Binary Search Trees

Dr. Baldassano

chrisb@princeton.edu

Yu's Elite Education

Last week recap

- Associative arrays and sets
- Hash functions
  - Minimize collisions, spread items evenly
- Dealing with collisions
  - Open addressing vs chaining
- Cryptographic hashes (e.g. passwords)
- Locality-sensitive hashing for nearest neighbors

Last week's assignment: Anagrams

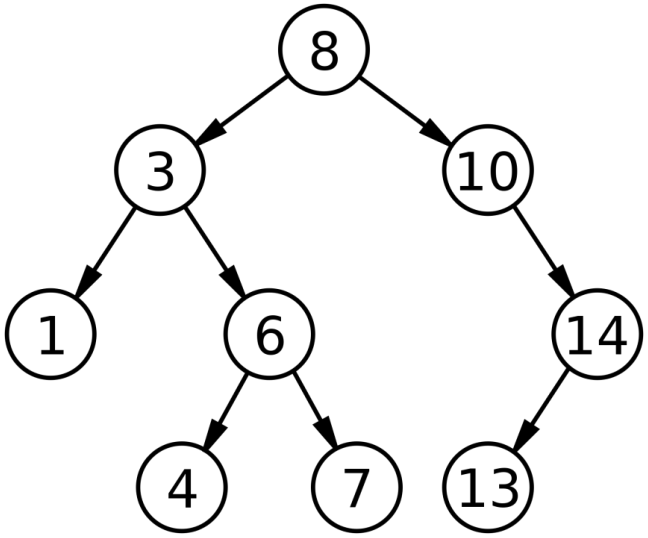
- Given a dictionary http://www.codeabbey.com/data/words.txt divide it into groups of anagrams
- Example: tea, asleep, plus, ate, please
  - [tea, ate], [asleep,please], [plus]

Maintaining a sorted list

- Sorted lists make many things easier
- What if we want to add/remove items from a sorted list?
  - Option 1: Resort list each time
    - Expensive, O(N log N) each time
  - Option 2: Insert into array
    - Easy to find insertion point O(log N), hard to insert O(N)
  - Option 3: Insert into linked list
    - Hard to find insertion point O(N), easy to insert O(1)

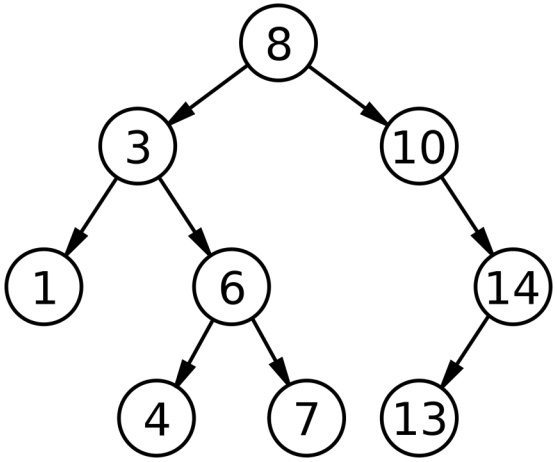
Binary search tree

- Solve both problems with new kind of linked data structure: binary search tree
- BST property: every node is greater than its left child, less than its right child



Finding element in BST

- Start at root (top)
- If root too big, go left, else go right
- Iterate until finding element or end of tree



- Big O?

Inserting element in BST

- Exactly the same as finding, except once we get to end of tree we add the new node
- https://www.cs.usfca.edu/~galles/visualization/BST.html
- Big O?

Log(N) operations in BST

- Find and insert
- Minimum element
  - How to find?
- Maximum element
  - How to find?
- Remove (we'll come back to this)

- So this is like a sorted list or array, but without any O(N) operations

Traversals of BST

- How to print every node of a BST?
- Recursive solution:

```
Visit(Node):
    print(Node)
    Visit(Left child)
    Visit(Right child)
```

Types of Traversals

Visit(Node):

    # **Pre-order:** print(Node)

    Visit(Left child)

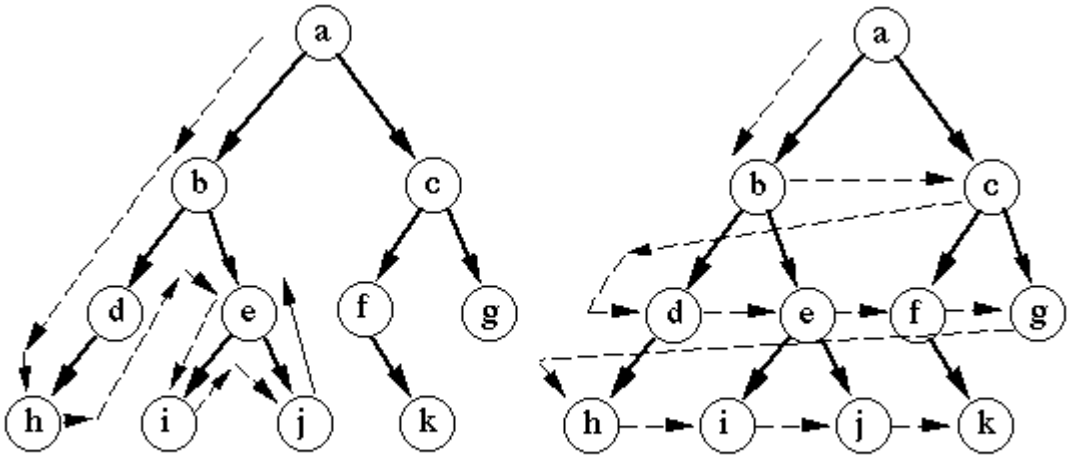
    # **In-order:** print(Node)

