Priority Queues & Heaps

CS16: Introduction to Data Structures & Algorithms
Spring 2019
Outline

- Priority Queues
  - Motivation
  - ADT
  - Implementation

- Heaps
  - insert() and upheap()
  - removeMin() and downheap()
Motivation

‣ Priority queues store items with various priorities

‣ Examples
  ‣ Plane departures: some flights have higher priority than others
  ‣ Bandwidth management: real-time traffic like Skype transmitted first
Priority Queue ADT

- Stores key/element pairs
  - key determines position in queue
- **insert**(key, element):
  - inserts element with key
- **removeMin**( ):
  - removes pair w/ smallest key and returns element
Priority Queue Implementations

Activity #1

2 min
Priority Queue Implementations

Activity #1

2 min
Priority Queue Implementations

Activity #1

1 min
Priority Queue Implementations

Activity #1
## Priority Queue Implementation

<table>
<thead>
<tr>
<th>Implementation</th>
<th>insert</th>
<th>removeMin</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unsorted Array</td>
<td>(O(1))</td>
<td>(O(n))</td>
</tr>
<tr>
<td>Sorted Array</td>
<td>(O(n))</td>
<td>(O(1))</td>
</tr>
<tr>
<td>Unsorted Linked List</td>
<td>(O(1))</td>
<td>(O(n))</td>
</tr>
<tr>
<td>Sorted Linked List</td>
<td>(O(n))</td>
<td>(O(1))</td>
</tr>
<tr>
<td>Hash Table</td>
<td>(O(1))</td>
<td>(O(n))</td>
</tr>
<tr>
<td>Heap</td>
<td>(O(\log n))</td>
<td>(O(\log n))</td>
</tr>
</tbody>
</table>
What is a Heap?

- Data structure that implements priority queue
- Heaps can be implemented with
  - Tree
  - Array
- Tree-based heap
Heap Properties

- Binary tree
  - each node has at most 2 children
- Each node has a priority (key)
- Heap has an order
  - min-heap: \( n.\text{parent}.\text{key} \leq n.\text{key} \)
  - max-heap: \( n.\text{parent}.\text{key} \geq n.\text{key} \)
- Left-complete
- Height of \( O(\log n) \)
Heap Properties

- To implement priority queue
  - insert key/element pair at each node

![Heap Diagram]

- Node: (1, Tsunade)
  - Children: (5, Kakashi), (3, Danzo)
- Node: (5, Kakashi)
  - Child: (9, Naruto)
- Node: (3, Danzo)
  - Child: (7, Sakura)
Heap — insert()

- Need to keep track of “insertion node”
  - leaf where we will insert new node...
  - ...so we can keep heap left-complete

```
insertion node
```

```
2
/  \
5   6
/  |  /
9   7
```

```
insertion node
```
Heap — insert( )

- Ex: insert(1)
  - replace insertion node w/ new node

Heap order violated!
Heap — upheap()

- Repair heap: swap new element up tree until keys are sorted
- First swap fixes everything below new location
  - since every node below 6’s old location has to be at least 6…
  - …they must be at least 1
Heap — `upheap()`

- One more swap since `1 ≤ 2`
- Now left-completeness and order are satisfied
Heap insert( )

Activity #1

2 min
Heap insert( )

Activity #2

2 min
Heap insert()
Heap insert()
Heap — upheap() Summary

- After inserting a key $k$, order may be violated
- upheap() restores order by
  - swapping key upward from insertion node
  - terminates when either root is reached
  - …or some node whose parent has key at most $k$
- Heap insertion has runtime
  - $O(\log n)$, why?
  - because heap has height $O(\log n)$
  - perfect binary tree with $n$ nodes has height $\log(n+1)-1$
Heap — removeMin()

- Remove root
  - because it is always the smallest element
- How can we remove root w/o destroying heap?
Heap — removeMin()

- Instead swap root with last element & remove it
  - removing last element is easy

Order destroyed!

23
Heap — removeMin()

- Now swap root down as necessary

Heap is in order!
Heap — downheap( ) Summary

- downheap( ) restores order by
  - swapping key downward from root with smaller of 2 children
  - terminates when either leaf is reached or
    - ...some node whose children has key at least k
- downheap( ) has runtime
  - $O(\log n)$, why?
  - because heap has height $O(\log n)$
Heap removeMin()
Heap removeMin()
Heap removeMin()
Heap removeMin( )
Summary of Heap

- `insert(key, value)`
  - insert value at insertion node
    - insertion node must be kept track of
  - `upheap()` from insertion node as necessary

- `removeMin()`
  - swap root with last item
  - delete (swapped) last item
  - `downheap()` from root as necessary
Array-based Heap

- Heap with \( n \) keys can be represented with array of size \( n+1 \)
- Storing nodes in array
  - Node stored at index \( i \)
    - left child stored at index \( 2i \)
    - right child stored at index \( 2i+1 \)
  - Leaves & edges not stored
  - Cell 0 not used
- Operations
  - insert: store new node at index \( n+1 \)
  - removeMin: swap w/ index \( n \) and remove
Finding Insertion Node

- Can be found in $O(\log n)$
- Start at last added node
- Go up until a left child or root is reached
- If left child
  - go to sibling (corresponding right child)
  - then go down left until leaf is reached

Can be done in $O(1)$ time by using additional data structure...need this for project!
## Priority Queue Implementation

<table>
<thead>
<tr>
<th>Implementation</th>
<th>insert</th>
<th>removeMin</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unsorted Array</td>
<td>$O(1)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>Sorted Array</td>
<td>$O(n)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>Unsorted Linked List</td>
<td>$O(1)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>Sorted Linked List</td>
<td>$O(n)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>Hash Table</td>
<td>$O(1)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>Heap</td>
<td>$O(\log n)$</td>
<td>$O(\log n)$</td>
</tr>
</tbody>
</table>
References

- Slide #4
  - “Queue” in French means tail
  - The picture depicts the tail of a whale
- Slide #7
  - The picture is of a Transformers character named Junkheap which transforms from a waste management garbage truck
- Slide #8
  - The names are characters from the Anime series **Naruto** ([https://en.wikipedia.org/wiki/Naruto](https://en.wikipedia.org/wiki/Naruto))
  - The picture is the symbol of the Hidden Leaf Village (where the character Naruto is from)
  - The heap priorities represent the importance of the character in the village