Heaps

Dr. Baldassano

Yu's Elite Education

Last week recap

- Dynamic Programming: strategy for creating an algorithm when a problem:
  - Can be broken into optimal subproblems
  - Subproblems are non-overlapping
- Moovies assignment

Today: we meet our first data structure

- A data structure has two key features:
  - What kind of data it holds
  - What kinds of operations it can do quickly
- Example: an ordered list and an unordered list both store numbers
  - Unordered list: appending is fast, search is slow
  - Ordered list: appending is slow, search is fast

Binary Heap

- A binary heap is a data structure that holds numbers called keys
  - Keys might be part of a larger data element
- Supports the following operations:

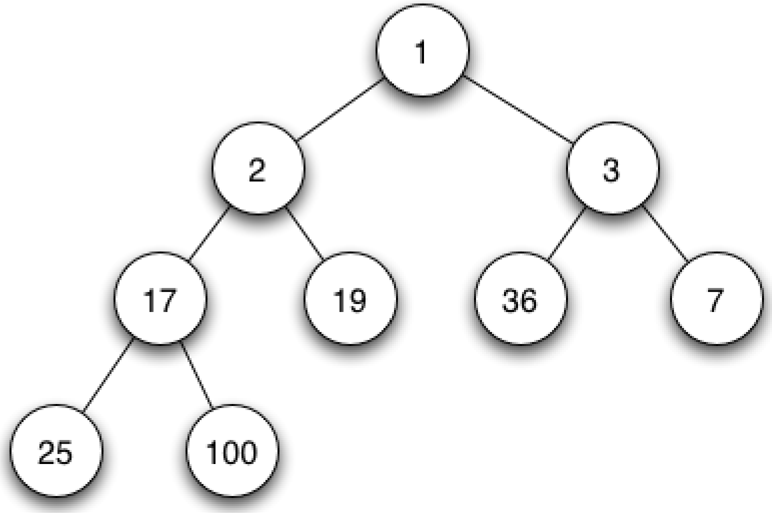
Operation	Time complexity
Find minimum	O(1)
Delete minimum	O(log n)
Insert	O(log n)
Decrease key	O(log n)

Uses of heaps

- Sorting: insert all elements, then keep removing minimum
- As a priority queue: keeping track of the "highest priority" item to process next

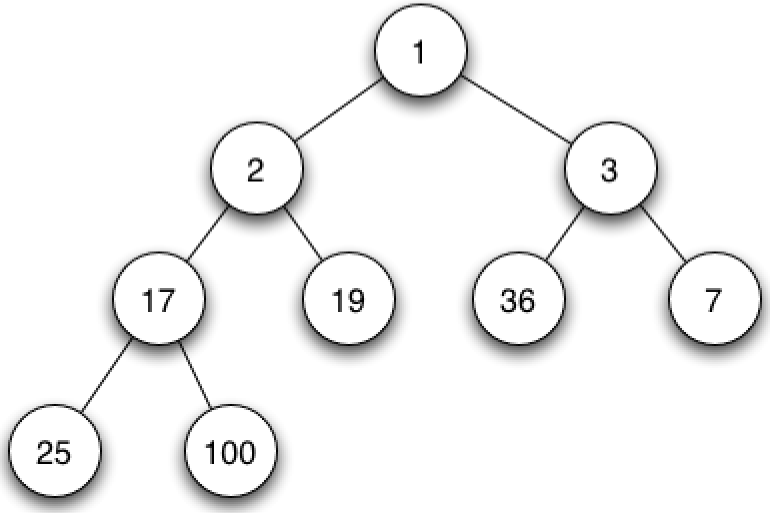
Implementing a heap

- Represent heap as a *complete* binary tree, with all children keys greater than their parent
- Note: no ordering among siblings/cousins



Find minimum

- Easy! Minimum is always at the top, just return it in O(1)



Insert key

- Add new element to next position in complete array
- Swap child with parent until the heap ordering is fixed
- Takes O(levels of tree) = O(log N)

Decrease key

- Similar to insertion: swap child with parent until heap is ordered
- O(log N)

Delete minimum

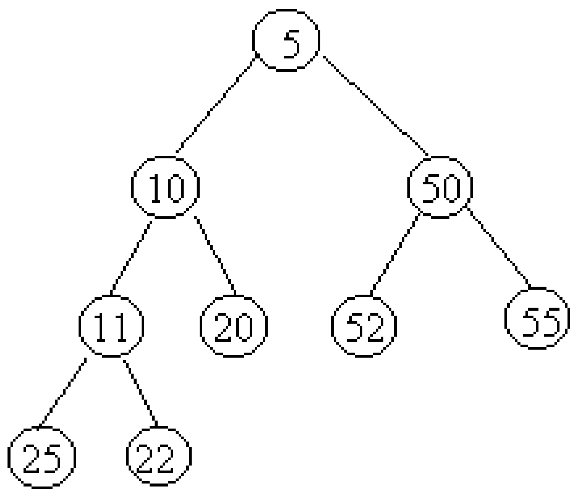
- Swap minimum with rightmost leaf and delete
- Bubble root down, promoting smaller child
- Again O(log N)

Building a heap from scratch

- We would approximate that inserting N elements should take O(N log N) time
- But a more careful analysis shows that most elements in a heap are near the bottom, so they don't take many swaps
- Can actually build heap in O(N) time

Actually implementing a heap

- Often store values in array, calculate links



5	10	50	11	20	52	55	25	22
0	1	2	3	4	5	6	7	8

Priority Queue example 1

- We have k sorted arrays of size n each
  - Might have broken up a sorting task across multiple machines in a datacenter
- Merge them into a single sorted array
- O (n*k log k)

Priority Queue example 2

- Given unsorted array, find the k minimum elements
  - Might want to get the 10 best scores from a very large database
  - Or k closest points to some position
- O(n + k log n)

File compression

- Say we are given some text data to store
- If there are 32 letters + punctuation possibilities, we could represent each letter as a 5-bit binary codeword ($2^5 = 32$)
- Better idea: Use shorter codewords for frequent letters, longer codewords for infrequent letters

Huffman encoding

- Count frequencies of each symbol
- Create tree merging least-frequent symbols
- Repeat until all symbols merged
- Path to a symbol is its codeword

- This is a prefix code – don’t need explicit separators, since no codeword is prefix of another

Implementing with min heap

- Create heap in O(N) time
- Remove two smallest elements and re-insert sum of frequencies, until only one left
- O(N log N) in total

Heapsort

- Build heap, then remove minimum N times
- O(N log N), so asymptotically optimal
- https://www.cs.usfca.edu/~galles/visualization/HeapSort.html

Assignment: Near-sorted data

- Given an array for size n that is mostly sorted: each element is at most k places away from its correct position
- How can we sort this array efficiently using a heap?
- What is the Big-O time complexity in terms of n and k?