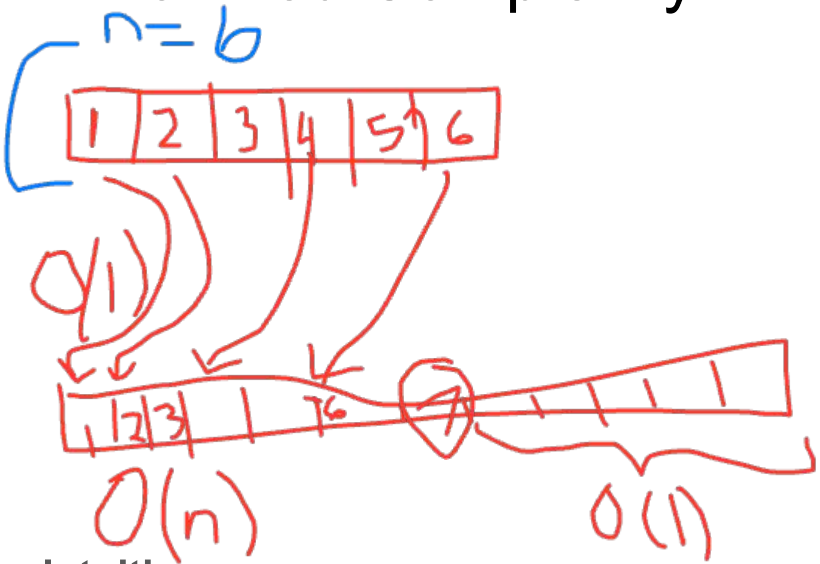
CSE 332
Data Structures & Parallelism

Priority Queues

Melissa Winstanley
Spring 2024

Amortized Complexity



- “Cost” of first n inserts is $1 per insert
- “Cost” of the (n+1)th insert is $(n+1)
- How many “cheap” ($1) inserts can we do before we encounter another “expensive” insert?

Intuition:

(n * $1) + (1 * $(n+1)) + ((n-1) * $1) = **$(3n)** - cost of entire sequence of 2n inserts

$3n / 2n = $3/2 = O(1) average cost per operation given this sequence

Chapter 11 - a lot more complicated to “prove” that this

Today & Next Time - Priority Queues / Heaps

- What is a Priority Queue?
- Introduction to the heap
- Heap operations
- Heap implementation
- Building a heap

Scenario

What is the difference between waiting for service at a pharmacy versus an ER?

- Pharmacies usually follow the rule: First Come, First Served
- Emergency Rooms assign priorities based on each individual's need

Scenario

What is the difference between waiting for service at a pharmacy versus an ER?

- Pharmacies usually follow the rule: First Come, First Served

  Queue (FIFO)

- Emergency Rooms assign priorities based on each individual's need

  Priority Queue

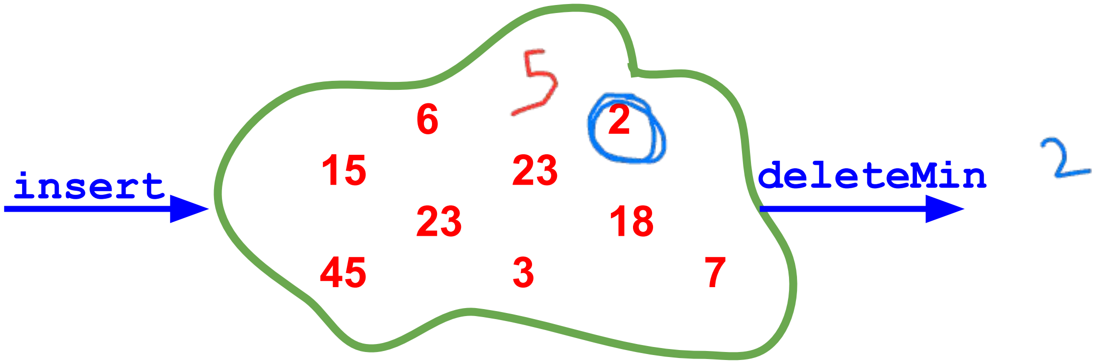
A new ADT: Priority Queue

- Textbook Chapter 6
    - We will go back to binary search trees (ch4) and hash tables (ch5) later
    - Nice to see a new and surprising data structure first
- A **priority queue** holds *compare-able* data
    - Unlike stacks and queues need to compare items
    - Given **x** and **y**, is **x** less than, equal to, or greater than **y**
    - What this means can depend on your data
    - Much of course will require comparable data: e.g. sorting
- Integers are comparable, so will use them in examples
    - But the priority queue ADT is much more general
    - Typically two fields, the *priority* and the *data*

