2020-03-26 Priority Queues

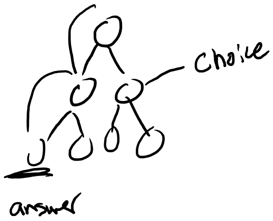
Thursday, March 26, 2020 8:45 AM

Online Resources

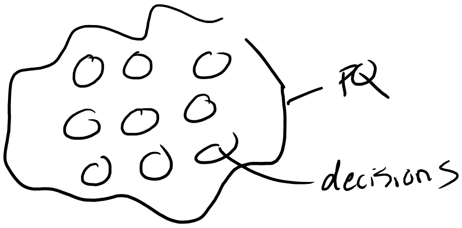
- Heap visualizer: https://www.cs.usfca.edu/~galles/visualization/Heap.html

General Overview

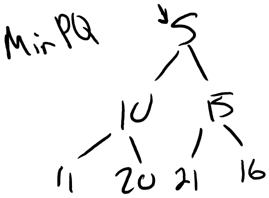
- Priority queues are great for making decisions when given several viable alternatives.
- Contrast to decision trees
  - Decision trees make either/or choices.

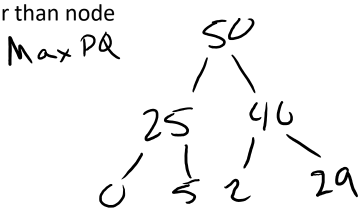


- A decision tree is a good choice when particular outcomes are impossible given a set of circumstances.
  - E.g. If it is not raining, we never need to bring an umbrella
- Priority queues also make decisions, but they always consider **all possible** choices.



- In a PQ, items come out based on a preconstructed "priority"
  - In contrast to all other data structures where items "come out" based on insert sequence (vector, LL, stack, queue) or traversal pattern (trees).
- In a PQ, the rule is that all nodes must be less important than that node
  - MinPQ: All things must be larger than node
  - MaxPQ: All things must be smaller than node





Observations from student reflections

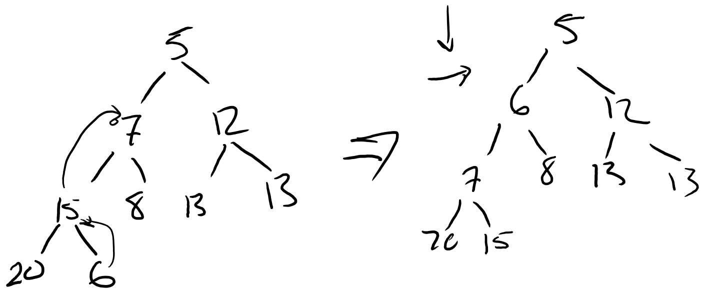
- What is the free store (also sometimes called a heap)?
  - Free store (a.k.a. heap) is where all of our magic unlimited memory comes from. It has nothing to do with the heap data structure.

Priority Queue Properties

Student questions…

- What is a complete binary tree?
  - A complete tree is one in which all nodes from levels 0…(n-1) are full
    - Full: node has zero or two children
  - At level N, nodes are filled in from left to right
  - (See 01 - intro to trees.pptx in repository)
- Why is insertion time LogN? / Where do you insert new values into a priority queue?
  - New values in a PQ are always inserted such that the tree maintains its completeness property.
  - After insertion, the new value "bubbles" to its final resting place

- New values in a PQ are always inserted such that the tree maintains its completeness property.
- After insertion, the new value "bubbles" to its final resting place



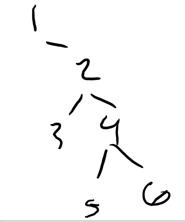
- For algorithm analysis, we ask how big is our tree and how many comparisons were required in order to get the new value into its final resting place.
  - Tree side: 9 size 2 comparisons (3 in worst case)
  - 2^3 = 8

Array-based representation:

5	6	12	7	8	13	13	20	15
0	1	2	3	4	5	6	7	8

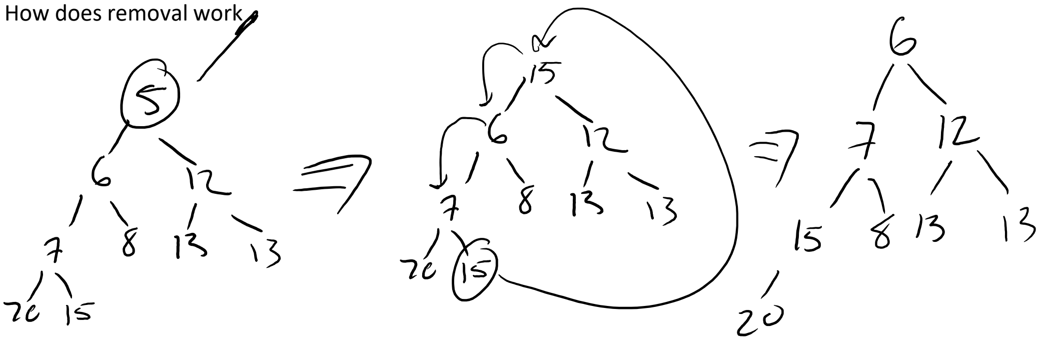
- Rather than using pointers to find children, we use math (which ends up being faster)
- LeftChild = 2*index + 1
- RightChild = 2*index + 2
- Parent = Floor(index / 2)

- Array vs LinkedList-based trees - -what's the difference?
  - As we saw above, one version uses pointers and the other arithmetic to find children.
  - Array based versions use less memory because they don't need to track pointers
  - Array-based trees only work well when the tree is near complete. Consider the following tree:



1	NULL	2	NULL	NULL	3	4	NULL	NULL	NULL	NULL	NULL	NUL	NULL	5	6
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

- What does a PQ of more complex data types look like?
  - See accompanying C# code
- How does removal work?


5
6
12
7
8
13
13
20
15
15
6
12
7
8
13
13
20
15
6
7
12
15
8
13
13
20