CS5112: Algorithms and Data Structures for Applications

Balanced search trees

Ramin Zabih

Some pictures from Wikipedia

Cornell University

CORNELL TECH

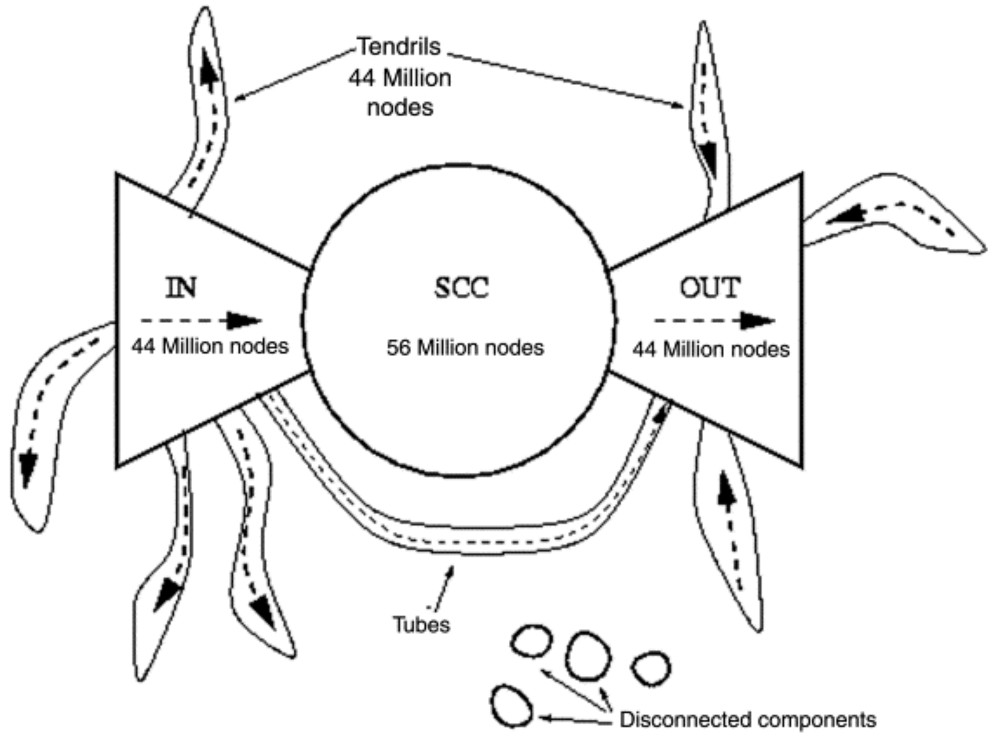
Administrivia

- Prelim (midterm) date: Wednesday October 16
- Web site is: https://cornelltech.github.io/CS5112-F19/
  - Now has graders, tentative syllabus
  - Office hours are up, locations coming shortly
- Quiz #2 will be out Thursday, due in 24 hours
- We are looking into getting lectures recorded
- Digression = fun material you are not responsible for

Lecture Outline

- Digression: the web as a graph
- Data structure invariants
- Binary search trees
- Asymptotic complexity and balancing
- Red-black trees

Digression: web as a graph



Structure	Altavista, May 1999
Total	203.5 million
SCC	56.5 million
IN	43.3 million
OUT	43.1 million
Tendrils	43.7 million
Links	1466 million (7.2 per page)

- See: https://kharshit.github.io/blog/2017/09/08/structure-of-the-web

Data structure invariants and debugging

- An invariant is a property of a program/data structure that must always hold for it to be correct
- Good programmers check invariants frequently!
  - Tools like assert()
    - E.g., in `sqrt(float x)` you assert `x >= 0`
  - Optimized out in production code
- Key to debugging is finding errors as early as possible
  - Kind of like a crime scene

assert() is your friend

- See: Writing Solid Code (from Microsoft!)
- Good discussion here of its importance

  “How many times have you written the assumptions your function makes in a comment at the top of your code? How many times have you traced bugs down deep into the bowels of a codebase only to find that somebody (maybe you!) disregarded the assumptions outlined in the comment? If you … employ the use of assert statements to enforce the assumptions made in your code, you can rest assured that you’ll be notified if any of those assumptions are ever ignored again.”

