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@ iashanki@

Sorting
Non-Compariso

Radix Balance Trees

# 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

cs2521@ jashankj@

Sorting
Non-Compari
Radix

Balance Trees

# 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

1019 | 2301 | 3129 |

cs2521@ jashankj@

Sorting Non-Comparis

Radix Balance Trees

# 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
```

```
        1019
        2301
        3129
        2122

        1019
        2301
        3129
        2122
```

cs2521@ jashankj@

Sorting Non-Compariso

Radix Balance Trees

# 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

```
    1019
    2301
    3129
    2122

    1019
    2301
    3129
    2122

    1019
    2301
    2122
    3129
```

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.

cs2521@ jashankj@

Sorting
Non-Compariso
Radix

Balanceo Trees

cs2521@ jashankj@

Sorting Non-Compa

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

1019 | 2301 | 3129 | 2122

cs2521@ jashankj@

Non-Compari Radix

Balance Trees

```
    1019
    2301
    3129
    2122

    1019
    2301
    3129
    2122
```

cs2521@ jashankj@

Non-Compari Radix

Balance Trees

```
    1019
    2301
    3129
    2122

    1019
    2301
    3129
    2122

    2301
    2122
    1019
    3129
```

cs2521@ jashankj@

Non-Compari Radix

Balance Trees

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

cs2521@ jashankj@

Non-Compar Radix

Balance

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

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-Compariso
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

Primitive:

Rotation

Partitio

Simple Appro-

Clabal

Root Inser

Pandom Tr

Complex Approache

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.

#### Sorting

Balance

Reca

Searchin

Search Trees

Search II

Properties

Primitive

Rotation

- di dicioni

Simple Approache

Global

Random Trees

Complex Approach

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

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  $|\log_2 n|$ .

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]

### giiiiioo

Balance

Recar

Searchin Troop BT

Search Trees

Search fre

Propertie:

Potatio

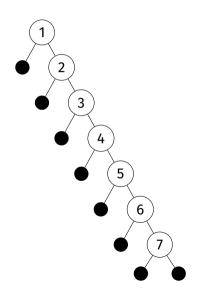
Partition

Simple Approach

Root Insert

Complex Approache

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

#### Droportion

Rotatio

Partitio

Simple Appro-

0.0000

Rootilisert

Compley 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)



#### Sorting

#### Balance

Docon

Reca

Troos RTr

Search Trees

Donostino

- - - - - -

Rotatio

Dartitio

Simple Approa

Globat

Root Insert

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.

#### ----

#### Balance

Reca

Searchin

Trees, BTre

#### Properties

FIIIIIIIIVE

Rotation

Partition

Simple Approache

Clobal

Root Insert

Complex Approach

### PERFECTLY BALANCED

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

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

#### Properties

### PERFECTLY BALANCED

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

