7.4 Augmenting Data Structures

Suppose you want to develop a data structure with:

- **Insert**\((x)\): insert element \(x\).
- **Search**\((k)\): search for element with key \(k\).
- **Delete**\((x)\): delete element referenced by pointer \(x\).
- **find-by-rank**\((\ell)\): return the \(\ell\)-th element; return “error” if the data-structure contains less than \(\ell\) elements.

Augment an existing data-structure instead of developing a new one.

How to augment a data-structure

1. choose an underlying data-structure
2. determine additional information to be stored in the underlying structure
3. verify/show how the additional information can be maintained for the basic modifying operations on the underlying structure.
4. develop the new operations

• Of course, the above steps heavily depend on each other. For example it makes no sense to choose additional information to be stored (Step 2), and later realize that either the information cannot be maintained efficiently (Step 3) or is not sufficient to support the new operations (Step 4).

• However, the above outline is a good way to describe/document a new data-structure.

Goal: Design a data-structure that supports insert, delete, search, and find-by-rank in time \(\Theta(\log n)\).

4. How does find-by-rank work?

\[
\text{Find-by-rank}(k) = \text{Select}(\text{root}, k)
\]

```markdown
Algorithm 11 Select(x, i)
1: if \(x = \text{null}\) then return error
2: if \(\text{left}[x] \neq \text{null}\) then \(r \leftarrow \text{left}[x].\text{size} + 1\) else \(r \leftarrow 1\)
3: if \(i = r\) then return \(x\)
4: if \(i < r\) then
5: \quad return Select(left[x], i)
6: else
7: \quad return Select(right[x], i - r)
```

Goal: Design a data-structure that supports insert, delete, search, and find-by-rank in time \(\Theta(\log n)\).

4. How does find-by-rank work?

Find-by-rank\((k) = \text{Select}(\text{root}, k)\) with

```python
Algorithm 11 Select(x, i)
1: if x = null then return error
2: if left[x] != null then r ← left[x].size + 1 else r ← 1
3: if i = r then return x
4: if i < r then
5: \quad return Select(left[x], i)
6: else
7: \quad return Select(right[x], i - r)
```
Select($x, i$)

Find-by-rank:
- decide whether you have to proceed into the left or right sub-tree
- adjust the rank that you are searching for if you go right

Rotations
The only operation during the fix-up procedure that alters the tree and requires an update of the size-field:

- $\text{LeftRotate}(x)$
- $\text{RightRotate}(z)$

The nodes $x$ and $z$ are the only nodes changing their size-fields. The new size-fields can be computed locally from the size-fields of the children.

7.4 Augmenting Data Structures
Goal: Design a data-structure that supports insert, delete, search, and find-by-rank in time $O(\log n)$.

3. How do we maintain information?

Search($k$): Nothing to do.

Insert($x$): When going down the search path increase the size field for each visited node. Maintain the size field during rotations.

Delete($x$): Directly after splicing out a node traverse the path from the spliced out node upwards, and decrease the size counter on every node on this path. Maintain the size field during rotations.

Augmenting Data Structures

Bibliography
MIT Press and McGraw-Hill, 2009

See Chapter 14 of [CLRS90].