Priority Queue ADT

- Assume each item has a “priority”
  - The lesser item is the one with the greater priority
  - So “priority 1” is more important than “priority 4”
  - Just a convention, could also do a maximum priority

- Main Operations:
  - `insert(int)`
  - `deleteMin()`



- Key property: **`deleteMin`** returns and deletes from the queue the item with greatest priority (lowest priority value)
  - Can resolve ties arbitrarily

Aside: ints as data *and* priority

- For simplicity in lecture, we'll often suppose items are just ints and the int is also the priority
- So an operation sequence could be

```
insert 6
insert 5
x = deleteMin // Now x = 5
```

- `int` priorities are common, but really just need comparable
- Not having "other data" is very rare
  - Example: print job has a priority and the file to print is the data

Priority Queue Example

insert a with priority **5**
insert b with priority **3**
insert c with priority **4**
w = **deleteMin**
x = **deleteMin**
insert d with priority **2**
insert e with priority **6**
y = **deleteMin**
z = **deleteMin**

after execution:

w = b
x = c
y = d
z = a

To simplify our examples, we will just use the priority values from now on
Analogy: insert is like enqueue, deleteMin is like dequeue
But the whole point is to use priorities instead of FIFO

Priority Queue Example

insert a with priority **5**
insert b with priority **3**
insert c with priority **4**
w = **deleteMin**
x = **deleteMin**
insert d with priority **2**
insert e with priority **6**
y = **deleteMin**
z = **deleteMin**

after execution:

w = b
x = c
y = d
z = a

To simplify our examples, we will just use the priority values from now on
Analogy: insert is like enqueue, deleteMin is like dequeue
But the whole point is to use priorities instead of FIFO

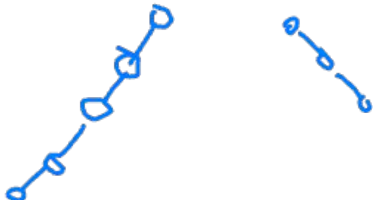
Applications of Priority Queues

- Like all good ADTs, the priority queue arises often
    - Sometimes “directly”, sometimes less obvious
- Run multiple programs in the operating system
    - “critical” before “interactive” before “compute-intensive”
    - Maybe let users set priority level
- Treat hospital patients in order of severity (or triage)
- Select print jobs in order of decreasing length?
- Forward network packets in order of urgency
- Select most frequent symbols for data compression (cf. CSE123)
- Sort: insert all, then repeatedly deleteMin

Preliminary Implementations of Priority Queue ADT

	insert	deleteMin
Unsorted Array	1	n
Unsorted Linked List	1	n
Sorted Circular Array	n	1
Sorted Linked List	n	1
Binary Search Tree (BST)	n	n

Note: Worst case, assume arrays have enough space



Preliminary Implementations of Priority Queue ADT

	insert	deleteMin
Unsorted Array	θ(1)	θ(n)
Unsorted Linked List	θ(1)	θ(n)
Sorted Circular Array	θ(n)	θ(1)
Sorted Linked List	θ(n)	θ(1)
Binary Search Tree (BST)	θ(n)	θ(n)

Note: Worst case, assume arrays have enough space

Today - Priority Queues / Heaps

- What is a Priority Queue?
- Introduction to the heap
- Heap operations
- Heap implementation
- Building a heap

Our Data Structure: The Heap

Or more specifically, a "binary min heap"

- Worst case: O(log n) for insert
- Worst case: O(log n) for deleteMin
- If items arrive in random order, then the average-case of insert is O(1)
- Very good constant factors

Key idea: Only pay for functionality needed

- We need something better than scanning unsorted items
- But we do not need to maintain a full sorted list

- We will visualize our heap as a tree, so we need to review some tree terminology

Reviewing Some Tree Terminology

root(**T**): A

leaves(**T**):

children(**B**):