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

#### LESS STRINGENTLY

a height-balanced tree has. for every node.

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

Juiting

Balance

Recap

Sean

Trees, BT

Search

#### Properties

Primit

Rotati

Partition

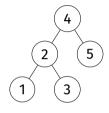
Jimpte Ap

Global

Daniel and T

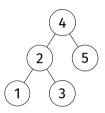
Complex Approach

Splay



jashankj@

Properties



SIZE 
$$(\tau_4) = 5$$
  
SIZE  $(\tau_2) = 3$   
SIZE  $(\tau_5) = 1$   
SIZE  $(\tau_1) = 1$   
SIZE  $(\tau_3) = 1$ 

## Balance

Tree

Reca

Trees, BTr

#### Properties

Primi

Rotatio

Simple

Global

Root Inse

Random Trees

Complex

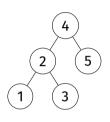
2 5

$$\begin{aligned} \operatorname{SIZE}\left(\tau_{4}\right) &= 5\\ \operatorname{SIZE}\left(\tau_{2}\right) &= 3\\ \operatorname{SIZE}\left(\tau_{5}\right) &= 1\\ \operatorname{SIZE}\left(\tau_{1}\right) &= 1\\ \operatorname{SIZE}\left(\tau_{3}\right) &= 1\\ \operatorname{SIZE}\left(\tau_{2}\right) - \operatorname{SIZE}\left(\tau_{5}\right) &= 2 \end{aligned}$$

NOT SIZE BALANCED

# iashanki@

#### Properties



SIZE 
$$(\tau_4) = 5$$
  
SIZE  $(\tau_2) = 3$   
SIZE  $(\tau_5) = 1$ 

SIZE 
$$(\tau_5) = 1$$
  
SIZE  $(\tau_1) = 1$ 

$$SIZE(\tau_3) = 1$$
$$SIZE(\tau_2) - SIZE(\tau_5) = 2$$

**NOT SIZE BALANCED** 

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

HEIGHT  $(\tau_4) = 2$ 

HEIGHT  $(\tau_2) = 1$ 

HEIGHT  $(\tau_3) = 0$ 

Balanced or Not?

#### Properties

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

SIZE  $(\tau_5) = 1$ 

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

SIZE  $(\tau_2)$  – SIZE  $(\tau_5) = 2$ NOT SIZE BALANCED

HEIGHT  $(\tau_5) = 0$ HEIGHT  $(\tau_1) = 0$ HEIGHT  $(\tau_3) = 0$ 

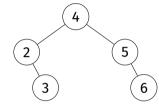
HEIGHT  $(\tau_4) = 2$ 

HEIGHT  $(\tau_2) = 1$ 

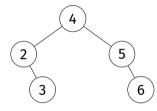
 $\text{HEIGHT}(\tau_2) - \text{HEIGHT}(\tau_5) = 1$ HEIGHT BALANCED

jashankj@

#### Properties



#### Properties



SIZE 
$$(\tau_4) = 5$$

SIZE 
$$(\tau_2) = 2$$

SIZE 
$$(\tau_5) = 2$$

SIZE 
$$(\tau_3) = 1$$

SIZE 
$$(\tau_6) = 1$$

# jashankj@

## Balance

#### irees

Reca

Trees, BTre

#### Properties

. . . . .

Postitio

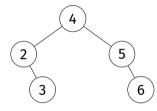
Simple Appr

Global

Root Insert

Pandom Tro

Complex Approach



SIZE 
$$(\tau_4) = 5$$
  
SIZE  $(\tau_2) = 2$   
SIZE  $(\tau_5) = 2$ 

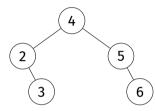
SIZE 
$$(\tau_3) = 1$$

SIZE 
$$(\tau_6) = 1$$

SIZE BALANCED

jashankj@

#### Properties



SIZE 
$$(\tau_4) = 5$$
  
SIZE  $(\tau_2) = 2$ 

SIZE 
$$(\tau_2) = 2$$
  
SIZE  $(\tau_5) = 2$ 

SIZE 
$$(\tau_3) = 2$$
  
SIZE  $(\tau_3) = 1$ 

SIZE 
$$(\tau_6) = 1$$

HEIGHT 
$$(\tau_4)=2$$

HEIGHT 
$$(\tau_2) = 1$$

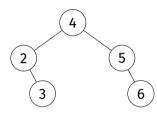
HEIGHT 
$$(\tau_5)=1$$

HEIGHT 
$$(\tau_3) = 0$$

HEIGHT 
$$(\tau_6) = 0$$

jashankj@

#### Properties



SIZE 
$$(\tau_4) = 5$$
  
SIZE  $(\tau_2) = 2$ 

SIZE 
$$(\tau_2) = 2$$
  
SIZE  $(\tau_5) = 2$ 

SIZE 
$$(\tau_3) = 1$$

SIZE 
$$(\tau_6) = 1$$

HEIGHT 
$$(\tau_4)=2$$

HEIGHT 
$$(\tau_2) = 1$$

HEIGHT 
