CSCB63 – Design and Analysis of Data Structures

Anya Tafliovich[1]

[1]based on notes by Anna Bretscher and Albert Lai

augmented data structures

An augmented data structure is simply an existing data structure modified to store additional information and / or perform additional operations.

Our task: a data structure that implements an ordered set/dictionary and, in addition to `insert`, `delete`, `search`, `union` (we'll see `union` shortly), etc., also supports two types of "rank queries":

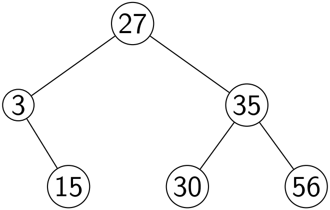
- `rank(S, k)`: given a key k in set S, what is its rank, i.e., the key's position among the elements?
- `select(S, r)`: given a rank r and set S, which key in S has this rank?

For example, in the set of values `S = `$\{$`3,15,27,30,35,56`$\}$:

- `rank(S, 15) =`
- `select(S, 4) =`

augmented data structures

For example, in the set of values `S = {3,15,27,30,35,56}`:



- `rank(S, 15) =`
- `select(S, 4) =`

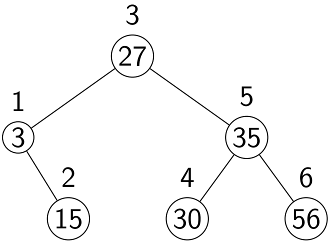
AVL tree without modification

If we use AVL tree without modifications:

- To implement `rank`:
- To implement `select`:
- What is the complexity of `rank`?
- What is the complexity of `select`?
- Will operations `search`, `insert`, and `delete` need to change?

augmented AVL tree — attempt 1

Idea: store $rank(T, n.key)$ in each node n in tree T.



- To implement `rank(T, k)`:

- To implement `select(T, r)`:

augmented AVL tree — attempt 1

Idea: store $rank(T, n.key)$ in each node n in tree T.

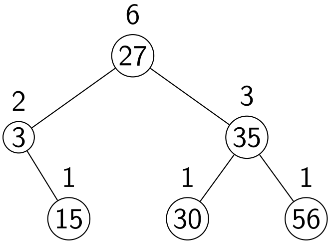
- What is the complexity of `rank(T, k)`?

- What is the complexity of `select(T, r)`?

- Will operations `search`, `insert`, and `delete` need to change?

augmented AVL tree — attempt 2

Idea: store `size(n)` — the number of nodes in subtree rooted at *n* including *n* itself — for each node *n*.

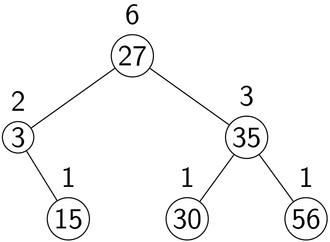


Q. How is size related to rank?
Define relative rank `rank(n, k)` as rank of key *k* relative to the keys in the tree rooted at node *n*.

augmented AVL tree — rank

`rank(T, k)` — idea

- do `search(T, k)` keeping track of the rank computed so far
- at each move to the right, add size of left subtree we skipped plus 1 for the key itself
- if found key in node n, add $size(n.left) + 1$ to rank so far, to get the real rank



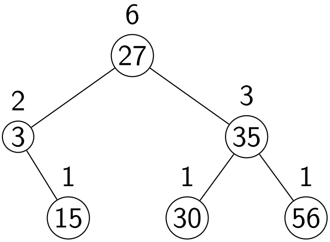
`rank(T, 35)`:

-
-
-
-

augmented AVL tree — rank

`rank(T, k)` — idea

- do `search(T, k)` keeping track of the rank computed so far
- at each move to the right, add size of left subtree we skipped plus 1 for the key itself
- if found key in node n, add $size(n.left) + 1$ to rank so far, to get the real rank



`rank(T, 15)`:

-
-
-

augmented AVL tree — rank

`rank(T, k)` — pseudocode

```
if T == nil:   # k not in T
    deal with special case
if k == T.key:
    return size(T.left) + 1
if k > T.key:
    return size(T.left) + 1 + rank(T.right, k)
else:
    return rank(T.left, k)
```

where

```
size(T) = 0 if T == nil else T.size
```

augmented AVL tree — select

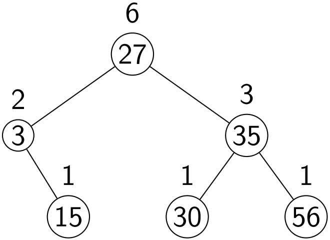
`select(T, r)` — idea

- at each visited node n, compare r to $size(n.left) + 1$
- if equal, found the node: return $n.key$
- if $<$, then key with rank r is in left subtree
  - relative rank in left subtree is the same
  - look for rank r in $n.left$
- if $>$, then key with rank r is in the right subtree
  - relative rank in the right subtree is $r - (size(n.left) + 1)$
  - look for rank $r - size(n.left) - 1$ in $n.right$

augmented AVL tree — select

`select(T, r)` — idea

- at each visited node n, compare r to $size(n.left) + 1$
- ...



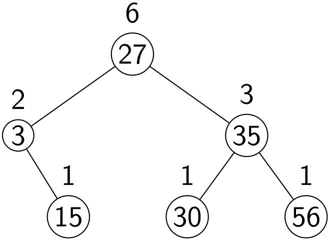
`select(T, 5)`:

-
-
-

augmented AVL tree — select

`select(T, r)` — idea

- at each visited node n, compare r to $size(n.left) + 1$
- ...



`select(T, 2)`:

-
-
-

augmented AVL tree — select

`select(T, r)` — pseudocode

```
if T == nil:   # r not in T
    deal with special case
r' = size(T.left) + 1
if r == r':
    return T.key
if r < r':
    return select(T.left, r)
else:
    return select(T.right, r - r')
```

where

```
size(T) = 0 if T == nil else T.size
```

augmented AVL tree — insert / delete

- `insert(T, k, v)`:

- `delete(T, k)`:

- rebalancing:

Therefore, each operation is $\Theta(\log n)$.