Binomial Heaps and Queues

Binary heaps based queues may be improvable.

1. perform some operations faster?
2. binomial queue
   - $O(\log n)$ merge.
3. redundant binomial queue
   - $O(1)$ insert
   - $\Theta(\log n)$ delete-min, reduce-key.
4. Fibonacci heaps
   - $O(1)$ insert, reduce-key
   - $O(\log n)$ delete-min
   - Dijkstra’s shortest paths + fibonacci heap $\Rightarrow \Theta(n \log n + m)$.

Binomial Heaps

Def: a **binomial heaps** is a tree that satisfies

**heap order**: key at node is smaller than keys of children.
Def: a **binomial queue** is
- a list of binomial heaps.
- for each \( i \), at most one heap of size \( 2^i \).

**Algorithm**: binomial heap merge
Input: \( H_1 \) and \( H_2 \), two \( n \) node binomial heaps.
- Make tree with greater root-key child of root of other tree.

**Runtime**: \( O(1) \)

**Analogy**:
- size \( n \) binomial queue \( \iff \) number \( n \)
- heap of size \( 2^i \) in list \( \iff \) \( i \)th binary digit of \( n \) is 1

**Example**:

```
      0
     / \
   1   2
     / \
   3   4
     / \
   5
```

**Algorithm**: merge queue
Input: \( Q_1 \), \( Q_2 \)
1. treat queues like numbers
2. add numbers.

**Runtime**: \( \Theta(\log n) \)

**Algorithm**: insert
Input: queue \( Q \), key \( k \)
1. make \( k \) into single-node queue \( Q' \).
2. merge \( Q \) and \( Q' \).

**Runtime**: merge = \( \Theta(\log n) \)

**Algorithm**: delete-min
Input: \( Q \)
1. find heap with min key \( \Rightarrow H_i \), size \( 2^i \)
2. remove heap \( H_i \) \( \Rightarrow Q' \)
3. delete root of \( H_i \) \( \Rightarrow i \) heaps sizes \( \{1, \ldots, i\} \)
4. treats heaps as queue \( Q'' \).
5. merge \( Q' \) and \( Q'' \).

**Runtime**: search + merge = \( \Theta(\log n) \)

**Algorithm**: reduce-key
Input: \( Q \), key \( k \), value \( v \).
1. find \( (k', v) \) in queue (assume we have pointer)
2. reduce \( k' \) to \( k \) and percolate-up.

**Runtime**: percolate-up = \( \Theta(\log n) \).

**Claim**: from empty queue, \( n \) inserts costs \( \Theta(n) \).

**Proof**:
- merge \( \iff \) add
- insert \( \iff \) increment
- \( n \) increments from zero costs \( \Theta(n) \).

**Algorithm**: build-heap
Input: unordered list
1. initialize empty queue.
2. insert each object.
Redundant Binomial Queue

“Θ(1) worst-case insert”

Recall: redundant binary counter

- digits are 0, 1, or 2.
- regularity property: 0s and 2s alternate.

Def: a redundant binomial queue is

- a list of binomial heaps.
- for each $i$, at most two heaps of size $2^i$.
- heaps (as digits) satisfy redundant counter regularity.

Claim: all queue operations can be implemented to maintain regularity.

Proof: all queue operations are like arithmetic operations

Claim: redundant binary counter has $Θ(1)$ insert, and $Θ(\log n)$ merge, delete-min, reduce-key.