$$(\tau_5) = 1$$

$$\operatorname{HEIGHT}(\tau_3) = 0$$

$$\operatorname{HEIGHT}\left(\tau_{6}\right)=0$$

HEIGHT BALANCED

### Sortin

### Balance

Recap

.....

Trees, BTI

Search 1

#### Properties

Primiti

Rotatio

Partition

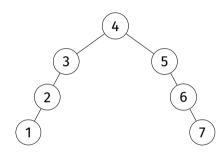
Clabal

Poot Inco

Pandom T

Complex Approache

Splay



### Balance

### Troos

Recap

Trone BT

Trees, BTre

### Properties

Primitiv

Partition

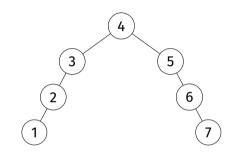
Simple Appro

Global

Root Insert

kandom frees

Complex Approaches
Splay



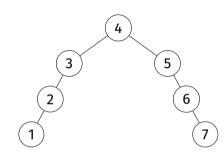
### Let's look at $\tau_3$ .

SIZE 
$$(\tau_2)=2$$

$$\operatorname{SIZE}\left(\tau_{\varnothing}\right)=0$$

$$2 - 0 = 2 \not< 2$$

### Properties



### Let's look at $\tau_3$ .

SIZE 
$$(\tau_2)=2$$

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

**NOT SIZE BALANCED** 

### Balance

### Trees

Reca

Trees, BT

#### Properties

Primiti

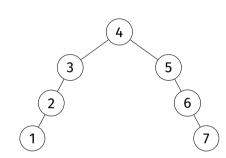
Dartition

Simple Approa

Boot Incort

Pandom Trees

Complex Approache



## Let's look at $au_3$ .

SIZE  $(\tau_2) = 2$ 

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

**NOT SIZE BALANCED** 

Let's look at  $\tau_5$ .

 $\operatorname{HEIGHT}\left(\tau_{\varnothing}\right)=0$ 

HEIGHT  $(\tau_6) = 1$ 

|0-1|=1<2

### 2011111

### Balance

Reca

Searchin

Trees, BTr Search Tr

#### Properties

Primitiv

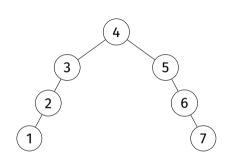
Partition

Clabal

Boot Insert

Pandom Troos

Complex Approaches



# Let's look at $\tau_3$ . 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  $\tau_5$ .

### Sortin

### Balance

Recar

Reca

Toron DT

Search T

### Properties

Primiti

Partition

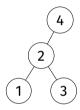
Simple Appr

Global

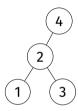
Dandom T

Complex Approache

Splay



### Properties



### Let's look at $\tau_4$ .

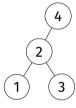
SIZE 
$$(\tau_2)=3$$

$$\operatorname{SIZE}\left(\tau_{\varnothing}\right)=0$$

$$3 - 0 = 3 \not< 2$$

**NOT SIZE BALANCED** 

#### Properties



Let's look at  $\tau_4$ .

SIZE 
$$(\tau_2) = 3$$
  
SIZE  $(\tau_{\varnothing}) = 0$   
 $3 - 0 = 3 \nleq 2$ 

NOT SIZE BALANCED

Let's look at  $\tau_4$ .

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



### **Rotations**

**Rebalancing Primitives** 

### Sorting

Balance

Reca

Searching

Search T

Droportio

- i operae.

#### Rotation

Doublelow

Simple Approac

Global

Root Insert

Complex Approach

LEFT ROTATION and RIGHT ROTATION:

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

**Rebalancing Primitives** 

### Sorting

### Balance

#### Reca

Trees, BT

Propertie

Doloo laloo

#### Rotation

Simple Approach

Global

Root Insert

Random Tree:

Complex Approaches
Splay

### LEFT ROTATION and RIGHT ROTATION:

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

$$\begin{array}{cccc}
 & n_1 & & & & & \\
 & n_2 & \gamma & & & & \\
 & \alpha & \beta & & & & \\
 & \alpha & \beta & & & & \\
 & (n_1 & (n_2 & \alpha, \beta), \alpha) \Rightarrow (n_2 & \alpha, (n_2 & \beta, \alpha))
\end{array}$$

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

