

CMPT 280

Topic 16: 2-3 Trees

Mark G. Eramian

University of Saskatchewan

References

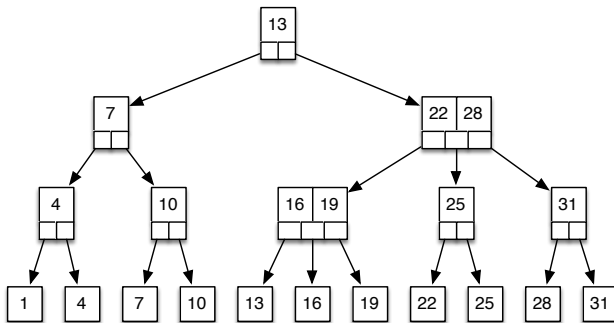
- Textbook, Chapter 16

Review: Properties of 2-3 Trees

- Internal nodes have exactly 2 or 3 children
- Internal nodes contain keys k_1 and k_2 :
 - Elements in left subtree have keys $< k_1$
 - Elements in middle subtree have keys $\geq k_1$ and $< k_2$
 - Elements in right subtree (if it exists) have keys $\geq k_2$.
- Internal nodes do not store elements.
- Leaf nodes contain key-element pairs.
- All leaf nodes are at the same level.
- Above properties result in elements in leaf nodes being in sorted order from left to right.

Exercise 1

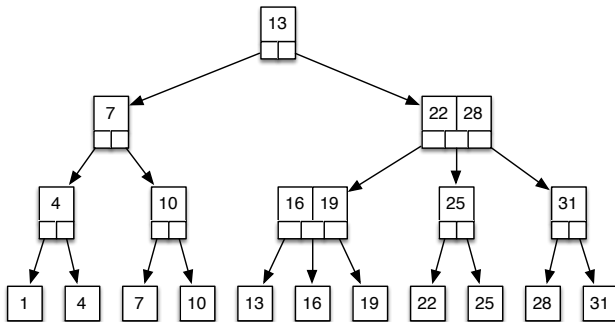
Searching



- Which nodes are visited when searching for key 7?
- What about key 19? 28? 1? 26?

Exercise 2

Insertion



- Starting with the above tree, insert elements with the following keys: 26, 40, 9, 12, 30.
- Repeat the previous exercise, but start with an empty tree.

Exercise 3

- What is the time complexity of 2-3 tree insertion?
 - Time for base case?
 - Time before each recursive call?
 - Time after each recursive call?
 - Number of recursive calls?
 - Added time for special cases?

2-3 Tree Insertion Algorithm

```
1  Algorithm insert(p,i,k):
2  This is the auxiliary recursive algorithm called by the
3  previous insertion algorithm, above.
4  p is the root of the tree into which to insert (k,i)
5  i is the element to be inserted
6  k is the key of the element i
7
8  if the children of p are leaf nodes // base case
9      create new leaf node c containing (k,i)
10     if p has exactly two children
11         make c the appropriate child of p, adjust p.k1, p.k2
12         return null
13     else // p already has 3 children
14         let p1, p2, p3 be the three children of p
15         sort keys of c, p1, p2, p3 in ascending order
16         make smallest two keys the children of p
17         make largest two keys the children of a new internal node q
18         set keys in p and q according to the keys in their middle children
19         ks = third largest key of {c, p1, p2, p3}
20         return (q, ks) // but now q needs a parent.
21                         // attach q to the parent of p as the recursion unwinds
22                         // by returning it.
23
24
25 // continued next slide...
```

Time complexity of base case?

2-3 Tree Insertion Algorithm

```
1  else    // recursive cases
2      if k < p.k1
3          Rs = p.left;
4      else if k < p.k2 or p has only 2 children
5          Rs = p.middle
6      else
7          Rs = p.right
8
9      (n,ks) = insert(Rs, i, k)
10     if n is not null // n is new node resulting from a split, needs a parent.
11         // Make it the child of p
12         if p has exactly two children
13             // This will be one of the case illustrated in Figure 12.5
14             make n the appropriate child of p.
15             update p.k1 and p.k2 appropriately using ks
16             return null
17         else // p already has 3 children, split p, return q
18             // to attach to parent of p
19             // This will be one of the cases illustrated in Figure 12.6
20             // the split() function determine which case, and performs the
21             // adjustments to the tree.
22             (q, ks) = split(p, n, Rs, ks)
23             return (q, ks)
```

Time before recursive call? Time after recursive call? Number of recursive calls?

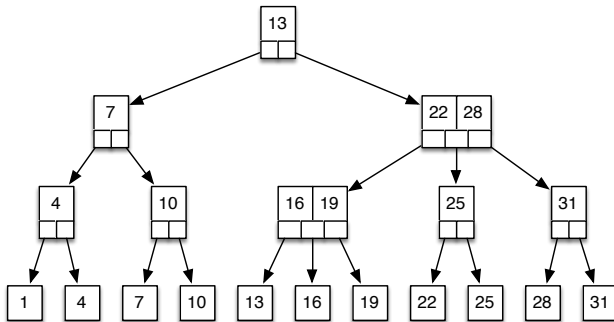
2-3 Tree Insertion Algorithm

```
1  Algorithm insert(i, k)
2  The insertion operation for a 2-3 tree.
3
4  i - element to be inserted
5  k - key of element to be inserted
6
7  if the tree is empty, create a single root (leaf) node containing i
8  else if the tree contains a single leaf node m:
9      create a new leaf node n containing i
10     create a new internal node p with m and n as its children,
11         and with appropriate keys p.k1 and p.k2
12  else
13     call auxiliary method insert(this.root, i, k)
```

How much time is added by checking the special cases?

