cost of splay on \(R \leq m \times \mathcal{O}(\log n) \quad \text{(Thm)}\)

  \(\text{OPT}(R) \geq n \quad \text{(each search can at least 1 visit the root)}\)

  \(\Rightarrow \frac{\text{Splay}(R)}{\text{OPT}(R)} \leq \mathcal{O}(\log n)\)

Geometry of Binary Search Trees

Search ref. \(R = r_1, \ldots, r_m\)

  e.g. \(R = 5, 1, 4, 3\) \(\quad \text{\((m = 5)\)}\)

  \(n = 5\)

Represent \(R\) as set of points \((r_i, i)\) for \(i = 1, \ldots, m\)

e.g.,

```
   5 ---->
  /     |
/       |
/        |
  3 ---->
 /     |
/       |
2 ---->
/     |
/       |
1 ---->
  
```

"input points"

```
1 2 3 4 5
A B C D E
```

key-values
Serve R in a BST,
(Splay tree or any other BST
based on pointer moves and
rotations)

\[ R = 2, 5, 1, 4, 3 \quad (m = n = 5) \]

\[ \text{Cost: } \# \text{ rotations} + \# \text{ pointer moves} \]

\[ \text{initial tree: } \]

- search(2)
- search(5)
- search(4)
- search(1)
- search(3)

*Visited to rotate*

*re-arrange by rotation*

(touched: 4, 3, 2)
(touched: 5, 1, 2)
(touched: 1, 3, 4, 5)
(touched: 5, 4, 3)

Plot the BST sequence as a set of points (that contains the input points)

- True
- Input points obtained from R
- Touched points (other than input points)

- Nodes touched in the tree at each time

Claim: Let \( T_1, T_2 \) be two BSTs of size \( k \)
over same set of nodes.

Then we can transform \( T_1 \rightarrow T_2 \) with \( O(k) \) rotations.

(\text{was exercise})

[This is true even if we allow rotations only at
pointer and also count pointer moves]

(exercise: check this)

In BST execution we only
need to do \( O(k) \) rotations +
pointer moves

where \( k \) is the \# touched nodes

Cost of BST execution of \( R \)

\[ *= \# \text{ points in geom.view} \]

(exercise: check this)
Given a point set $P$, find a superset of $P$ that describes a valid BST execution and $R$ as small as possible (= small cost).

**Theorem:** A point set $S$ describes a "valid execution" of $P$ by a BST if:

1. $P \subseteq S$, and
2. $S$ has no "empty rectangles" ($S$ is "satisfied").

For all $x, y \in S$:
- rectangle with corners $x, y$ contains some point from $S \setminus \{x, y\}$ (possibly on boundary or other corner)

**E.g.**

- ![not empty](image)
- ![not empty](image)
- ![not empty](image)

(observe that there must be some point on a rectangle side adjacent to $x$ and to $y$, as we can apply condition repeatedly)

$|S|$ = cost of BST execution

(BST problem is reduced to finding a small satisfied superset of points)
Given P, find smallest S \supseteq P with no empty rectangles

**Note**
In general, no efficient algorithm is known to solve this problem.
(It is also not known to be NP-hard)

\[ \bullet \; \text{vs.} \; \bullet \]
\[ \bullet \; \text{vs.} \; \bullet \]
\[ \bullet \; \text{vs.} \; \bullet \]

---

**Proof of Theorem**

1) \( S \) valid BST execution

\[ S \text{ has no empty rectangles (S is satisfied)} \]

Consider any two points \( a, b \in S \)

\[ (x_1, t_1), (y_1, t_2) \]

w.l.o.g. \( x < y \)

\[ t_1 < t_2 \]
BST touched \( x \) at time \( t_1 \). Want to prove touched \( y \) at time \( t_2 \).

1) \( \exists \) there is a point \((z, t_1) \in S\) s.t. \( z \in [x, y]\), then DONE.

2) ow. Let \( x' > x \) be the smallest value s.t.