```
Sortin
```

### Balance

Recap

Searching

Search Tr

Propertie:

### Rotation

Partition
Simple Approaches

Global

Root Insert Random Trees

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.

```
Sortin
```

### Balance

Reca

Searching Trees, BTre

Search Tr

Properties Primitives

### Rotation

Simple Approaches

Global Boot Insort

Root Insert Random Trees

Complex Approaches

```
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.



# Partition

**Rotation in Context** 

### sortin

Balance

Reca

Searching Trees, BTr

Search T

Propertie

Primitive

Rotatio

#### Partition

Global

Root Inser

Random Tre

Complex Approaches
Splay

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



cs2521@ iashanki@

### **Partition Rotation in Context**

Partition

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

PARTITION :: BTree  $\rightarrow$  Word  $\rightarrow$  BTree

PARTITION Empty k = Empty

**Rotation in Context** 

### 301 till§

### Balance

Reca

Searchin Trees, BT

Search Tr

Properties Primitives

Rotatio

#### Partition

Simple Approaches

Global

Random Trees PAR

Complex Approache
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  $n \ l \ r) \ k$  | k < Size l = RotateR (Node  $n \ (partition \ l \ k) \ r)$ 

Rotation in Context

### sortin

### Balance

Reca

Searchin

Search Tre

Properties

Rotati

### Partition

Simple Approaches

Global

Random Trees

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

| k < SIZE l = ROTATER (Node n \ (PARTITION l \ k) \ r)
| SIZE l < k = ROTATEL (Node n \ l \ (PARTITION r \ (k - 1 - SIZE l )))
```

Rotation in Context

### 301 1111

### Balance

Reca

Searching Trees, BTr

Properties

Primitives

#### Partition

Simple Approaches

Global

Random Trees

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

Recar

Searcl

Trees, BTI

Search T

Propertie

Primitiv

Rotatio

Partition

Global

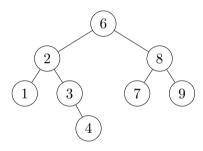
Root Insert

Random Trees

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

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;
```

### Surting

### Balance

Reca

Search

Troos Pl

Search

......

Propertie:

Primitives

Rotatio

#### Partition

Simple App

Global

Root Insert

Random Tree:

Complex App

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;
}
```



#### Global

### Approach #1: Global Rebalancing **Problems**

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



### Sorting

### Balance

....

Searching

Trees, BTr

Properties

Drimitive

Dotatio

Partition

Simple Approa

Global

Root Inser

Complex Approach

Approach #1: Global Rebalancing

• 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.

# Approach #1: Global Rebalancing

**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.



Sorting

Balanced

Docor

C ---------

Searching

Dearen

Propertie

Primitive

Rotatio

Partition

Simple Approache

Global

Root Inse

Random Tree

Complex Approach

Splay

# Global vs Local Rebalancing

### **GLOBAL REBALANCING**

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



# Global vs Local Rebalancing

cs2521@ jashankj@

Balance

Recap

Searchin

Trees, BT

Search

Propertie

Paul on late or

Rotatio

D-----

Cimula Anneantha

Global

Root Inse

Random Tree

Complex Approach

### **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



### cs2521@ iashanki@

### Sorting

Balance

Reca

Searchin

Trees, BT

Search

Properti

Primitiv

Rotat

Simple Approx

Global

Root Insert

Random Tre

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



### Sorting

### Balance

Reca

Searching

Trees, BTr

Search

Properties

Dulm ition

Potatio

Partition

Simple Approa

Global

### Root Insert

Complex Approach

### **Root Insertion**

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.

### **Root Insertion**

cs2521@ jashankj@

Sorting

Balanc

Reca

Searching

Search Tr

Properties

Primitiv

Rotatio

Partition

Simple Appro

Root Insert

Random T

Complex Approache

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!)

C Implementation

### Sorting