Deletion from 2-3 Trees

Example



- Let's try to delete 19 from this tree and see what happens.
- Now let's try to delete 25. Uh oh...

Deletion from 2-3 Trees

- Deletion from the 2-3 tree is similar to insertion:
 - Recursively find the leaf node to delete, if found, delete it.
 - As recursion unwinds, if the returned-from node has only one child then fix up the tree as necessary so that it is still a 2-3 tree, possibly leaving the current node with only one child, and proceeding up the tree.
- One special case to consider: when the tree consists of only a leaf node, just delete it and set the root to null.

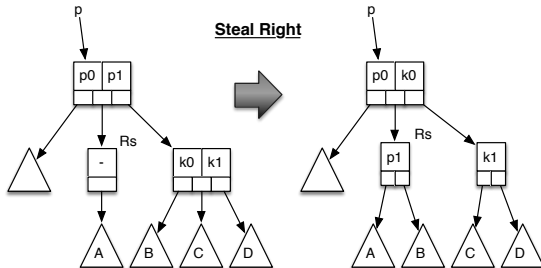
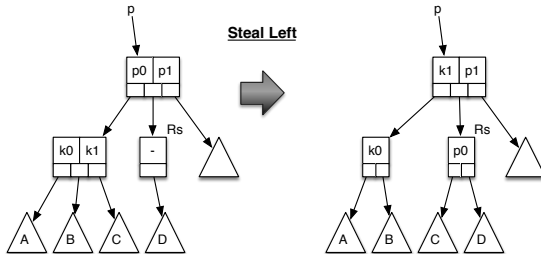
Pseudocode for delete(k)

```
1 delete(k):  
2   if tree is empty  
3       do nothing (or throw exception?)  
4   if tree consists of only a leaf node  
5       destroy it and set the root to null  
6   else  
7       call auxiliary method delete(root, k)  
8       if root of tree has only one child  
9           replace the root with its child
```

Deletion from 2-3 Trees

```
1  Algorithm delete(p,k)
2  p is root of tree from which to delete element w/ key k.
3
4  if children of p are leaf nodes
5      if any child of p matches k, delete it
6      adjust remaining children of p and its keys appropriately
7      (if only one child remains, make it the left child)
8  else
9      // recurse
10     if k < p.k1
11         Rs = p.left
12     else if p has 2 children or k < p.k2
13         Rs = p.middle
14     else
15         Rs = p.right
16     delete(Rs, k);
17
18     if( Rs has only one child )
19         perform first possible of:
20         steal left, steal right, give left, give right
```

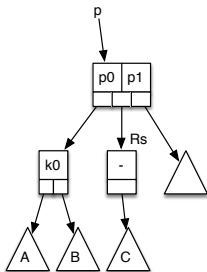
Adjusting keys: Stealing



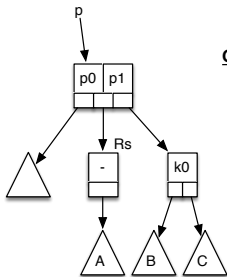
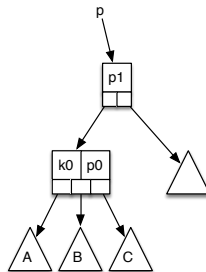
Exercise 4

- What are the other two stealing scenarios?

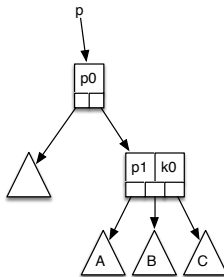
Adjusting keys: Giving



Give Left



Give Right



Exercise 5

- What are the other two giving scenarios?

Exercise 6

- What is the time complexity of 2-3 tree deletion?
 - Time for base case?
 - Time before each recursive call?
 - Time after each recursive call?
 - Number of recursive calls?
 - Added time for special cases?

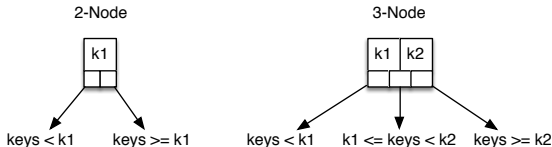
Objects and 2-3 Trees

What objects do we need to implement 2-3 trees?

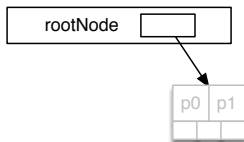
- An object for leaf nodes:

Keyed data item

- An object for interior nodes:



- An object for the tree itself:



Objects and 2-3 Trees

Two kinds of Nodes

- Depending on the tree structure, `rootNode` or `subtree` fields of an internal node may need to refer to either a leaf node or an interior node.
- How do we permit this in Java?

Objects and 2-3 Trees

An abstract class for Nodes

- Define a common ancestor for interior and leaf nodes.
- Use an abstract class that has abstract versions of methods that are invoked by the 2-3 tree class.
- The leaf and interior node classes inherit the common ancestor and implement the abstract methods.
- What should be the type of
 - The `rootNode` field?
 - The subtree fields of interior nodes?

Exercise 7

- Write the necessary classes for implementing nodes in a 2-3 tree.

Next Class

- Next class reading: Chapter 17: B+ trees and B trees.