COMP2521 19T0 lec12 cs2521@ jashankj@

Sorting

Balance Trees

COMP2521 19T0 Week 7, Tuesday: A Question of Balance

Jashank Jeremy
jashank.jeremy@unsw.edu.au

radix sort balanced trees

cs2521@ jashankj@

Sorting

Non-Compari

Balance Trees

Sorting

Sorting Non-Comp Radix

Balance

Can we decompose our keys?
Radix sorts let us deal with this case.

Keys are values in some base-R number system. e.g., binary, R=2; decimal, R=10; ASCII, R=128 or R=256; Unicode, $R=2^{16}$

Sorting individually on each part of the key at a time: digit-by-digit, character-by-character, rune-by-rune, etc.

cs2521@ jashankj@

Sorting

Radix

Radix Sorting, Most-Significant-Digit First

Consider characters, digits, bits, runes, etc., from left to right; partitioning input into R pieces according to key . 0; recurse into each piece, using succesive keys — key . 1, key . 2, ..., key . w

with R = 2, roughly a quicksort.

Radix Sorting, Least-Significant-Digit First

cs2521@ jashankj@

Non-Compari Radix

Balance Trees Consider characters, digits, bits, runes, etc., from right to left; use a **stable** sort using the *d*th digit as key, using (e.g..) key-indexed counting sort.

```
1019
      2301
             3129
                    2122
1019
      2301
             3129
                    2122
2301
      2122
             1019
                    3129
2301
      1019
             2122
                    3129
1019
      2122
             3129
                    2301
1019
      2122
             2301
                    3129
```

this will not work if the sort is not stable!

Sorting Non-Compari

Radix Balance Trees Complexity: $O\left(w\left(n+R\right)\right)\approx O(n)$, where w is the 'width' of data; the algorithm makes w passes over n keys

LSD

Not in-place: O(n+R) extra space required. May be stable! Usable on variable length data.

MSD

Not in-place: O(n+DR) extra space required. (D is the recursion depth.) May be stable! Usable on variable length data. Can complete before examining all of all keys.

COMP2521 19T0 lec12 cs2521@ jashankj@

Sorting

Balanced Trees

Reca

Searching

irees, bire

Search Tree

Properties

Fillindve

Rotatioi

Partition

Simple Approa

Global

Root Inser

Pandom Tr

Complex Approach

Splay

Balanced Trees

Sorting

Balance Trees

Searching

Search T

Propertie

Primitiv

Rotati

Simple Approach

Global Poot Insert

Random Trees Complex Approaches Common variations:

input a key value

keys are unique; key value matches 0 or 1 items

output item(s) containing that key

- multiple keys in search, items containing any key
- multiple keys in search/item, items containing all keys

We assume: keys are unique, each item has one key.

Jording

Balance

Recap

Trees, BTrees

Search T

Propertie

Primitive

Rotation

- unutition

Simple Approache

Root Insert

Random Trees

Complex Approache

Trees are branched data structures, consisting of nodes and edges, with no cycles.

Each node contains a value. Each node has edges to $\leq k$ other nodes. For now, k=2 — binary trees

Trees can be viewed as a set of nested structures: each node has k (possibly empty) subtrees.

Search Trees

For all nodes in the tree:

the values in the left subtree are less than the node value the

values in the right subtree are greater than the node value

A binary tree of *n* nodes is degenerate if its height is at most n-1.

A binary tree of *n* nodes is balanced if its height is at least $|\log_2 n|$.

Structure tends to be determined by order of insertion:

[4, 2, 1, 3, 6, 5, 7] vs [6, 5, 2, 1, 3, 4, 7]

The Worst Case

Palanco

Balance

Recap

Searchin Trees Pi

Search Trees

Search fre

- i iii

Rotatio

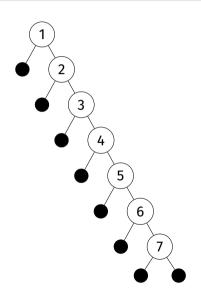
Partition

Simple Approache

Root Insert

Complex Approaches

Ascending-ordered or descending-ordered data is a pathological case: we always right- or left-insert along the spine of the tree.





Sorting

Balance

Recan

.....

Trees, BT

Search Trees

Dropostica

Paul on Intern

Rotatio

Dartitio

Simple Appro

0100011

Root msert

Complex Approach

Splay

Binary Search Trees

Performance

Cost for insertion:

balanced $O(\log_2 n)$, degenerate O(n) (we always traverse the height of the tree)

Cost for search/deletion:

balanced $O(\log_2 n)$, degenerate O(n) (worst case, key $\notin \tau$; traverse the height)

We want to build balanced trees.