\((x', t_1) \in S\)

(i) no such point, set \( x' = x + 1 \)
(we know \( x' > y \), as otherwise we are in case 1)

\( \Rightarrow \) cannot happen: \( x \) and \( x' \) were touched but \( \text{lca}(x, x') \) not
(if \( x < \text{lca}(x, x') < x' \))
(we assume nobody between \( x, x' \) is touched)

So it must be that one of \( x, x' \) is ancestor of the other.

as \( y \) was not touched, and \( x < y < x' \), \( y \) must be in a subtree hanging below \( x \)

\( \Rightarrow \) after time \( t_1 \), \( y \) is descendant of \( x \)

If \( x \) not touched in time interval \([t_1 + 1, t_2]\), then \( y \) remains descendant of \( x \):

\( \Rightarrow y \) cannot be touched at time \( t_2 \).

\( \Rightarrow \) \( y \) not empty, \( x \) must have been touched in this time interval, so rectangle not empty
S describes a valid BST execution

\[ \uparrow \]

S has no empty rectangles

We need to show how to recover a BST execution from a point set S. (may be not unique)

Build tree \( T_0 \): treetop of nodes, whose priority is given by earliest touched time (with key values 1-7 m) of each key value. (breaking tie arbitrarily)

![Diagram of tree construction]

Define \( T_i \): BST after \( i \)-th search.

Invariant: \( T_i \) is a treetop according to earliest touched time of keys in time interval \([i+1, \ldots, m] \)

Operation at time \( i \):

\( H_i \rightarrow H_i' \) s.t. it is a treetop according to earliest touch time in future

\( T_i \) is \( T_{i-1} \), with the top part \( H_i \) transformed into treetop \( H_i' \)

(rst of tree unchanged)

(need to prove that invariant is maintained)

From this it follows that the set of touched nodes \( H_i \) is indeed at the top of the tree.

Since these are the "earliest touched", they must have smallest priority, so they are on top by treetop property.
Proof that invariant is maintained:

Reminder:

-Invariant: $T_i$ is a treap acc. to earliest future touch times.

Induction: true for $T_0$, ..., $T_{i-1}$.

Suppose $T_i$ not a treap (some edge $x,y$ must witness this)

$x$ is above $y$, even though

$y$ will be touched earlier than $x$

- $xy$ cannot be both in $H_i$! (we just made it a correct treap)
- $xy$ cannot be both in $T_i \setminus H_i$!

($T_{i-1}$ was a correct treap and we didn't disturb this part)

Suppose some point $z$ touched at time $t$: $(x \leq z \leq y)$

$y$ cannot be child of $x$

Or $z$ above $(x,y)$ empty, since $y$ will be touched first.

But then both sides of rectangle empty $\Rightarrow S$ not satisfied.
Summary:
Find a satisfied superset of $P$ $\leftrightarrow$ Find a BST execution of $R$

($P$: geometric view of $R$)

BST execution $\rightarrow$ point set: plot touched elements at each time

point set $\rightarrow$ BST execution: at every time $i$, touched
   elements form the top part of current
   tree, transform into a tree
   with future touch time as the priority.
   (need to look at point set “globally”)

This is called “offline equivalence”

- Find satisfied superset
- No efficient algorithm is known

Two heuristics

1. Divide and Conquer
   - Split point set with vertical line in middle
   - Project each point to middle line
   - On no empty rectangles that cross middle line
   - Recurse on left/right side

Cost: size of point set

$T(n) = m + 2 \times T\left(\frac{n}{2}\right)$

$T(1) = 1$

$\rightarrow$ BST execution that serves $m$ searches in time $O(m \log n)$

BST with $O(\log n)$ cost per search
2) A different geometric algorithm:

- geometric sweepline algorithm
- move through the input with horiz. line, step-by-step
- odd new points only on the line
- invariant: below the line there are no empty rectangles

Claim. Alg. maintains invariant.
Show that no new empty rectangle is created at or below line $i$

at time $i$, current search point $(r_i, i)$ forms empty rectangle with points $a_1, \ldots, a_k, b_1, \ldots, b_k$.

$\Rightarrow$ project $a_1, \ldots, a_k$ to line $i$.

as $b_{k-1}, b_k$ increase by $y$-coordinate LTR

Suppose that some newly added point $X$ created an empty rect.
Other corner is $X'$
Split plane in quadrants 1, 2, 3, 4.
$X'$ cannot exist $\Rightarrow$ (argue case-by-case)
This is called the Greedy algorithm.

