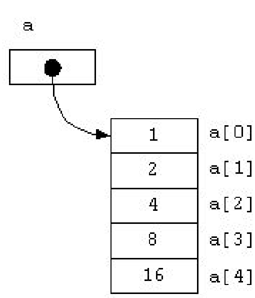
self-balancing trees

colin shaw
05.12.2016

1. **introduction**
    a. welcome to computer science club
    b. personal introduction
    c. thank you to revunit for sponsoring
    d. what is it about
        i. promoting the community
        ii. demonstrating competencies
        iii. practice presenting

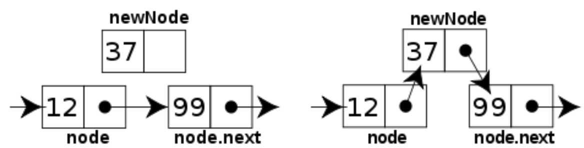
2. **what are we doing here**
    a. fundamental concepts
        i. ordered structures
        ii. ordered tree are an important extension to ordered lists
            1. somewhat similar form
            2. potential guarantees of O(log n) rather than O(n)
        iii. relationship to expressions (aside)
            1. diagram a brief expression tree
            2. much different than imperative computation
            3. expressions have no side effects
            4. show ability to parallelize tasks
            5. example of scene graph and nVidia parallelization (deep learning)
        iv. graphs are like trees but more general (cycles, etc.)
    b. how to best choose the right data structure for your problem

3. **arrays**
    a. lecture
        i. diagram



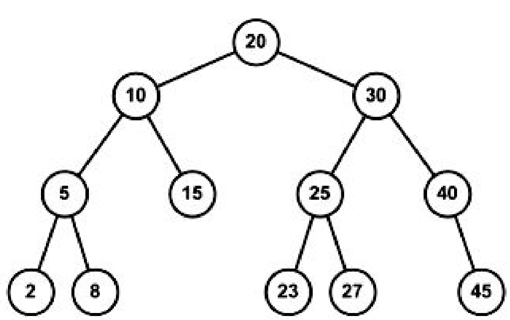
ii. pros
    1. fast (can be aligned for best cache use, etc.)
    2. easily reasoned indexing
iii. cons
    1. relatively inflexible (can be hard to change problem size, etc.)
    2. cannot be implemented as expressions (side effects)

4. **linked lists**
    a. linked_list.ml
    b. lecture
        i. diagram



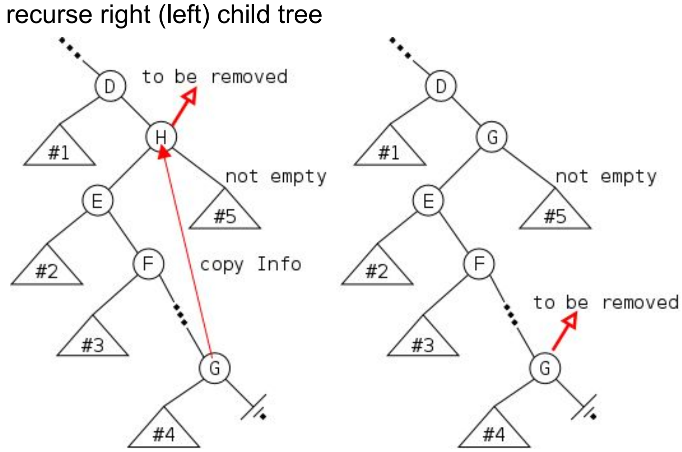
        ii. pros
            1. easy to change size
            2. evaluates as expression
        iii. cons
            1. linear performance
            2. more challenging reasoning
    c. code
        i. example is a bit different than the built-in list
        ii. comparison evaluation (will mentioning this again)
    d. performance characteristics - O(n)

5. **ordered binary trees**
    a. ordered_binary_tree.ml
    b. type difference compared to linked lists
        i. value and two pointers vs. value and one pointer
        ii. could actually be doubly-linked list or graph as well
    c. lecture
        i. diagram



        ii. pro - major breakthrough in potential speed
        iii. con - not guaranteed performance

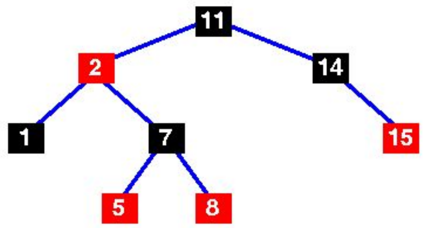
d. code
    i. not much different than linked list, with caveats
        1. more comparisons yield more cases (linked list vs. tree)
            a. insert
            b. member
        2. delete is more complicated than expected
    ii. delete cases
        1. case 1 - two empty children
            a. replace node with empty
        2. case 2 - one empty child
            a. replace node with child node
        3. case 3 - two children
            a. depends on ordering convention
            b. find smallest element in right child tree (or largest of left child)
            c. replace current value with smallest (or largest) value
            d. recurse right (left) child tree