Properties

PERFECTLY BALANCED

a weight-balanced or size-balanced tree has. for every node.

$$|\operatorname{SIZE}(l) - \operatorname{SIZE}(r)| < 2$$

LESS STRINGENTLY

a height-balanced tree has. for every node.

$$|\text{HEIGHT}(l) - \text{HEIGHT}(r)| < 2$$

Balanced or Not?

(I)

Properties

SIZE $(\tau_4) = 5$ SIZE $(\tau_2) = 3$

SIZE $(\tau_5) = 1$

SIZE $(\tau_1) = 1$ SIZE $(\tau_3) = 1$

SIZE (τ_2) – SIZE $(\tau_5) = 2$

HEIGHT $(\tau_4) = 2$ HEIGHT $(\tau_2) = 1$

HEIGHT $(\tau_5) = 0$ HEIGHT $(\tau_1) = 0$

HEIGHT $(\tau_3) = 0$

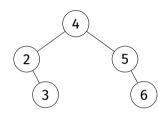
 $\text{HEIGHT}(\tau_2) - \text{HEIGHT}(\tau_5) = 1$ HEIGHT BALANCED

NOT SIZE BALANCED

HEIGHT $(\tau_4) = 2$

jashankj@

Properties



SIZE $(\tau_4) = 5$ SIZE $(\tau_2) = 2$ SIZE $(\tau_5) = 2$ SIZE $(\tau_3) = 1$ SIZE $(\tau_6) = 1$

HEIGHT $(\tau_2) = 1$ HEIGHT $(\tau_5) = 1$ HEIGHT $(\tau_3) = 0$ HEIGHT $(\tau_6) = 0$ SIZE BALANCED HEIGHT BALANCED

2011111

Balance

Reca

Searchin

Trees, BTr Search Tr

Properties

Primitiv

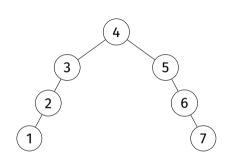
Partition

Clabal

Boot Insert

Pandom Troos

Complex Approaches



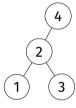
Let's look at τ_3 . SIZE $(\tau_2) = 2$ SIZE $(\tau_{\varnothing}) = 0$ $2 - 0 = 2 \nleq 2$

HEIGHT
$$(\tau_{\varnothing}) = 0$$

HEIGHT $(\tau_6) = 1$
 $|0-1| = 1 < 2$
HEIGHT BALANCED

Let's look at τ_5 .

Properties



Let's look at τ_4 .

SIZE
$$(\tau_2) = 3$$

SIZE $(\tau_{\varnothing}) = 0$
 $3 - 0 = 3 \nleq 2$

NOT SIZE BALANCED

Let's look at τ_4 .

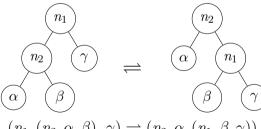
HEIGHT
$$(au_2)=1$$
 HEIGHT $(au_arnothing)=0$ $1-0=1<2$ HEIGHT BALANCED

Rebalancing Primitives

Rotation

LEFT ROTATION and RIGHT ROTATION:

a pair of 'primitive' operations that change the balance of a tree whilst maintaining a search tree.



$$(n_1, (n_2, \alpha, \beta), \gamma) \rightleftharpoons (n_2, \alpha, (n_1, \beta, \gamma))$$

```
Sorting
```

Balanced

```
Recap
```

Searching Trees, BTree

Search T

Propertie:

Rotation

Simple Approaches

Global

Complex Approaches

```
btree_node *btree_rotate_right (btree_node *n1)
{
    if (n1 == NULL) return NULL;
    btree_node *n2 = n1->left;
    if (n2 == NULL) return n1;
    n1->left = n2->right;
    n2->right = n1;
    return n2;
}
```

 n_1 starts as the root of this subtree and is demoted; n_2 starts as the left subtree of this tree, and is promoted.

```
Sorting
```

Balance

Recap

Searching Trees, BTre

Search Tr

Propertie:

Rotation

Simple Approaches

Global

Root Insert Random Trees

Complex Approaches Splay

```
btree_node *btree_rotate_left (btree_node *n2)
{
    if (n2 == NULL) return NULL;
    btree_node *n1 = n2->right;
    if (n1 == NULL) return n2;
    n2->right = n1->left;
    n1->left = n2;
    return n1;
}
```

 n_2 starts as the root of this subtree and is demoted; n_1 starts as the right subtree of this tree, and is promoted.

