Announcements

- Homework 3 is out. Due 3/9
- Midterm review March 9th
- Midterm Exam March 11th
- Manu is stepping down as a TA
- New office hour Priyamvad Tuesday 3-5 PM
Review

- HW1 solutions
- Splay Trees
  - Move accessed node to root
    - Zig-Zag: double rotate (a la AVL)
    - Zig-Zig: reverse order
- Prefix Trees (tries)
Today’s Plan

- Note about HW1 Problem 5
- Visualization of Splay Trees
- Cover Tries at normal pace
- Introduction to Priority Queues
Amortized Running Time

- In classical analysis, we try to prove:

\[ MT(N) = O(M \log N) \]

- If this is impossible, we can guarantee that \( M \) operations take:

\[ \sum_{i=1}^{M} T_i(N) = O(M \log N) \]
Prefix Trees (Tries)

* Nicknamed “Trie”, short for retrieval
* Efficiently store objects for fast retrieval via keys
  * Usually key is a String
* Basic strategy:
  * split into sub-tries based on current letter
Trie Example

“cat”, “cow”, “dog”, “doberman”, “duck”
Trie Details

- Not all words are at leaves
  - cat, cataclysm, cataclysmic
- Initially, one letter is enough to uniquely identify
- When a new word is inserted that conflicts, need to branch
  - Originally-unique word must be moved to lower level
Trie Analysis

- In the worst case, inserting a key of length $k$ or (looking up) is $O(k)$
- This is not dependent on $N!$ (surprise, not factorial)
- Much better than $\log(N)$ for huge data like dictionaries
- Sometimes we can access words even faster.
  - E.g., we can find qwerty uniquely with just “qw”
Priority Queues

- New abstract data type Priority Queue:
  - Insert: add node with key
  - deleteMin: delete the node with smallest key
  - (increase/decrease priority)
Heap Implementation

- Priority queues are most commonly implemented using Binary Heaps
- Binary tree with special properties
- Heap Structure Property: all nodes are full, (except possibly one at the bottom level)
- Heap Order Property: any node is smaller than its children
Array Implementation

- A full tree is regular: we can easily store in an array
  - Root at $A[1]$
  - Node $i$ has children at $2i$ and $(2i+1)$
  - Parent at $\text{floor}(i/2)$
- No links necessary, so faster (in most languages)
Insert

- To insert key $X$, create a hole in bottom level
- **Percolate up**
  - Is hole’s parent is less than $X$
    - If so, put $X$ in hole, heap order satisfied
    - If not, swap hole and parent and repeat
DeleteMin

- Save root node, and delete, creating a hole
- Take the last element in the heap X
- **Percolate down:**
  - Check if X is less than hole’s children
    - if so, we’re done
    - if not, swap hole and smallest child and repeat
Assignments

- Start/continue HW3
- Read Weiss Section 6.1-6.3