```
Balance
Trees
```

Searchi

Trees, BTI

Properties

Primitives

Rotatio

Simple Appr

Root Insert

Random Trees

Complex Approach Splay

```
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;
```



Sorting

Balance

Recar

Searching

Trees, BTr

Search Ti

Properties

Primitive

Rotatio

Partition

Simple Apple

Root Insert

Random Trees

Splay

### **Randomised Insertion**

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.

### 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

### **Splay Trees**

cs2521@ jashankj@

Sorting

Balance

Recai

Searchin

Trees, B1

Search

Properti

Primitiv

Rotati

Cimple App

Global

Poot Insert

Dandon Trees

Compley Appro-

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.

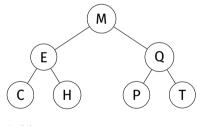
No guarantees, but improved performance.

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

- me, 2016

Splay

#### Four choices to consider for a double-rotation:



1: LL 2: LR

3: RL 4: RR



Splay Rotations
Double-Rotation: Left, Left

ROTATER  $au_{\mathsf{M}}$  ROTATER  $au_{\mathsf{E}}$ 

## Trees

Searching Trees, BT

Properties

Primitives

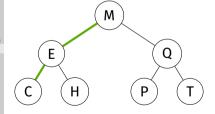
Rotation

Simple Appr

Global Root Insert

Random Trees

Complex Approaches
Splay





Splay Rotations
Double-Rotation: Left, Left

n 1

Balance

Reca

Trees, BTre

Search T

Propertie

Priopertie

Rotatio

Partition

Simple Appro

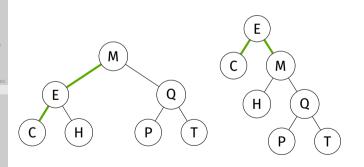
Global

Root Insert

Compley Ann

Splay

#### ROTATER $au_{\mathsf{M}}$ ROTATER $au_{\mathsf{E}}$





# Splay Rotations Double-Rotation: Left, Left

Delene

#### Balance

Reca

Searchin

Search

Search

Propertie

Primitive

Partition

Simple App

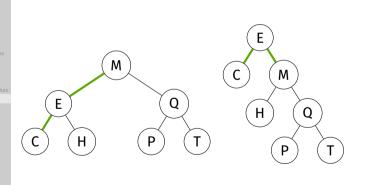
Global

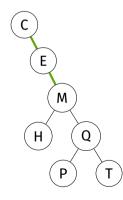
Root Insert

Complex

Splay

### ROTATER $au_{\mathsf{M}}$ ROTATER $au_{\mathsf{E}}$







### Splay Rotations

Double-Rotation: Left, Right

3011111

Balance

Reca

Troos RTr

Search T

Proportio

Propertie

Rotatio

Partitio

Simple Appi

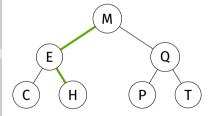
Global

ROOT INSERT

Complex Approach

Splay

### ROTATEL $\tau_{\mathsf{E}}$ ROTATER $\tau_{\mathsf{M}}$





Sorting

Balance

Recap

Searching

Search Tr

Propertie

Primitive

Rotatio

Olmonto Am