Rotation in Context

Sortin

Balance

Reca

Searching Trees, BT

Properties

Primitives

Partition

Simple Approaches

Root Insert

Complex Approaches

Splay

A way to brute-force some balance into a tree: lifting some kth index to the root.

```
PARTITION :: BTree \rightarrow Word \rightarrow BTree PARTITION Empty k = Empty PARTITION (Node n \ l \ r) k = \mid k < SIZE l = ROTATER (Node n \ ( PARTITION l \ k) r) \mid SIZE l < k = ROTATEL (Node n \ l \ ( PARTITION r \ (k-1- SIZE l))) \mid otherwise = Node n \ l \ r
```



Sorting

Balance

Pocar

кеса

- --

Sparch Tr

Dropostio

Primitive

Rotatio

Partition

Clobal

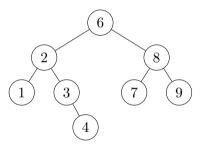
Poot Insert

Random Tree

Complex Approaches



Partition Partition in Context



What happens if we partition at index 3 (node 4)?

```
COMP2521
19T0 lec12
```

Partition

C Implementation

```
Sorting
```

Balance

Recap

Searching Troos PTro

Search T

Properties

Rotati

Partition

Simple Approaches Global

Root Insert

Complex Approache Splay

```
btree node *btree partition (btree node *tree, size t k)
    if (tree == NULL) return NULL;
    size t lsize = size (tree->left);
    if (lsize > k) {
        tree->left = btree_partition (tree->left, k);
        tree = btree rotate right (tree);
    if (lsize < k) {</pre>
        tree->right = btree partition (tree->right, k - 1 - lsize):
        tree = btree_rotate_left (tree);
    return tree;
```

.

Balance

Reca

Search

Troos RT

Sparch

......

Propertie

Primitives

Rotatio

Partition

Simple App

Global

Root Insert

Random Trees

Complex Approach

With our primitive operations in hand —

ROTATEL :: BTree \rightarrow BTree

ROTATER :: BTree \rightarrow BTree

 $\mathtt{PARTITION} :: BTree \to Word \to BTree$

— let's go balance some trees!

Approach #1: Global Rebalancing

cs2521@ jashankj@

Sorting

Balance

Reca

Searching

Search Tr

Propertie

Drimitivo

Rotatio

Partition

Clobal

Global

Root Insert

Complex Approache

Move the median node to the root, by partitioning on $\operatorname{SIZE} \tau/2$; then, balance the left subtree, and balance the right subtree.

```
btree_node *btree_balance_global (btree_node *tree)
{
    if (tree == NULL) return NULL;
    if (size (tree) < 2) return tree;
    tree = partition (tree, size (tree) / 2);
    tree->left = btree_balance_global (tree->left);
    tree->right = btree_balance_global (tree->right);
    return tree;
}
```

Problems

Global

 cost of rebalancing: for many trees, O(n); for degenerate trees, $O(n \log n)$

- what if we insert more keys?
 - rebalance on every insertion
 - rebalance every k insertions; what k is good?
 - rebalance when imbalance exceeds threshold.

we either have more costly instertions or degraded performance for (possibly unbounded) periods. ... given a sufficiently dynamic tree, sadness.



Global vs Local Rebalancing

cs2521@ jashankj@

Balance

Recar

Searchin

T..... D7

Search

Propertie

Potatio

Rotatio

Partition .

Global

Poot Inco

Root mse

Compley Approac

Splay

GLOBAL REBALANCING

walks every node, balances its subtree; ⇒ perfectly balanced tree — at cost.

LOCAL REBALANCING

do small, incremental operations to improve the overall balance of the tree ... at the cost of imperfect balance



Sorting

Balance

Reca

Searchir

Trees, BT

Search

Properti

Primitiv

Rotat

Simple Approa

Global

Poot Insert

Dandon Tro

Complex Approa

amortisation: do (a small amount) more work now to avoid more work later randomisation: use randomness to reduce impact of BST worst cases optimisation: maintain structural information for performance

Local Rebalancing Approaches

Juling

Balanc

Recap

Searching

Search Tr

Properties

Delimite

Potatio

D-----

Simple Appro

Global

Root Insert

Complex Approach

How do we insert a node at the root of a tree? (Without having to rearrange all the nodes?)

We do a leaf insertion and rotate the new node up the tree.

More work? **No!**Same complexity as leaf insertion,
but more actual work is done: amortisation.

(Side-effect: recently-inserted items are close to the root. Depending on what you're doing, this might be very useful!)

Root Insertion

C Implementation

```
iashanki@
```

Root Insert

```
btree node *btree insert root (btree node *tree, Item it)
    if (tree == NULL)
        return btree_node_new (it, NULL, NULL);
   if (less (it, tree->value)) {
        tree->left = btree insert root (tree->left, it);
        tree = btree rotate right (tree):
    } else {
        tree->right = btree_insert_root (tree->right, it);
        tree = btree rotate left (tree):
    return tree;
```

Randomised Insertion

cs2521@ jashankj@

Balance

Recai

Searching

.....