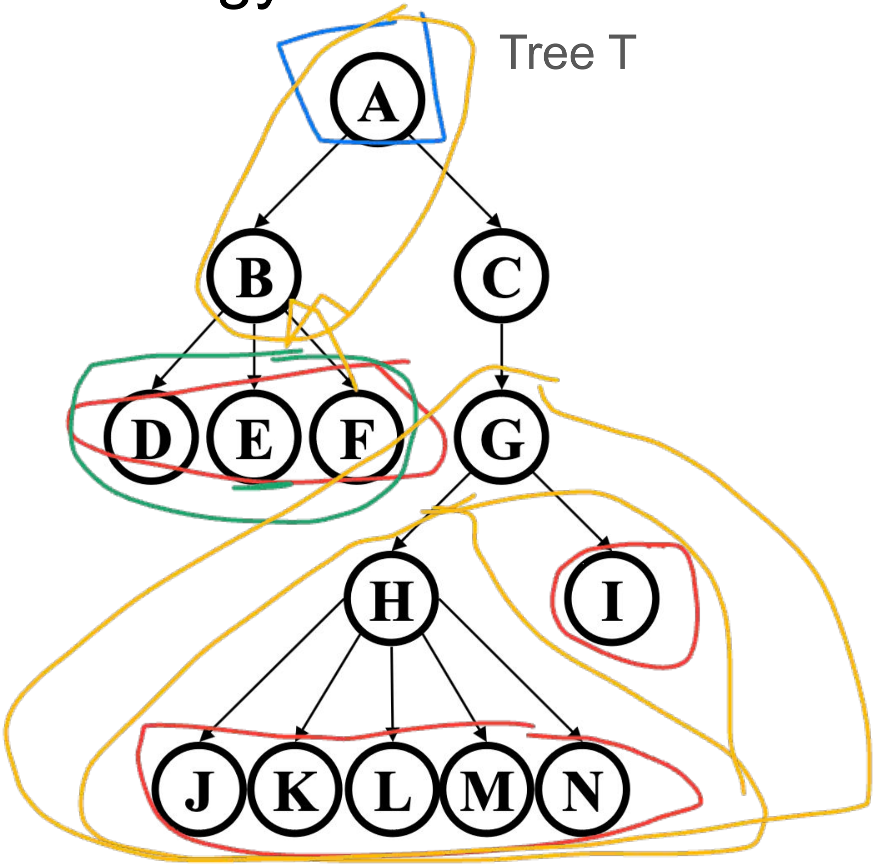
parent(**H**):

siblings(**E**):

ancestors(**F**):

descendents(**G**):

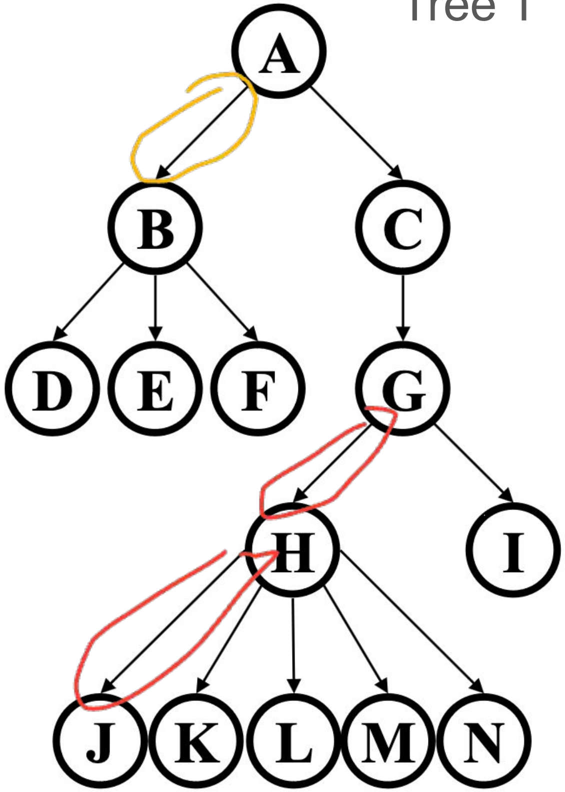
subtree(**G**):



Some More Tree Terminology

depth(**B**): 1
height(**G**): 2
height(**T**): 4
~~degree(**B**):~~
~~branching factor(**T**):~~

Tree T



height – number of edges in path fron node to deepest descendent

depth – number of edges in path from node to root

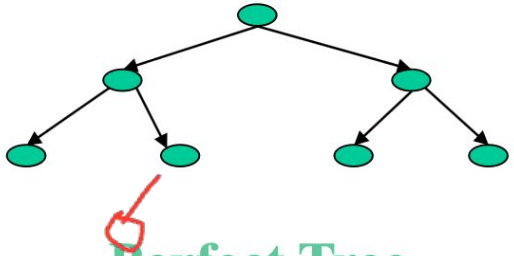
Types of Trees

Binary tree: Every node has ≤2 children

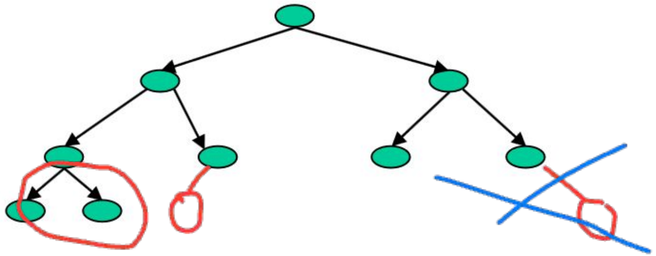
n-ary tree: Every node has ≤n children

Perfect tree: Every row is completely full

Complete tree: All rows except possibly the bottom are completely full, and it is filled from left to right