Global

Root Insert

Pandom Troos

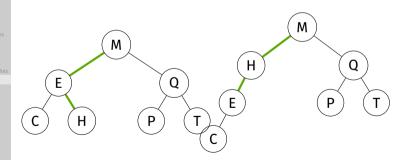
Complex Approa

Splay

## Splay Rotations

Double-Rotation: Left, Right

#### ROTATEL $au_{\mathsf{E}}$ ROTATER $au_{\mathsf{M}}$



COMP2521 19T0 lec12

cs2521@ jashankj@

Sorting

Balance

Rocar

Searchin

Trees, BTr

Search 1

Propertie

rioperue

Dotatio

D-----

Simple Ann

Global

Root Insert

Donalous Toron

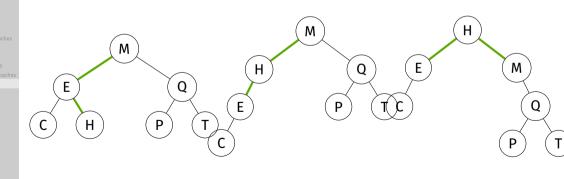
Complex Ar

Splay

### Splay Rotations

Double-Rotation: Left, Right

ROTATEL  $au_{\mathsf{E}}$  ROTATER  $au_{\mathsf{M}}$ 





## Splay Rotations

Double-Rotation: Right, Left

301 (111

Balance

Reca

Searching Trees BT

Search T

Propertie

Polosition

Rotatio

Partition

Simple Appro

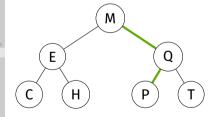
Root Insert

Random Trees

Complex Approache

Splay

### ROTATER $\tau_{\mathsf{Q}}$ ROTATEL $\tau_{\mathsf{M}}$





Sorting

Balance

Docas

Saar

Trees, BT

Search T

Properties

Parimition

Rotatio

Partitio

Jimpte Appir

Global

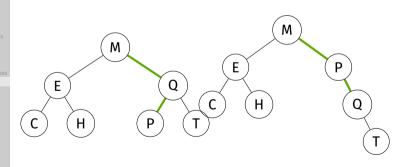
Root misert

Complex Approa

Splay

# Splay Rotations Double-Rotation: Right, Left

ROTATER  $au_0$  ROTATEL  $au_M$ 



COMP2521 19T0 lec12

cs2521@ jashankj@

### Splay Rotations

Double-Rotation: Right, Left

Sorting

Balance

Recap

Searching

Convoluti

Search

Propertie

Drimitivo

Rotation

Partition

Simple App

Global

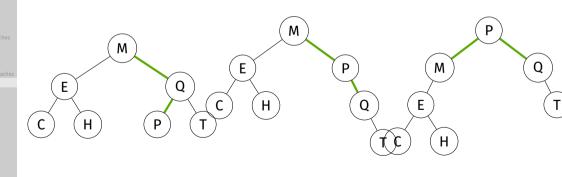
Root Insert

Pandom Troos

Complex Ap

Splay

ROTATER  $au_{\mathsf{Q}}$  ROTATEL  $au_{\mathsf{M}}$ 





Sorting

Balance

Reca

Searching

Search 1

Search

Propertie

. . . . .

D-----

Simple Appro

Global

ROOT INSERT

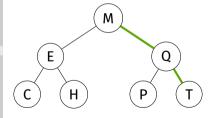
Compley Approache

Splay

### Splay Rotations

Double-Rotation: Right, Right

ROTATEL  $au_{\mathsf{M}}$  ROTATEL  $au_{\mathsf{Q}}$ 

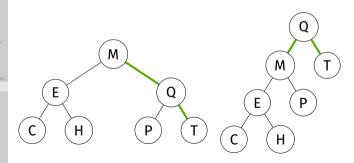




Splay

**Splay Rotations** Double-Rotation: Right, Right

ROTATEL  $au_{\mathsf{M}}$ ROTATEL  $\tau_0$ 



COMP2521 19T0 lec12

cs2521@ jashankj@

### **Splay Rotations**

Double-Rotation: Right, Right

2011111

#### Balance

Reca

Searchin

Carach 1

Search

Propertie

Prioperu

Dotati

Partitio

Simple A

Global

Root Insert

Random I

Complex App Splay ROTATEL  $au_{\mathsf{M}}$  ROTATEL  $au_{\mathsf{Q}}$ 

