Data Structures/Algorithms Wrap Up

Download Demo Code

Goals

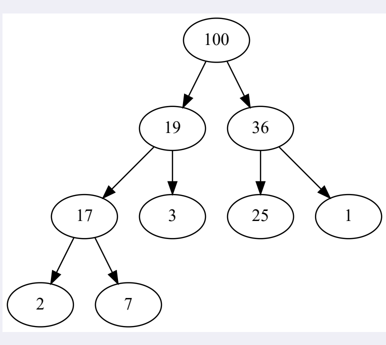
- Overview of how heaps work
- Lightly introduce other DSAs
- Overview what is most important to study, and how to do so

Heap

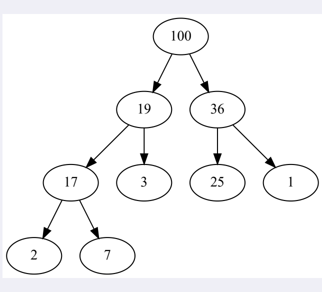
- Often used to implement priority queues
- Often used as part of a larger algorithm/data structure

Max Heap

Each parent must be greater than children



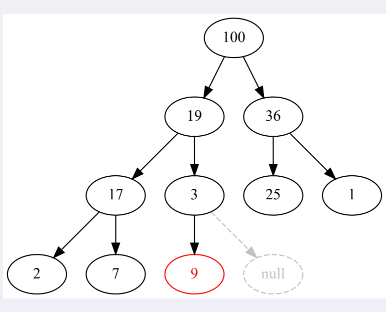
We'll use a "complete max binary heap"



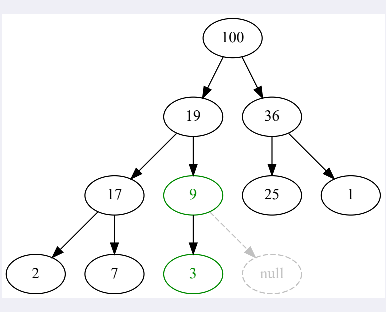
- **complete**: filled top → bottom, left → right
- **max**: each parent is greater than children
- **binary**: parent can have at most two children
- **heap**: tree with rule between parent/children

Adding Item

Add at bottom right

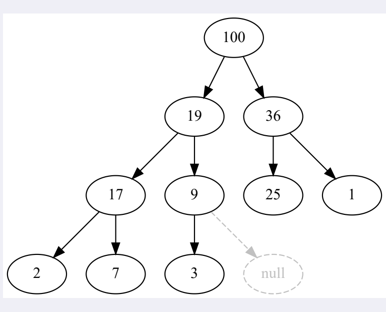


Swap upward until correct



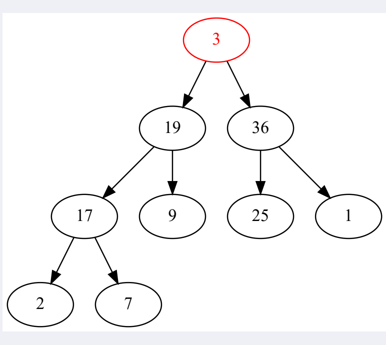
Removing Top Item

Highest-priority item is top of tree!

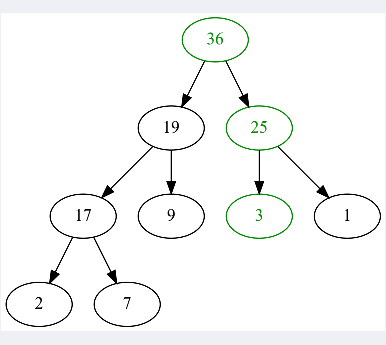


If we just remove it, our tree won't have a head

Put bottom right node at top



Swap downward until correct



(Swapped with 36, not 19, as 19 < 36)

Runtime

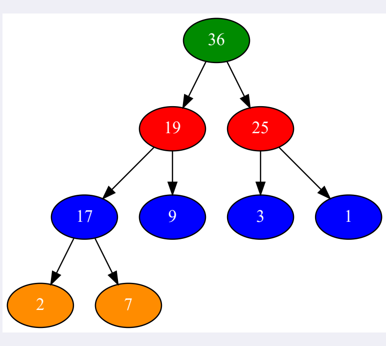
Adding to bottom right is O(1)

Swapping top & bottom right is O(1)

The swapping up & down limits the runtime

- **bubbleUp:** O(log n) *(max # swap up = height)*
- **sinkDown:** O(log n) *(max # swap down = height)*