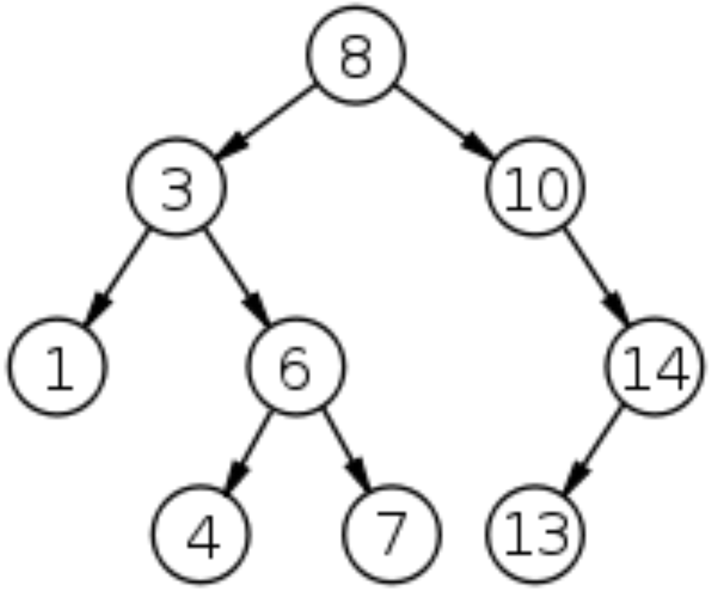
Example data structure invariants

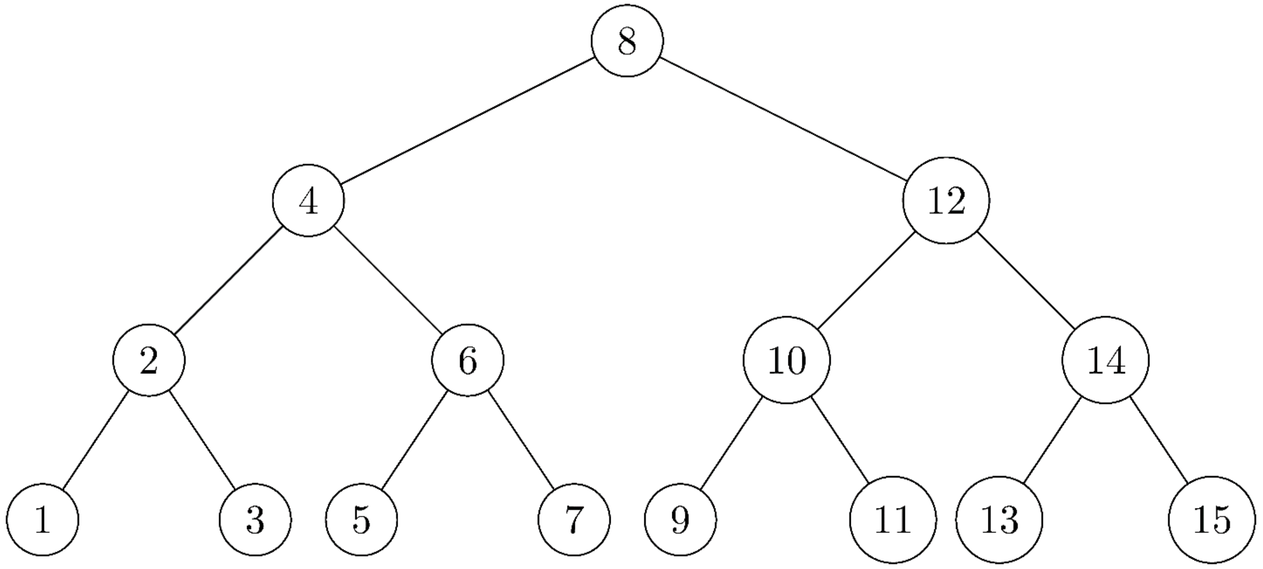
- In the union find array, a node’s predecessor is itself iff it is the root of a tree
- In a sorted linked list, each element is greater than its predecessor
- Lots of other examples
  - See: https://fgiesen.wordpress.com/2010/09/27/data-structures-and-invariants/ for a detailed example in C

