Parallel Algorithms
Lecture 2

Umut Acar
Carnegie Mellon University

July 12, 2018

1 Recap

Binary Search Trees
Key Functions:

- find
- insert
- delete
- intersection
- union
- difference
- split
- join
- joinM

$$\text{Work: } O\left(m * lg\left(\frac{m+n}{m}\right)\right)$$
$$\text{Span: } O(lg^2(m+n))$$

Achieve this by breaking up the largest tree into m trees of size $\frac{n}{m}$

2 Balancing: Randomized Technique

One of many ways to balance a tree - treaps.
Assuming sets and dictionaries (no duplicates).

2.1 What is the need for balancing?

The probability of getting the worst possible case for a binary tree is very low, but is still possible. It would be nice to exploit this property in a probabilistic model to maintain balance in our tree.

2.2 Threaps

Assign a priority to each key.

- Priorities are random.
- Pretend priorities are the pertubation.
- Priorities are heap ordered
- Keys are tree ordered
- Tree-Heap = Treap

```
datatype α treap = NODE of α treap × (α × int) × α treap
                 | LEAF

fun singleton k =
    let
        val p = random()
    in
        NODE (LEAF, (k, p), LEAF)
    end

fun split t k =
    case t of
        LEAF => (false, LEAF, LEAF)
        NODE (l, (kk, p), r) =>
            if k = kk then
                (true, l, r)
            else if k < kk then
                let
                    val(found, ll, rr) = split l k
                in
                    (found, ll, NODE (rr, (kk,p), r))
                end
            else
                let
                    val(found, ll, rr) = split r k
                in
                    (found, NODE (l, (kk,p), ll), rr)
                end
```

```
            end
    end

fun join t u =
    case (t, u) of
        (LEAF, u) => u
        (t, LEAF) => t
        (NODE (lt, (kt, pt), rt),
         NODE (lu, (ku, pu), ru)) =>
            if pt < pu then
                NODE (lt, (kt, pt), (join rt u))
            else
                NODE ((join t lu), (ku, pu), ru)
            end
    end
```

2.3 Augmentation

BST's allow for looking up by key, which is useful, but what if we want to search by the "5th" person, or "8th" person?
So need to create an augmentation function that is carried by the code.
Augmentation: Keep the number of nodes (including the current node) below each node in the value for each node.
Select Query: Gives the ith element in the sorted order of keys in lg n work.
Rank Query: Given a key find the number of keys less than that key.
So keep the size of each node in each node, can use a function called makeNode to achieve this, instead of manually creating the nodes.