Implementation

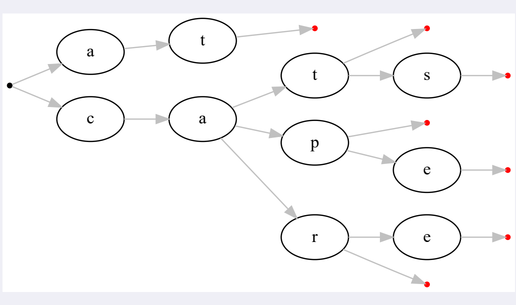


- Can represent as an array

```
[36, 19, 25, 17, 9, 3, 1, 2, 7]
[#0,  1,  2,  3, 4, 5, 6, 7, 8]
```

- Can easily find *i*'s children:
  - Left: `2 * i + 1`
  - Right: `2 * i + 2`

Tries



(stores *at*, *cat*, *cats*, *cap*, *cape*, *car*, *care*)

- Often called "prefix trees" *(pronounced like "trees" or "tries")*
- Often used for searching for words, autocorrect, or autocomplete

Typical Methods

Method names vary across implementations, but one set:

addWord(str)
Add a word to the trie

removeWord(str)
Remove a word from the trie

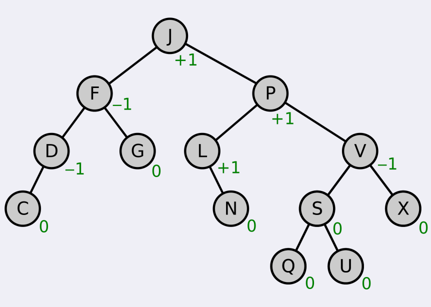
hasWord(str)
See if a Trie has a word

autoComplete(str)
Given a string, return a list of all words

Self Balancing BSTs

- Binary search trees can suffer from being imbalanced
- BSTs with internal algorithms for self-balancing:
  - AVL *(easier, faster)*
  - Red Black *(more complex, ends up more balanced)*
  - B-Tree *(more complex, offers other features for large data)*

For example, AVL Trees keep track in tree of current "balanced-ness":



- Each node is scored based on the height of its right subtree compared to the height of the left subtree
- In an AVL tree, this score must be -1, 0, or 1.

Bloom Filter

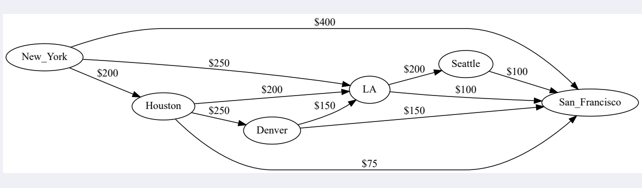
- A data structure that uses a hash function to store values
- Quickly answer questions like "Is particular item in a collection?"
  - Very rarely, it will say *Yes* when the answer is actually *No*
  - It will never say *No*, when the answer is actually *Yes*
- The more space, the fewer false positives

- Useful when you have tolerance for false negatives but not false positives
  - Caching
  - Recommendation Engines (has this person seen a particular article?)
  - Test if a URL has been visited for privacy / security concerns

Path Finding

Algorithms for finding the most efficient path in weighted graphs:

- Dijkstra's algorithm
- A*



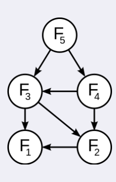
Dynamic Programming

Some problems have *overlapping subproblems* (it's easier to solve them once, rather than keep solving again and again)

Dynamic programming is a mix of recursive thinking and memoization

Classic example is "Fibonacci sequence":

- Starting with 1 and 1
- Next number is sum of two previous numbers
- `1 1 2 3 5 8 13 21 ...`



What Do You Need To Know?

The "Must Knows"

- Knowing when to use arrays, objects, LLs, trees, and graphs
- Big O notation & runtime of common operations in these
- Identifying recursion
- Traversal In LLs, trees, and graphs via BFS and DFS
- Searching and traversing of binary search trees
- How hashing/hash tables work

What Are You Likely To Be Asked About?

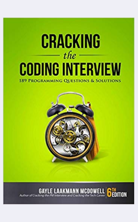
- Selecting a data structure
- Big O Notation
- Algorithm design

Should You Study This?

- It depends
  - On your developer goals
  - On your interest/aptitude for it
- Focus on the *must knows*

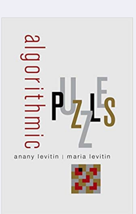
Resources

Books



Cracking the Coding Interview

- Overview of common DSAs
- Challenges with hints & solutions
- Excellent, well-organized
- Code is in Java, but very readable



Algorithmic Puzzles

- Logic problems for comp scientists
- Code-free! *(focusing on thinking)*
- Helps learn to think algorithmically

Online

- Rithm-developed Algorithms & Data Structures Masterclass
  - JavaScript-focused overview of this material
- Interviewing.io
  - Watch recorded interviews that focus on DSAs
- Problem Solving with Algorithms and Data Structures using Python
  - Free online textbook with interactive exercises
  - Written in Python, but ideas are same ini JS

Practice Online

General challenges:

- LeetCode
- HackerRank
- Codewars

Deeper dives:

- Interview Cake

Practice Interviewing:

- Pramp