Priority Queues
Priority Queue operations:
  • Insert
  • Delete Min

Recall basic “heap” data structure

Structure: “nearly” perfect binary tree
  • $n \geq 2^h$, where $n := \# \text{ nodes}$, $h := \text{ height}$

Heap condition: $key(v) \geq key(parent(v))$
Insert:

```
1    5    3
9   6  8  6  9  7  9
10 12 9  7  9
```

Insert 2

```
1    5    3
9   6  8  6  9  7  9
10 12 9  7  9
```

"float up"

```
1    5    3
9   6  8  6  9  7  8
10 12 9  7  9
```

```
1    5    3
9   6  8  6  9  7  8
10 12 9  7  9
```
Delete Min:

“sink down”
Insert and Delete Min: time $O(\log n)$

Array layout (an optimization)

If array is indexed from 1:

- $LeftChild(i) = 2i$
- $RightChild(i) = 2i + 1$
- $parent(i) = \lfloor i/2 \rfloor$
Building a heap from scratch in time $O(n)$

- Put all keys in the array
- Let $h$ be the height of the (implicit) tree
- Process nodes at levels $h - 1, h - 2, \ldots, 0$:
  - let the key at node $\nu$ “sink” to its correct position in the subtree rooted at $\nu$ (as in Delete Min)
- After processing level $j$, each node at level $j$ is the root of a heap
• Cost for level $j$: $O((h - j)2^j)$
  - $2^j$ nodes at level $j$, each costs time $O(h - j)$ to process
• Total cost: $O(t)$, where $t = \sum_{j=0}^{h-1} (h - j)2^j$
• Total cost: $O(t)$, where

$$t = \sum_{j=0}^{h-1} (h-j)2^j = \sum_{i=1}^{h} i2^{h-i}$$

$$= 2^h \sum_{i=1}^{h} i/2^i \leq n \sum_{i=1}^{h} i/2^i$$

Also, $\sum_{i=1}^{\infty} i/2^i = 2$:

\[ \begin{array}{cccc}
1/2 & 1/4 & 1/4 & \\
1/8 & 1/8 & 1/8 & \\
\vdots & \vdots & \vdots & \\
1 & 1/2 & 1/4 & \ldots \\
\end{array} \]

$\therefore t \leq 2n$
Application: Heap Sort

- Build heap: cost = $O(n)$
- For $i = 1, \ldots, n$: Delete Min
  - each Delete Min costs $O(\log n)$
  - total cost = $O(n \log n)$
- Total cost = $O(n \log n)$
Mergeable Priority Queues

Operations:

- Insert
- Delete Min
- Merge two queues

Using heaps:

- need to re-build — time $O(n)$

Using 2-3 trees:

- Can support all 3 operations in time $O(\log n)$
Mergeable Priority Queues using 2-3 trees

Same tree structure as ordinary 2-3 trees

Keys stored at leaves, but

- duplicates allowed
- keys not in any particular order

Internal nodes contain “min key values” as guides

**Insert:** just make a new leaf (anywhere), and update guides

**Delete Min:** follow guides to find min, delete, and update guides

**Merge:** use Join procedure, and update guides
Implementation notes for heaps

Instead of keys, each array entry $A[i]$ may point to some object, one of whose fields acts as a key, say $A[i].key$

Each object also stores its position in the heap, so $A[i].pos = i$.

These objects may be accessible through other data structures besides the heap

If $p$ points to such an object, we may modify $p.key$ directly

- $i = p.pos$ gives us $p$’s position in the heap
- We can “float” or “sink” $p$ as necessary to maintain heap condition
- All objects whose position changes must have their position fields updated as well
All heap operations still take time $O(\log n)$

Similar techniques can also be used for 2-3-tree-based priority queues