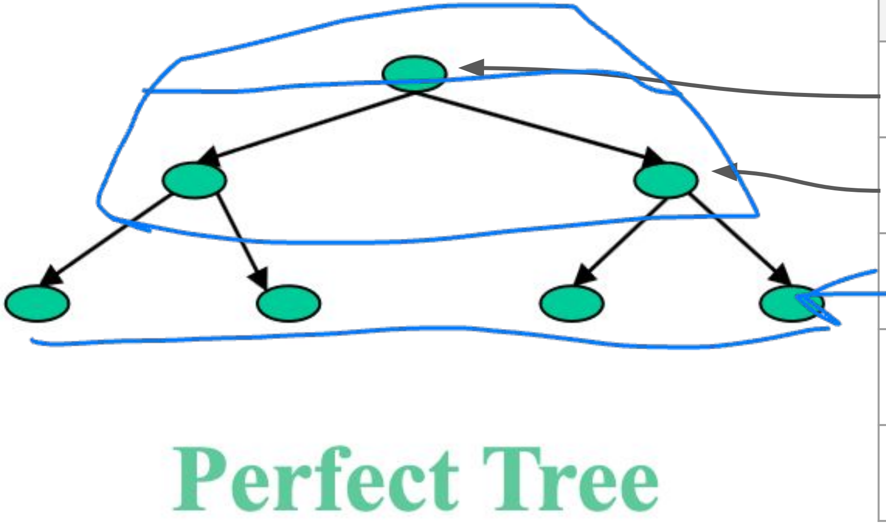
Perfect Tree



Complete Tree

More on Perfect Trees

Perfect tree: Every row is completely full



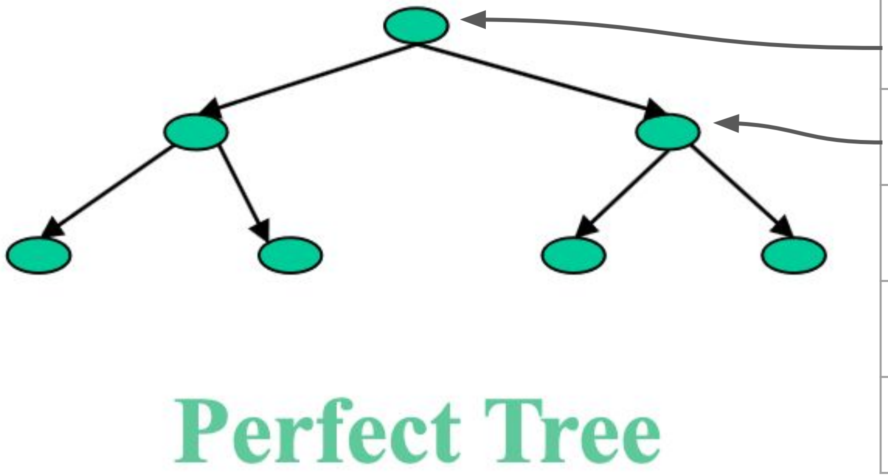
Height (h)	# nodes (n)	# leaves
0	1	1
1	3	2
2	7	4
3	15	8
h	$2^{h+1} - 1$	2^h

More on Perfect Trees

$$n = \sum_{i=0}^{h} 2^i = 2^{h+1} - 1$$

See Weiss 1.2.3 (p4)

Perfect tree: Every row is completely full

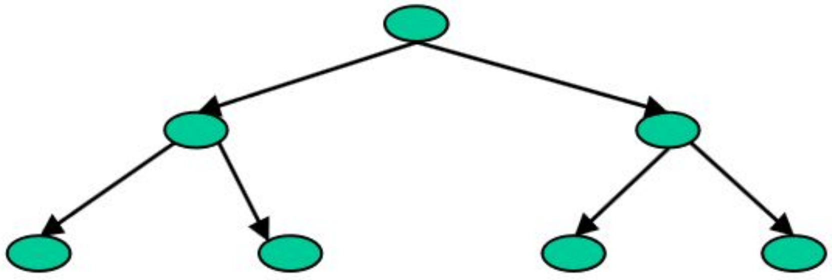


Height (h)	# nodes (n)	# leaves
0	1	1
1	3	2
2	7	4
3	15	8
h	$2^{h+1} - 1$	2^h

More on Perfect Trees

$$n = \sum_{i=0}^{h} 2^i = 2^{h+1} - 1$$

Perfect tree: Every row is completely full



Perfect Tree

How does the height of a perfect tree relate to the number of nodes?

$2^{h+1} - 1 = n$
$2^{h+1} = n$
$h + 1 = \log n$
$h = O(\log n)$