Class 4

Yesterday: applications of sequences

- maximum contiguous subsequence sum
- BFS on graphs — scan $O(n)$ W } optimal
  $O(\lg n)$ S }

  layer-by-layer starting from source

  Span: O(diameter) graph
  e.g. n for filters

  inject gives remove duplicates $O(m)$ work

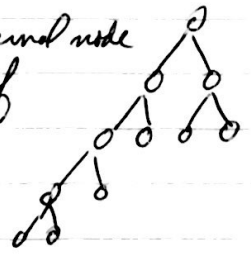
can parallelize other graph algorithms

analogous thing to contract w/ graphs

Binary search trees (sets & dictionaries)

Binary search tree

either internal node or leaf



keys on internal nodes

datatype α tree = NODE (α tree, α, α tree)
| LEAF

property: key at node
> keys on left subtree
< keys on right subtree

Scanned with CamScanner

operations:

find: $\alpha \times \alpha$ tree $\to$ bool

insert: $\alpha \times \alpha$ tree $\to \alpha$ tree

delete: $\alpha \times \alpha$ tree $\to \alpha$ tree

need bulk version of these ops.

bulk ~~find~~ =
intersection: α t_1 tree $\times \alpha$ t_2 tree $\to \alpha$ tree
only keys in both $k(t_1) \cap k(t_2)$

bulk insert
union: α tree $\times \alpha$ tree $\to \alpha$ tree
contains keys in both $k(t_1) \cup k(t_2)$

bulk delete
difference: α tree $\times \alpha$ tree $\to \alpha$ tree
has keys in first but not second
$k(t_1) \setminus k(t_2)$

design algs:
work efficient
low span

Scanned with CamScanner

Sequential BST: balancing

balance
- splay trees
- red-black
- AVL trees
- weight-balance trees (great to impl sequences)
- treeps - probabalistically balanced trees

ops: trees w/ balancing by using only split & join

split: α tree $\times$ α $\rightarrow$ bool $\times$ α tree $\times$ α tree

is key present | keys < | keys >

tree structure not important

join: α tree $\times$ α tree $\rightarrow$ α tree

< key | > keys | combined

requires:

like union except for requirement

Scanned with CamScanner

minimalist implementation based on split & join & singleton

find k t = let found, _, _ = split t k in found

insert k t = let found, l, r = split t k
in
if found then t
else
join (l, join (singleton k), r)
≡ joinM l k r

singleton k = NODE (LEAF, k, LEAF)
joinM : α tree × α × α tree → α tree

delete k t = let _, l, r = split t k
in join (l, r)

assume
lg work for split join

find, insert, delete all have lg work

Scanned with CamScanner

```
intersection t u =

  case (t, u) of

    (LEAF, u) => LEAF
    (t, LEAF) => LEAF
    (NODE(l1, k, r1), u) => let flag, l2, r2 = split u, k
                                l, r = intersect(l1, l2)
                                    || intersect(r1, r2)
                            in if flag
                               then joinM(l, k, r)
                               else join(l, r)

                            end

union t u =

  case t of

    LEAF => u
    NODE(l1, k, r1) =>

      let flag, l2, r2 = split u k

          l, r = union(l1, l2) || union(r1, r2)

      in joinM(l, k, r)
```

Scanned with CamScanner

Work

assume $m \leq n$

union

$W(m, n)$ — split 2 recursive calls join

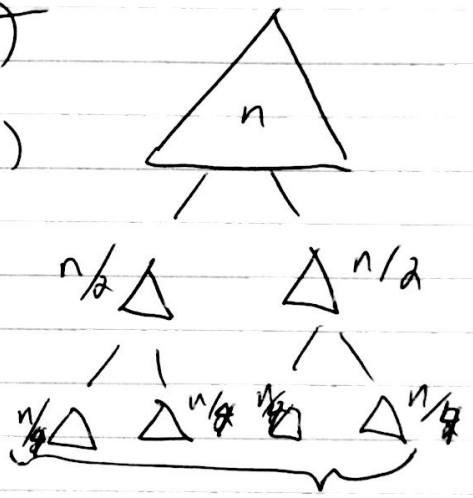
$= 2(W(\frac{m}{2}, \frac{n}{2})) + O(\lg(m+n))$

$= \cancel{O(m+n)}$

$= m \cdot \lg(m+n)$

$O(m \cdot \lg \frac{n+m}{m})$

$O(m \cdot \lg \frac{n}{m})$



(each size $\frac{n}{m}$) m of these

only working on the smaller tree

if n, m are about the same,

get $O(n)$ which is optimal

if m is a lot larger, looks more like:

$O(m \cdot \lg n)$

OPTIMAL alg in terms of work

Scanned with CamScanner

~~Span~~ $O(\log^2(m+n))$

impl many diff algs for balancing
works well in practice

if not balanced,
it improves work & span!

Scanned with CamScanner