Binary search trees & their invariants

- A binary search tree (BST) is a fast dictionary data structure
- Nodes have values (data) associated with them
  - Unlike in the previous lecture, where the number was just an index for a particular node in the tree
- INVARIANT: If a node contains value v then the left subtree has value $< v$ and the right subtree has values $> v$

Binary tree examples





Complete binary tree of height 3

Running time

- The worst case is actually fairly common
  - Add elements in sorted order: 1, 2, 3, 4, 5, 6, 7
  - Tree has height $O(n)$ with a single long branch
- A binary tree of depth d can have up to $O\left(2^d\right)$ elements in it
  - Maximum achieved when it's completely balanced
- So, a balanced tree with n elements in it has depth $O(\log n)$
  - Which is vastly better!
- How do we make a tree balanced?

CORNELL TECH

Self-balancing trees

- Several data structures balance binary trees
  - Normal BST's, but when you insert an element you occasionally 'rotate' the tree to keep it roughly balanced
- The more balanced the tree is, the closer to $O(\log n)$ speed
- However, if you want the tree to be really balanced the rotations get quite complicated
- Typical data structures:
  - Red-black trees (we will go into these in detail)
  - AVL trees (we won't)

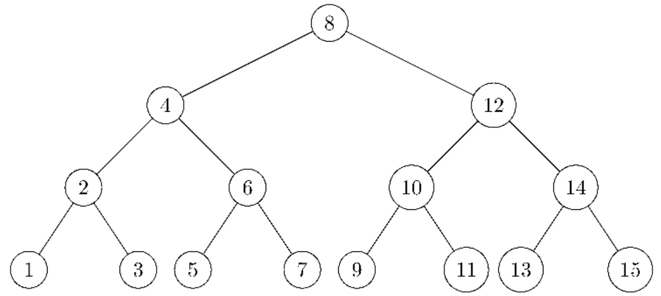
BST invariants

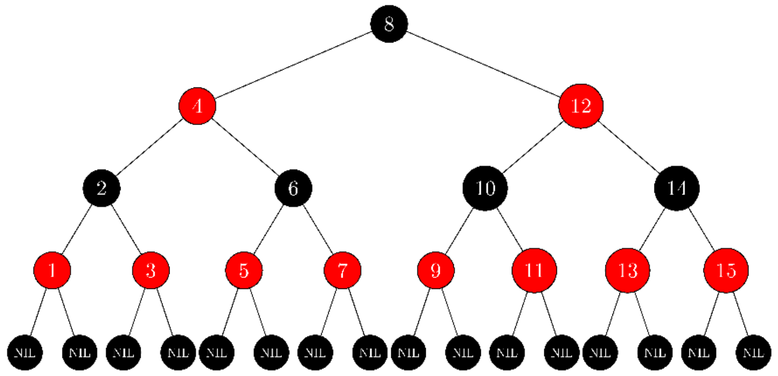
- AVL invariant:
  - For any node, the height of its left and right subtree differ by ≤ 1
  - If you insert an element that would break this, the tree rotates
  - LOTS of bookkeeping to make this work
- Red-black tree invariant:
  - The path from the root to the farthest leaf is no more than twice the path from the root to the nearest leaf
  - Much simpler, but still not trivial