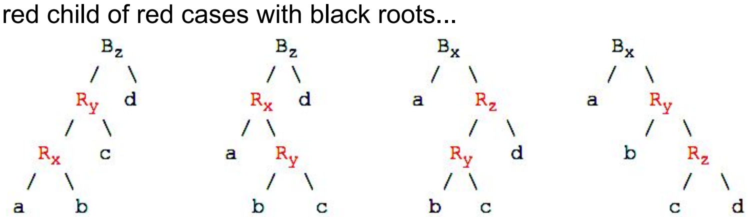
6. red black trees

a. red_black_tree.ml
b. red black trees or one of several balancing techniques for binary trees
    i. avl trees
    ii. aa trees
    iii. different invariants solving similar problem, same complexity class
c. many other tree types that are not binary but solve similar problem
    i. 2-3 trees
    ii. splay trees
    iii. b trees
d. why the name “red black” tree

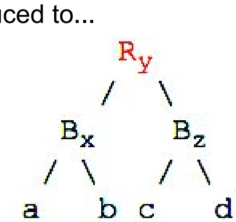
e. lecture
    i. Diagram



    ii. invariants
        1. all paths have same number of black nodes
        2. red nodes do not have red children
        3. root node is black
        4. empty child pointers must have color black
        5. observation
            a. consider an all black perfectly balanced tree
            b. consider alternating black and red perfectly balanced tree
            c. compare size of these structures
            d. red nodes are “slack” nodes and carry rebalancing information
    iii. pros
        1. guarantee of computational complexity
        2. guarantee of structural bounds
    iv. cons
        1. additional data required to track state in the red black tree type
        2. higher complexity code
f. code
    i. membership is still straightforward like ordered binary tree
    ii. insert
        1. similar to ordered tree except with restructuring
        2. cases
            a. restructure any insert recursively
            b. red child of red cases with black roots...

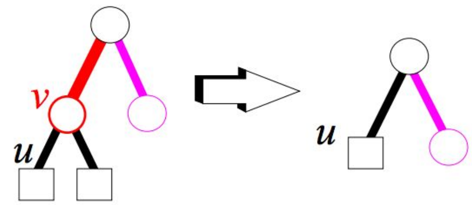


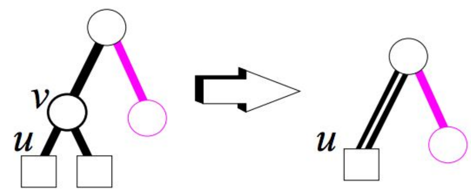
...can all be reduced to...



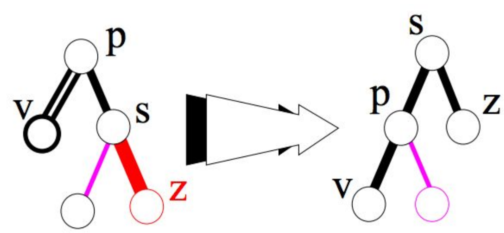
… and the root node dealt with as a special case

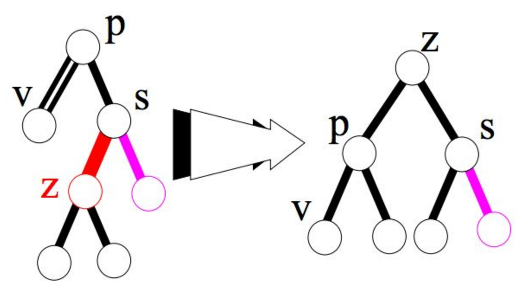
iii. delete
   1. must be able to propagate information if invariants break
      a. similar to ordered tree delete, but with color cases as well
      b. applied recursively
      c. deletion
         i. double-black nodes carry structural information





      d. restructuring
         i. Double-black converted to proper structure





      e. there are actually numerous cases to consider!
iv. optimizations
   1. split left and right balance cases
   2. minimize constructors

7. **applications and variations on red black trees**
    a. direct applications
        i. priority queues (task schedulers)
        ii. dictionary
    b. tree dictionary
        i. red_black_dict.ml
        ii. just make a key-value type
    c. typical implementation in the real world
        i. modules
        ii. functors
            1. types
            2. comparison function