    Visit(Right child)

    # **Post-order:** print(Node)

Breadth-first traversal

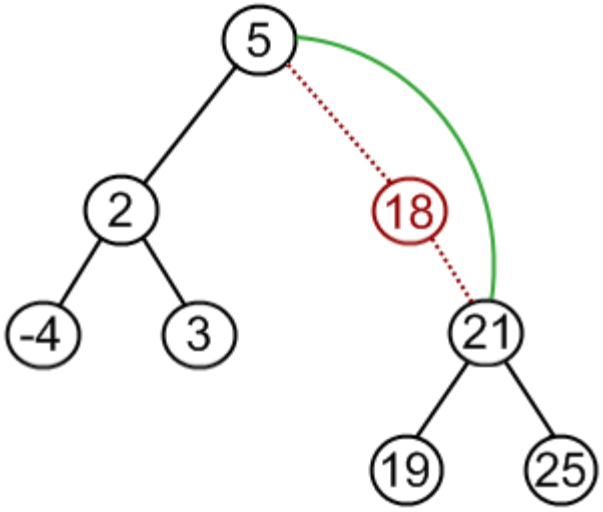
- Pre/In/Post order all depth-first – immediately go down to deepest nodes
- Breadth-first: rather than using a recursion stack, use a queue



Python Example: Insert and In-order traversal

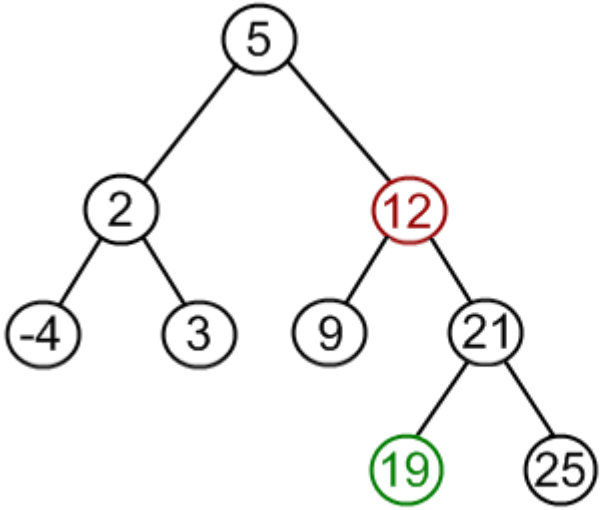
Removing element in BST

- Trickiest operation – need to make sure that BST property still holds
- If node has no children, just delete it!
- If node has only one child, connect node's parent to child



Removing with two children

- Idea: swap node with next-highest node



- Next-highest node is minimum of right subtree

Removing with two children

- Practice
- http://visualgo.net/bst.html

Problem: Check BST

- Given a binary tree, check to see if it is a BST

Problem: sorted array to BST

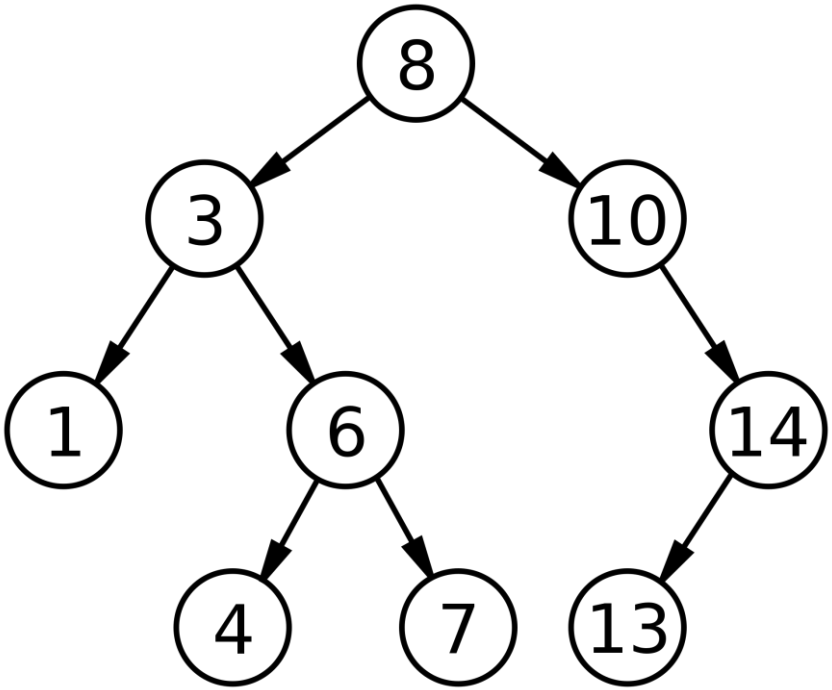
- Given a sorted array, can we build a *balanced* BST?

Problem: sorted LL to BST

- Given a sorted linked list, can we build a BST by just changing links?

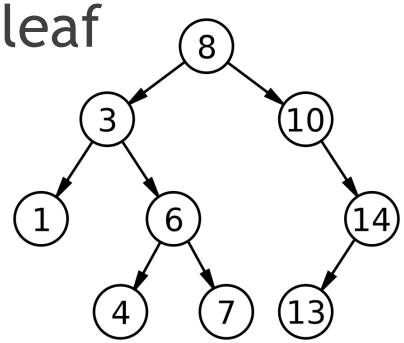
Problem: Lowest Common Ancestor

- Given two nodes, find their first ancestor



Homework: Maximum depth of BST

- Calculate the maximum depth of a BST - the longest path from the root to a leaf
- For example, max depth = 3



- This is often an important thing to keep track of – if max depth is too high, BST is turning into a list