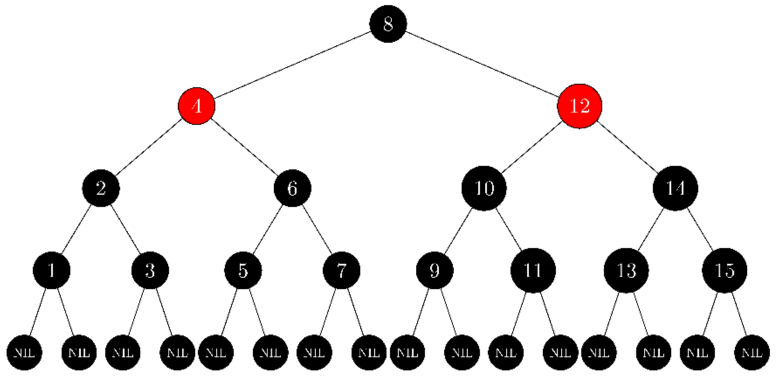
Red-black tree invariants

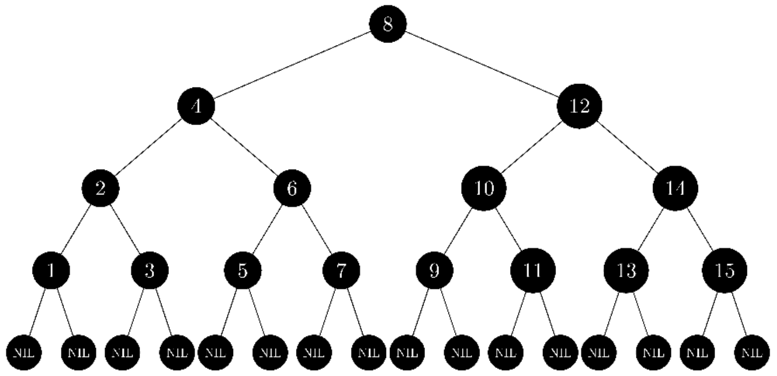
- P1: Every node is either red or black
- [Convention] Root is black
- P2: There are no two adjacent red nodes along any path
- P3: Every path from the root to a leaf has the same number of black nodes
  - This is the **black height** of the tree
- If a tree satisfies these conditions so does every subtree
  - if not, the whole tree would not

Black height = 2, 3, 4









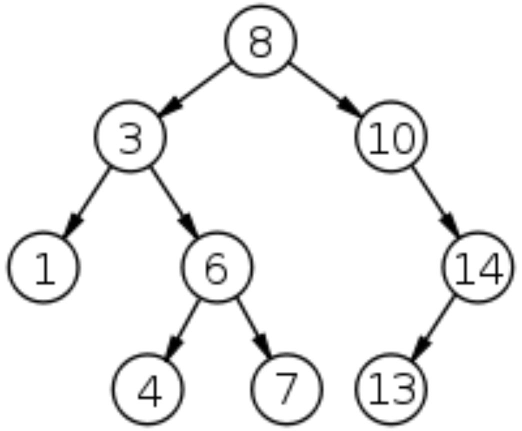
Observation: red nodes are optional!

Red-black tree consequences

- Subtlety: you can satisfy all of these invariants except P3 (= black height), and still have a wildly unbalanced tree
  - Just make it purely from black nodes!
- With all these properties, the path from the root to the farthest leaf is no more than twice the path to the nearest. Proof?
  - P3: both paths have the same number of black nodes
  - P2: there are never two adjacent red nodes
  - So the longest path can at most insert a red node after each black one

Red-black tree operations

- Lookup is easy
  - Just like a standard BST, ignore the node colors
- What about insert?
  - In a standard BST we do a lookup, which shows where the new value should go
  - Example: insert in this tree:
  - Note that we can pick a new root
    - This is basically a rotation!



