CIS 313

week of Nov 5

seventh week of the term
red-black trees

1. every node is either red or black
2. the root is black
3. every leaf (null) is black
4. if a node is red, both of its children are black
5. for each node, all simple paths from the node to descendant leaves contain the same number of black nodes
example insertions

after insertion of 1,2,3,4,5,6 into empty RB tree

let’s continue with 7, 8, ...
insert 7

single left rotate
insert 8

- Single left rotate (at 2)
- Color shift
insert 9
insert 10
insert 10 (cont’d)

note: 4 as root gets colored black at the end
RB Deletion

- much harder
- idea: start as with BST
- try to delete a leaf
- if it is not a leaf, find successor
- if it has only one child, then easy
  - it’s a B node with R child
  - replace B value with R value, remove the R node
- main problem now: delete a leaf
RB Deletion (cont’d)

• to delete a leaf x
• if x is red, easy: simply remove it (no change to black-height)
• if x is black, remove it also but …
• that path is now missing a black node
• view x as having an EXTRA BLACK
• the routine RB-DELETE-FIXUP removes the extra black by pushing it up the tree
Figure 13.7 from text

Case 1

Case 2

Case 3

Case 4

\text{new } x = \text{root}[T]
example: delete 4

replace 4 with its successor 5, then remove 5’s node and place “extra black” there
delete 4 (cont’d)

case 1 (rotate)
delete 4 (cont’d)

case 2 (adjust color)

x is now red and we place the extra black there by changing its color
delete 4 (done)

NEXT: delete 1 from this
deleting 1 from previous

case 2 (adjust color)
delete 1 (cont’d)

done since x is at root (we’ve reduced the black height of the tree)
remove 2

NEXT: delete 3 from this
removing 3 from previous
remove 3 (cont’d)

set up as case 4:

motivated by a “transfer” in 2-3-4 tree

A
x
B
C
D
w
E

case 4 (rotate and done)
follow template from text

A
B
C
D
E

8
6
5

9
10

8
6
5

9
10
remember: all cases come with mirror image

• here x is right child of parent
• the left child of w is red
• fix-up can be completed with a right rotation
• note that the blue nodes (B and C) can be either red or black
B-trees

• very important data structure in computer science
• database indexing, hard disk referencing, MongoDB, ...
• balanced, multi-way search tree
• many slight variations, we will use definition in CLRS text
• idea is that nodes are large and fit into a disk block (minimum amount of data that’s pulled off a hard drive)
• node size parameters (here called t) depend on disk speeds, block sizes, etc.
B-tree specifications

• fixed parameter \( t \), called *minimum degree*
• nodes have between \( t-1 \) and \( 2t-1 \) keys
• so therefore they have between \( t \) and \( 2t \) children
• root is exception: it may have as few as 1 key (2 children)
• all null pointers have the same depth (distance from root)
• a 2-3-4 tree is a B-tree with minimum degree \( t=2 \)
different texts: things to look for

• top-down versus bottom-up insertion
  • CLRS does top-down, split full nodes during search
  • unlike how it does RB trees
  • bottom-up more common in practice, less wasted space

• ties to left or right
  • no duplicates here
  • need to know for B+ trees, which have all keys at an additional “leaf level”

• left/right bias: if middle key not well defined (when splitting a node with even number of keys)