Augmented trees

Binary Search Trees (BST)

**Purpose**

- Store a set $S \subseteq U$ (ordered)

(Distinct) operations:

- **search** $(x)$: $x \in S$? (find the node labeled $x$)
- **insert** $(x)$
- **delete** $(x)$

$O(\log n)$ for $|S| = n$

- Balanced tree: depth $\leq O(\log n)$
- Many strategies known: AVL, red-black tree, treaps, etc.

Without balance:

```
1
2
3
. . .
```

Balance based on rotation operation:

Can use to reconstruct tree, make it balanced.
Why use BSTs as a dictionary (instead of, e.g., hash tables):
- ordered is essential, e.g., can use keys only in comparison
- support more advanced operations any ordering of keys:
  - traverse all stored keys in order (in BST: inorder traversal)
  - predecessor/successor
  - interval queries $a \leq b \Rightarrow \{S \cap [a, b]\}$

**Augmented tree**
- store some additional structural data in nodes
- e.g., exercise sheet 2.

**Example Applications**

1. **Range/Select dictionary**
   - Given a set $S$ (stored in BST)
     - operations:
       - insert/delete an item
       - search for an item
       - select($k$): outputs $k$th smallest key in $S$
       - rank($x$): output rank of $x$ (how many elements in $S$ are $\leq x$)
   - recall: select($k$) is kept
     - run in $O(k)$ time (for all keys)
   - now: select($k$) in BST
     - run in $O(k \log n)$ time.

   ![Diagram of BST with select and rank operations](image)
idea: size of node $x$, size of subtree rooted at $x$, denoted $s(x)$ (including $x$)

list: maintain $s(x)$ under all update operations.

in AVL - tree

$\text{update}(x)$

need to increment all sizes on path root $\frac{s(x)}{x}$

need to decrement all sizes on path root $\frac{s(x)}{x}$

 AVL restructuring via rotations

example to show for a single rotation

We can achieve AVL-tree with augmentation $s(x)$

select ($k$, root)

```
if $k < s(r$.left$)+1$
    return select ($k,r$.left$)
else if $k > s(r$.left$)+1$
    return select ($k-s(r$.left$)-1,r$.right$)
else
    $k = s(r$.left$)+1$
    return $r$.
```
\[ \text{rad}(x, \text{root}) \]
\[ \text{rad}(x, r) \]
\[ \text{if } (x < r) \]
\[ \text{return } \text{rad}(x, r.\text{left}) \]
\[ \text{else if } (x = r) \]
\[ \text{return } (r.\text{left}) + 1 \]
\[ \text{else if } (x > r) \]
\[ \text{return } \text{rank}(x, r.\text{right}) + \text{size}(r.\text{left}) + 1 \]

---

7. Interval queries

\[ S = \{a_1, \ldots, a_n\} \]

\[ \text{input, delete, search} \]

\[ \text{count}(a, b) \rightarrow \text{return } |S \cap [a, b]| \]  \hspace{1cm} (\text{interval query})

\[ \text{count}(a, b) \rightarrow \text{return } S \cap [a, b] \]

\[ \text{rad}(b) - \text{rad}(a) \rightarrow [a, b] \rightarrow O(\log) \]

\[ O(\log m + k) \rightarrow \# \text{ items listed} \]
[a, b]
a' smallest item in the tree s.t. a' ≥ a
b' largest item in the tree s.t. b' ≤ b
\[ \frac{a'}{b'} \frac{b}{b} \]
report all items in \([a', b'] \cap S\)

\(O(\log n)\) element admissibility

\(O(\log n)\)
output all elements in subtrees \(O(\log n + k)\)

\(0 \leq \ldots \leq 1\)

extend to 2D

Query: axis-parallel rectangles

* list points that fall within rectangle
* then query points 2
Static case: no insert/delete

Build BST on $x$-coordinate of points

index by $x$-coordinate

Each point stored $O(\log n)$
Total storage $O(n \log n)$

$O(k^2 n + k)$

Have tree $T_x$ only if we actually repeat the items.

- How to make it dynamic?
  (Double but complicated)

- How to get rid of extra log factor ($n \log n$)

  $O(k \log n + k)$

  (Fractional cascading)

  Is trick to speed up simultaneous
  buddy searches in multiple sets.
Store intervals post queries

$S$: set of intervals $(I_1, ..., I_n)$

$\text{query}(x)$ — needs to look through intervals containing $x$

- Binary search tree
- Leaves store all "elementary intervals"

Input intervals $I_1, ..., I_n$

$I_k$ stored at a node $x$ if interval of $x$ contained in $I_k$

but interval of $\text{pred}(x)$ not contained in $I_k$

In how many nodes is $I_k$ stored (at most?)

Claim: at most $O(10^m)$ places $\Rightarrow O(n10^m)$ storage

Each node stores union of two child-intervals

Input (left) $I_1, ..., I_n$
Claim: at most two nodes of same level in tree store an actual
$I_k$

$\forall k \leq \frac{\log n + \epsilon}{\log 2}$

Proof by contradiction

Claim \text{2} \implies \text{Claim 1}

query($x$) = Find leaf that contains $x$
- output all ancestors stored in nodes on search path of $x$

$O(\log n + \log n)$

\{
    \text{insert, delete, rotation (AUL)}
\}

\text{Example:}

query(45) \rightarrow \text{return } I_3, I_4

query(6.1) \rightarrow I_5, I_4

Segment Tree