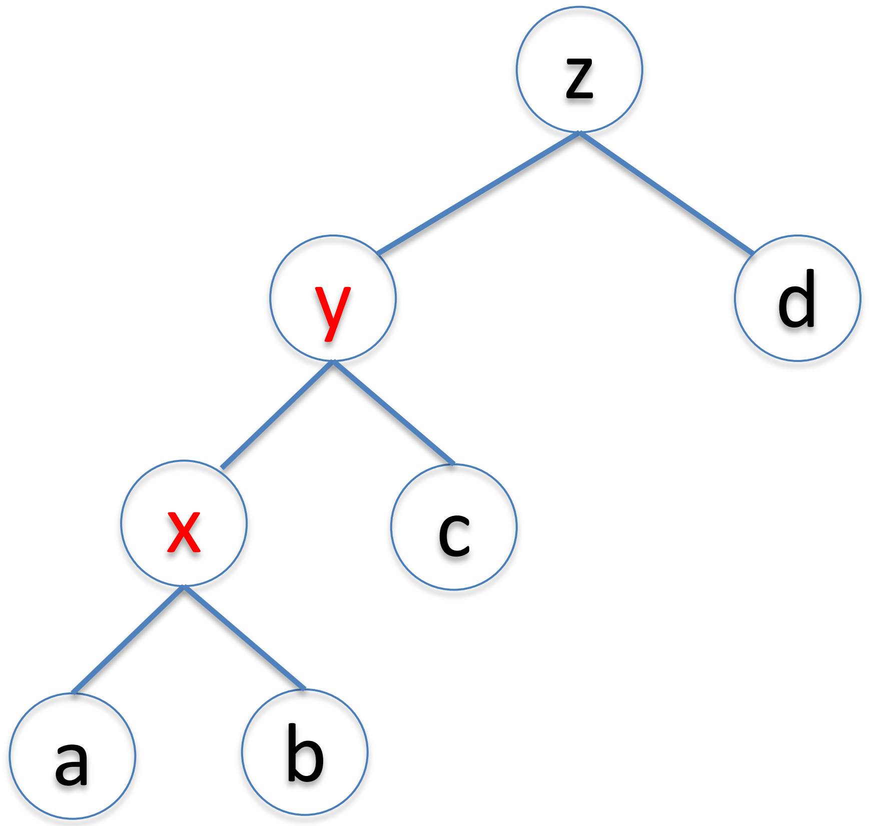
Red-black insert and rotations

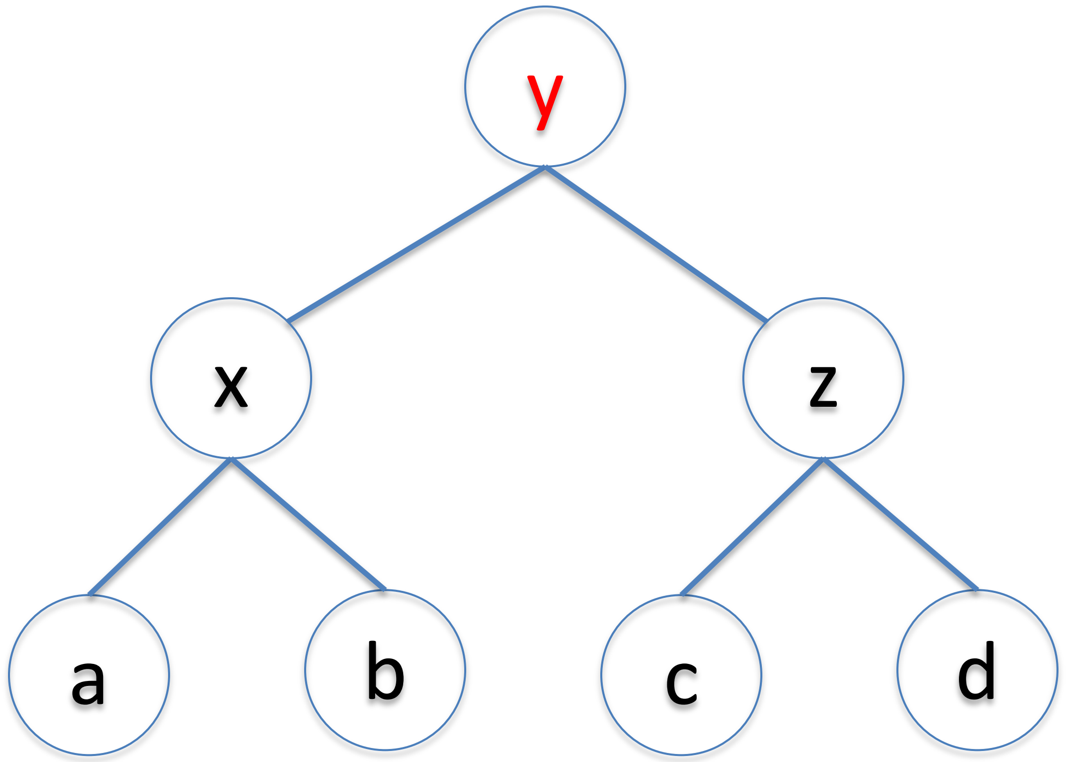
- We do the same thing but we color the node red to preserve P3 (black height of tree is unchanged)
- But we have now broken P2 if the parent was red
- So we need to patch up the tree by tree rotation
  - We must be able to increase black height (why?)
- We not only need to consider the red parent
  - But also its parent (new node’s grandparent)
  - Which is black (why?)

