CMPT 280

Topic 16: 2-3 Trees

Mark G. Eramian

University of Saskatchewan

References

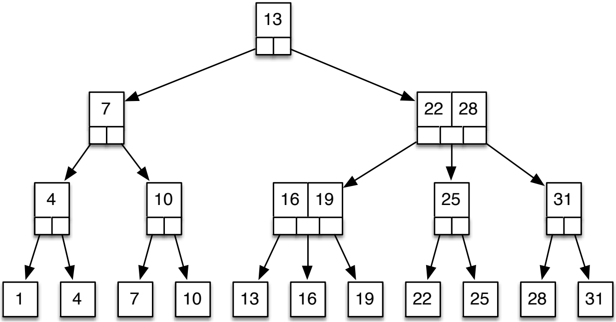
- Textbook, Chapter 16

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 $\geq k1$ 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

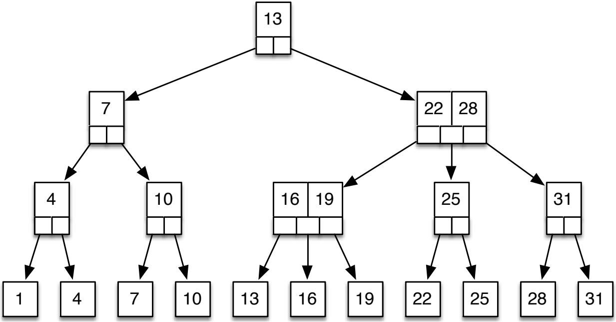
Searching



- 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

```
Algorithm insert(p,i,k):
This is the auxiliary recursive algorithm called by the
previous insertion algorithm, above.
p is the root of the tree into which to insert (k,i)
i is the element to be inserted
k is the key of the element i

if the children of p are leaf nodes  // base case
    create new leaf node c containing (k,i)
    if p has exactly two children
        make c the appropriate child of p, adjust p.k1, p.k2
        return null
    else // p already has 3 children
        let p1, p2, p3 be the three children of p
        sort keys of c, p1, p2, p3 in ascending order
        make smallest two keys the children of p
        make largest two keys the children of a new internal node q
        set keys in p and q according to the keys in their middle children
        ks = third largest key of {c, p1, p2, p3}
        return (q, ks)  // but now q needs a parent.
                       // attach q to the parent of p as the recursion unwinds
                       // by returning it.


// continued next slide...
```

Time complexity of base case?

2-3 Tree Insertion Algorithm

```
else    // recursive cases
    if k < p.k1
     Rs = p.left;
    else if k < p.k2 or p has only 2 children
        Rs = p.middle
    else
        Rs = p.right

    (n,ks) = insert(Rs, i, k)
    if n is not null // n is new node resulting from a split, needs a parent.
                     // Make it the child of p
        if p has exactly two children
            // This will be one of the case illustrated in Figure 12.5
            make n the appropriate child of p.
            update p.k1 and p.k2 appropriately using ks
            return null
        else // p already has 3 children, split p, return q
             // to attach to parent of p
             // This will be one of the cases illustrated in Figure 12.6
             // the split() function determine which case, and performs the
             // adjustments to the tree.
            (q, ks) = split(p, n, Rs, ks)
            return (q, ks)
```

