Lecture 21: Priority Queues & Heapsort

CS 62

Spring 2018

Alexandra Papoutsaki & William Devanny

Priority Queues

A collection of entries that are inserted in such a way as to allow them to be dequeued in decreasing priority.

Entries can be either a pair of (key, value) or just values.

`structure5` assumes the 2nd.

Lowest value has highest priority.

Examples: OS scheduler, ER, airport, etc.

Priority Queue ADT

```
public interface PriorityQueue<E extends Comparable<E>>{
      public E remove(); //removes the element with smallest value
      public E getFirst(); //fetches lowest valued item from queue
      public void add(E value); //adds a value to the PQ
      public boolean isEmpty();
      public int size();
      public void clear();
}
```

Priority Queue implementations

1. As regular queue (array or linked list based) where either keep in order or search for lowest to remove:
    - One of add or remove will be $O(n)$
2. Heap representation (in arraylist) is more efficient:
    - $O(\log n)$ for both add and remove.
    - Insert into heap:
        - Place in next free position
        - “Percolate” it up.
    - Remove:
        - remove root
        - move last element in array up to root.
        - “Push” it down.
    - Peek element with highest priority in $O(1)$

`VectorHeap`

Class in `structure5`

Most heap operations, including insert and remove, execute in logarithmic time, but the minimum element can be returned in constant time.

`PriorityQueue` in standard Java

Treesort

- Build Binary search tree from the elements to be sorted (will cover later)
  - Adding one element is $O(\log n)$
  - Adding n elements is $O(n \log n)$
  - If tree is unbalanced, adding one element is $O(n)$, therefore worst case complexity $O(n^2)$
- Traverse the tree (in-order traversal) so that elements come out in sorted order
  - $O(n)$

- $O(n \log n)$ in best & average case and $O(n^2)$ in worst case
- Heapsort is always better!

HeapSort

- Make vector into a heap (depending on definition of priority max or min heap):
  - n instert operations = $O(n \log n)$
- Remove elements in order (the root since it contains smallest) and insert it into a sorted array. Keep updating heap
  - n remove operations = $O(n \log n)$
  - Total: $O(n \log n)$
- If clever can make into heap in $O(n)$ (1/2 vertices are leaves)
  - … but still $O(n \log n)$ total.
  - $O(1)$ extra space (for swaps)
  - Not stable
- https://visualgo.net/en/heap

Comparing Sorts

- Quicksort:
  - fastest on average $O(n \log n)$, but worst case is $O(n^2)$
  - Takes $O(\log n)$ extra space
- Heapsort:
  - $O(n \log n)$ in average & worst case. No extra space.
  - A bit slower on average than quicksory and mergesorts.
- Mergesort:
  - $O(n \log n)$ in average and worst case.
  - $O(n)$ extra space.
  - On-disk mergesort performs well on external files where not all fit in memory