There are four cases to consider

- We can consider them all at once by symmetry

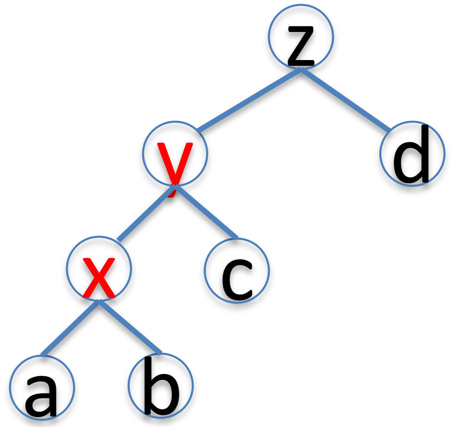
$$a < x < b < y < c < z < d$$

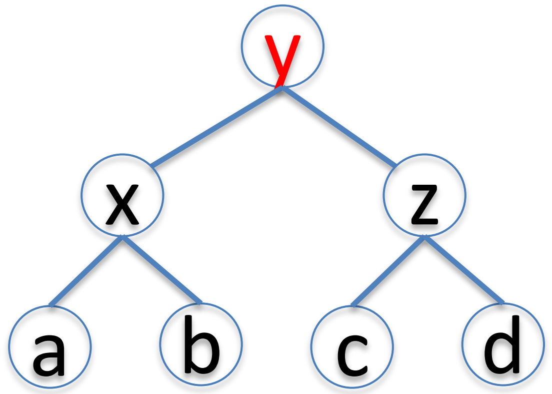




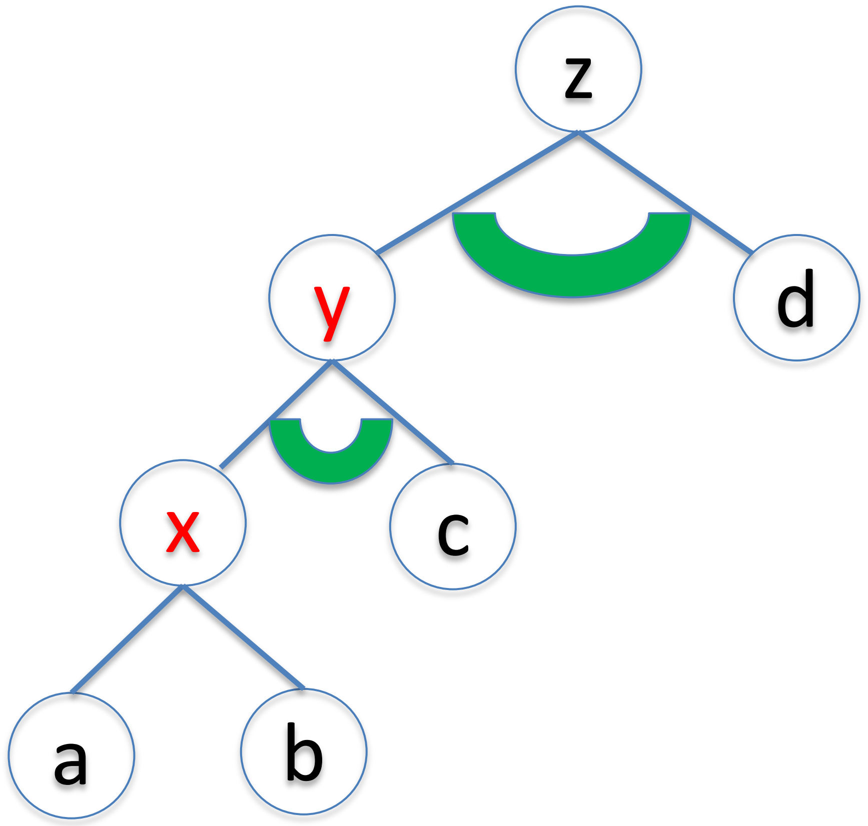
Why is this rotation valid?

- $a < x < b < y < c < z < d$
- 1. Promote y to root
- 2. Move c below left from z
  - Why?
  - $y < c, c < z$
- 3. Color x black
- 4. If y is the root of the whole tree color it black





Red-black tree four symmetries



AVL trees are much more painful

- Invariant is the balance factor at each node, height difference between left and right subtrees. Must be $\{0, \pm 1\}$