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

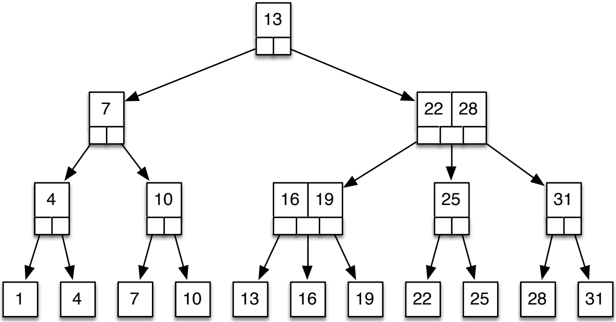
2-3 Tree Insertion Algorithm

```
Algorithm insert(i, k)
The insertion operation for a 2-3 tree.

i - element to be inserted
k - key of element to be inserted

if the tree is empty, create a single root (leaf) node containing i
else if the tree contains a single leaf node m:
    create a new leaf node n containing i
    create a new internal node p with m and n as its children,
        and with appropriate keys p.k1 and p.k2
else
    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.

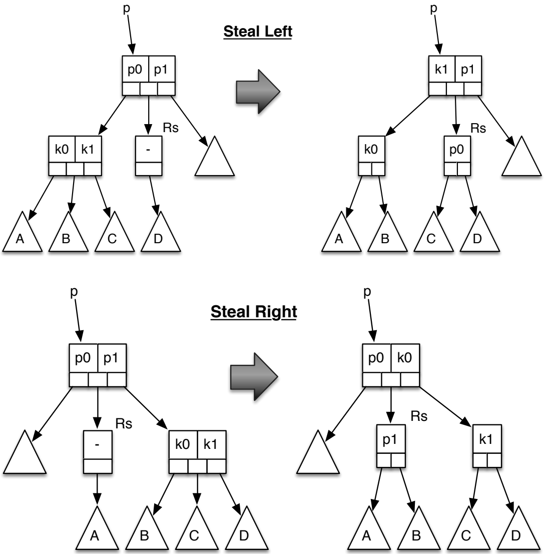
Pseudocode for delete(k)

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

Deletion from 2-3 Trees

```
Algorithm delete(p,k)
p is root of tree from which to delete element w/ key k.

if children of p are leaf nodes
    if any child of p matches k, delete it
    adjust remaining children of p and its keys appropriately
    (if only one child remains, make it the left child)
else
    // recurse
    if k < p.k1
        Rs = p.left
    else if p has 2 children or k < p.k2
         Rs = p.middle
    else
        Rs = p.right
     delete(Rs, k);

     if( Rs has only one child )
         perform first possible of:
         steal left, steal right, give left, give right
```

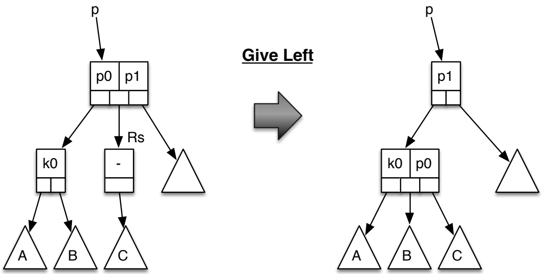
Adjusting keys: Stealing

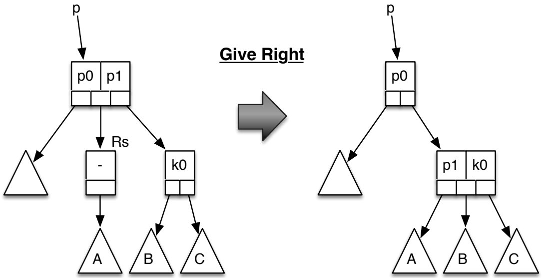


Exercise 4

- What are the other two stealing scenarios?

Adjusting keys: Giving





Exercise 5

- What are the other two giving scenarios?

Exercise 6

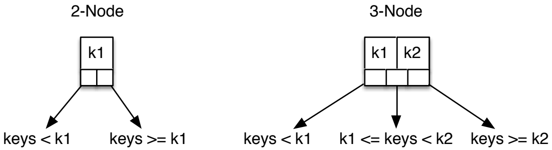
- What is the time complexity of 2-3 tree deletion?
  - Time for base case?
  - Time before each recursive call?
  - Time after each recursive call?
  - Number of recursive calls?
  - Added time for special cases?

Objects and 2-3 Trees

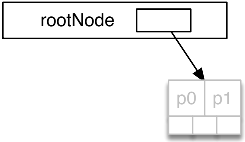
What objects do we need to implement 2-3 trees?

- An object for leaf nodes:

- An object for interior nodes:



- An object for the tree itself:



- 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 an 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.