Search II

Properties

Primitive

Rotatio

Partition

Global

Root Insert

Random Trees

Complex Approache Splay BSTS don't have control over insertion order. worst cases — (partially) ordered data — are common.

to minimise the likelihood of a degenerate tree, we randomly choose which level to insert a node; at each level, probability depends on remaining tree size.

> do a 'normal' leaf insertion, most of the time. randomly (with a certain probability), do a root insertion of a value.

Randomised Insertion

C Implementation

```
Sorting
```

```
Balance
Trees
```

Recap Searching Trees BTree

Search T

Propertie

Primitive

Partition
Simple Approach

Global Root Insert

Random Trees Complex Appro

Complex Approa Splay

```
btree node *btree insert rand (btree node *tree, Item it)
    if (tree == NULL)
        return btree_node_new (it, NULL, NULL);
    if (rand () < (RAND_MAX / size (tree)))</pre>
        return btree insert root (tree, it):
    else if (less (it, tree->value))
        tree->left = btree insert rand (tree->left, it);
    else
        tree->right = btree_insert_rand (tree->right, it);
    return tree;
```



Sorting

Balance

Recan

Convehic

Trees, BT

Search T

Droportio

Propertie

.

Rotatioi

r di dicioni

Simple Api

Global

Random Trees

Complex Approach

Randomised Insertion

Properties

building a randomised BST is equivalent to building a standard BST with a random initial permutation of keys.

worst-case, best-case, average-case performance: same as a standard BST but with no penalty for ordering!



Randomised Deletion

cs2521@ iashanki@

Random Trees

We could do something similar for deletion: when choosing a node to promote, choose randomly from the in-order predecessor or successor

Sorting

Balance

Reca

Searchin

Trees, B

Search

Properti

Partitio

Simple Approa

Global

Root Insert

Random Trees

Splay

Root insertion can still leave us with a degenerate tree.

Splay trees vary root-insertion,
by considering three levels of the tree
— parent, child, grandchild —
and performing double-rotations based on p-c-g orientation;
the idea: double-rotations improve balance.

No guarantees, but improved performance.

"... their performance is amortised by the amount of effort required to understand them."

- me, 2016

Sortin

Balance

Reca

Searching

Trees, BTre

Search T

Properties

rioperde:

Potation

Dartition

Simple Appro-

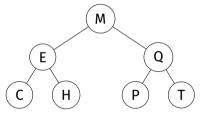
Global

Root Insert

Random Tre

Complex Ap Splay

Four choices to consider for a double-rotation:



1: LL 2: LR

3: RL 4: RR

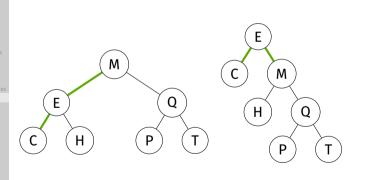


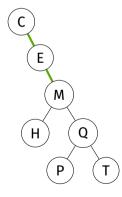
Splay Rotations

Double-Rotation: Left, Left

Splay

ROTATER au_{M} ROTATER τ_{E}





cs2521@ jashankj@

Sorting

Balanced

Rocar

Searchi

Trees, BTr

Search 1

Propertie

rioperue

Potation

Partition

Simple App

Global

Root Insert

Daniel and Taxan

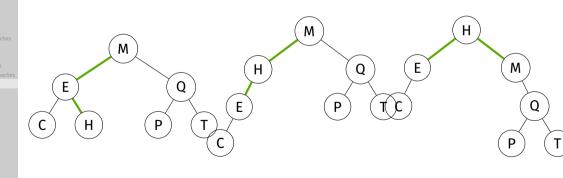
Compley Ar

Splay

Splay Rotations

Double-Rotation: Left, Right

ROTATEL au_{E} ROTATER au_{M}



cs2521@ jashankj@ Splay Rotations
Double-Rotation: Right, Left

Sorting

Balanced

Recar

Searching

Convoluti

Search

Propertie

D-4-4'--

Partition

Simple Appr

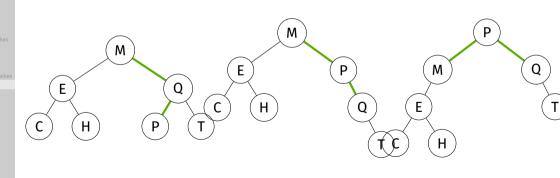
Global

Root Insert

Complex Apr

Splay

ROTATER au_{Q} ROTATEL au_{M}



cs2521@ jashankj@

Splay Rotations

Double-Rotation: Right, Right

2011111

Balance

Reca

Searchin

Search

Searcii

Propertion

Primitiv

Rotatio

Partition

Simple Ap

Global

Root Insert

1101100111

Splay

ROTATEL τ_{M} ROTATEL τ_{Q}

