Priority Queues

- MakeQueue: create new empty queue
- Insert($Q, k, p$): insert key $k$ with priority $p$
- Delete($Q, k$): delete key $k$ (given a pointer)
- DeleteMin($Q$): delete key with min priority
- Meld($Q_1, Q_2$): merge two sets
- Empty($Q$): returns if empty
- Size($Q$): returns #keys
- FindMin($Q$): returns key with min priority
Priority Queues – Ideal Times

MakeQueue, Meld, Insert, Empty, Size, FindMin: $O(1)$
Delete, DeleteMin: $O(\log n)$

Thm

1) Meld $O(n^{1-\varepsilon}) \Rightarrow$ DeleteMin $\Omega(\log n)$
2) Insert, Delete $O(t) \Rightarrow$ FindMin $\Omega(n/2^{O(t)})$

1) Follows from $\Omega(n \log n)$ sorting lower bound
Binomial Queues


- **Binomial tree**
  - each node stores a \((k,p)\) and satisfies **heap order** with respect to priorities
  - all nodes have a **rank** \(r\) (leaf = rank 0, a rank \(r\) node has exactly one child of each of the ranks 0..\(r-1\))

- **Binomial queue**
  - forest of binomial trees with roots stored in a list with strictly increasing root ranks
Problem

Implement binomial queue operations to achieve the ideal times in the *amortized* sense

**Hints**

1) Two rank $i$ trees can be linked to create a rank $i+1$ tree in $O(1)$ time

2) Potential $\Phi = \text{max rank} + \#\text{roots}$
Dijkstra’s Algorithm
(Single source shortest path problem)

Algorithm Dijkstra(V, E, w, s)
    Q := MakeQueue
    dist[s] := 0
    Insert(Q, s, 0)
    for v ∈ V \ { s } do
        dist[v] := +∞
        Insert(Q, v, +∞)
    while Q ≠ ∅ do
        v := DeleteMin(Q)
        foreach u : (v, u) ∈ E do
            if u ∈ Q and dist[v]+w(v, u) < dist[u] then
                dist[u] := dist[v]+w(v, u)
                DecreaseKey(u, dist[u])

n x Insert + n x DeleteMin + m x DecreaseKey
Binary heaps / Binomial queues : O((n + m)·log n)
# Priority Bounds

<table>
<thead>
<tr>
<th></th>
<th>Binomial Queues [Vuillemin 78]</th>
<th>Fibonacci Heaps [Fredman, Tarjan 84]</th>
<th>Run-Relaxed Heaps [Driscoll, Gabow, Shrairman, Tarjan 88] [Brodal 96]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Insert</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Meld</td>
<td>1</td>
<td>1</td>
<td>-</td>
</tr>
<tr>
<td>Delete</td>
<td>( \log n )</td>
<td>( \log n )</td>
<td>( \log n )</td>
</tr>
<tr>
<td>DeleteMin</td>
<td>( \log n )</td>
<td>( \log n )</td>
<td>( \log n )</td>
</tr>
<tr>
<td>DecreaseKey</td>
<td>( \log n )</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

\[\text{Amortized} \quad \downarrow \quad \text{Worst-case}\]

- Dijkstra’s Algorithm \( O(m + n \cdot \log n) \)
- (and Minimum Spanning Tree \( O(m \cdot \log^* n) \))

Empty, FindMin, Size, MakeQueue – \( O(1) \) worst-case time
Fibonacci Heaps


- **F-tree**
  - heap order with respect to priorities
  - all nodes have a rank \( r \in \{\text{degree, degree} + 1\} \)
    \((r = \text{degree} + 1 \iff \text{node is marked as having lost a child})\)
  - The \( i'\text{th child} \) of a node from the right has rank \( \geq i - 1 \)

- **Fibonacci Heap**
  - forest (list) of F-trees (trees can have equal rank)
Thm  Max rank of a node in an F-tree is $O(\log n)$

Proof  A rank $r$ node has at least 2 children of rank $\geq r - 3$. By induction subtree size is at least $2^{\lfloor r/3 \rfloor}$  

(in fact the size is at least $\varphi^r$, where $\varphi=(1+\sqrt{5})/2$)
Problem

Implement Fibonacci Heap operations with amortized $O(1)$ time for all operations, except $O(\log n)$ for deletions.

Hints

1) Two rank $i$ trees can be linked to create a rank $i+1$ tree in $O(1)$ time.

2) Eliminating nodes violating order or nodes having lost two children.

3) Potential $\Phi = 2 \cdot \text{marks} + \#\text{roots}$
Implementation of Fibonacci Heap Operations

- **FindMin**: Maintain pointer to min root
- **Insert**: Create new tree = new rank 0 node [+1]
- **Join**: Concatenate two forests unchanged
- **Delete**: DecreaseKey -∞ + DeleteMin
- **DeleteMin**: Remove min root [-1]
  + add children to forest [+O(log n )]
  + bucketsort roots by rank only O(log n ) not linked below
  + link while two roots equal rank [-1 each]
- **DecreaseKey**: Update priority + cut edge to parent [+3]
  + if parent now has r − 2 children, recursively cut parent edges [-1 each, +1 final cut]

* = potential change
Worst-Case Operations
(without Join)


Basic ideas

- Require $\leq \text{max-rank} + 1$ **trees** in forest
  (otherwise $\exists$ rank $r$ where two trees can be linked)
- Replace cutting in F-trees by having $O(\log n)$ nodes **violating heap order**
- **Transformation** replacing two rank $r$ violations by one rank $r+1$ violation