Claim: Nodes touched by Greedy at time $i$ (in row $i$)
are exactly the search path of the tree.

Proof. To prove correctly at earliest touch time

for $i=1$ claim is true

Suppose it is true for $i=1, \ldots, i-1$

at time $i$:

by induction hypothesis $x$ was on search path
at time $i'$,

so element in $E_{i', i}$ was on search path
at time $i'$

at time $i'$

$- x$ must have been a descendant of
$- x$ at time $i'$

since $x$ was not

touched in time interval
$(i', i)$, $i$ is still

descendant of $x$ when we search

$\Rightarrow x$ is on search path of $i_i$ at time $i$. 
Greedy corresponds to a BST algorithm called GreedyFuture:

- Search in BST

Transform search path into a treap, using earliest future search as priority.

\[ R = r_{1} \ldots r_{n} \]

**Diagram:**

Recall that Splay is an online algorithm.

GreedyFuture is an offline algorithm. We will see how to make GreedyFuture online.

**Question:** How good is GreedyFuture?

Conjectured \( O(\text{OPT}) \) (constant-competitive) \( \{ \text{OPEN} \} \)

Which is better: Splay/GreedyFuture?

GreedyFuture satisfies \( \text{Thm 1 - 6} \) (like Splay) and more.
Online BST algorithm in tree-view: serves search sequence $R = r_1, \ldots, r_m$ one-by-one, without knowing the future requests.

Online BST algorithm in geometric view: processes input points one-by-one (row-by-row) bottom-up, adds new points only in current row, without knowing input points above current row.

**Obs.** Greedy is an online algorithm.

GreedyFuture is offline because of the transformation $\text{geom view} \rightarrow \text{tree view}$ (and trees built with earliest future access times)

Thus (online equivalence)

Online BST algorithm (in tree view) that serves $R$ with cost $C$

$\iff$

Online geometric algorithm that serves $P$ (obtained from $R$) with cost $\Theta(C)$

**Proof** (easy)

Just run BST algorithm in tree-view at each time, plot in geometric view the points that are touched (visited by pointer)
Recall proof of offline equiv.

At time $i$:

- $T_i$: touched points at time $i$
- transform $T_i \rightarrow T_i'$, keep according to future access times.

Problem: we don't know the future.

Solution idea: do it locally.

- Splittable structure
- Splittable list
  
  (see doubling search lecture)

 amortized cost of split is $O(1)$

---

Split-tree data structure

Stores set $A$ of keys

- Node-split-tree ($A$) $\rightarrow O(|A|)$
- split ($x$) $\rightarrow O(1)$ amortized time
Implementation:
(0) - splittable lists
(1) - proper red black trees
(3) - any AVL / red-black / B-tree - trees (exercise)

\[ \text{need to use such tree-based implementation to remain in BST model.} \]

\[ \text{replace treap in proof of thm by split-tree.} \]

e.g. online greedy

\[ T_0 \]
\[ ABCD \]
\[ \text{split tree} \]

\[ T_1 \]
\[ C \]
\[ A \]
\[ B \]
\[ D \]

\[ \text{each node is a split tree} \]

\[ T_2 \]
\[ C \]
\[ A \]
\[ D \]

\[ T_3 \]
\[ C \]
\[ A \]
\[ D \]

\[ T_4 \]
\[ C \]
\[ A \]
\[ B \]

\[ T_5 \]
\[ C \]
\[ A \]
\[ D \]

\[ \text{A B C D} \]
Summary

Geometric view of BST
R = r_1, ..., r_m
view it as set of points

**offline equiv.**

- tree-strategy \rightarrow geometric alg.

**online equiv.**

- online
  - tree-strategy \leftrightarrow geometric algorithm row-by-row

- Greedy Future
- Online Greedy
  \leftrightarrow Greedy

**Conjecture**
Greedy is constant-competitive.

Like for Splay, conjecture is open.

Which is better? Splay or Greedy?

For both Splay and Greedy we cannot prove o(\log n) - competitive.

Is there any BST algorithm with o(\log n) - competitive cost?

\text{cost} \leq \text{OPT} = o(\log n)

**YES:** Tamo-tree is O(\log \log n) - competitive (next)

(best known)