Fibonacci Heaps

Dijkstra's Algorithm

\[ d[1] \leftarrow 0 \quad \text{ENQUEUE}(v) \in Q \]

for \( i = 1 \) to \( n-1 \) do
  \[ k \leftarrow \text{DEQUEUE-MIN}(Q) \]
  for each \((u, v) \in E\) do
    if \( d[u] + 1 \leq d[v] \)
      \[ d[v] \leftarrow d[u] + 1 \]
    if \( v \in Q \)
      \[ \text{ENQUEUE}(v) \in Q \]
    else
      \[ \text{DECREASE-KEY}(v, d[v]) \in Q \]
  endfor
endfor

Data structure to implement \( Q \)?

- \( \text{MIN-HEAP} \): \( O(\log n) \) time for all operations
  - Dijkstra has \( O(m \log n) \) running time (dominated by \( O(m) \) \( \text{DECREASE-KEY} \) operations)

Q: Can we do \( \text{DECREASE-KEY} \) in \( O(1) \) time and other operations in \( O(\log n) \)?
  (Note: There are \( O(m) \) \( \text{DECREASE-KEY} \) operations and \( O(n) \) \( \text{ENQUEUE}/\text{DEQUEUE-MIN} \) operations)

Fibonacci Heaps achieve this, thereby reducing the running time of Dijkstra to \( O(m + n \log n) \).

Amortization: Individual operations can be expensive, but will lead to structural simplification that makes future operations cheaper. Overall, the average cost of each operation will be smaller than the worst-case cost of an operation.

Fibonacci Heaps were introduced by Fredman and Tarjan in 1984.

Many min-heaps (recall: key [parent] \leq key [child]) with external pointers to min.
**ENQUEUE**: Add a new singleton tree: $O(1)$ time

Problem: If we enqueue $n$ elements and then try to implement
a **DEQUEUE-MIN** operation, it will take $O(n)$ time!

Indeed, it will!!! But remember, we are interested in
amortized running times—not worst-case bounds.

• **MERGE TREES**: Compare roots and make the larger root a
child of the smaller one; update root list.

• **CUT A NON-LEAF NODE**:

• **DECREASE-KEY**: Cut the node and decrease its key.

• **DEQUEUE-MIN**: Remove MIN (external pointer) and add
children to list of roots.

Problem: Updating the MIN pointer takes time $\propto \# \text{ of roots}$

Solution: When we find a MIN, we also do additional
work to consolidate trees and move the next
MIN cheaper.

Rank: $\# \text{ of children}$

Rule: When finding MIN, merge heaps until each
rank value has $\leq 1$ heap. (Note: Merges are free
with MIN operations!)

Ideally, every heap should have exponential
$\# \text{ of entries in its range } (2^i + 2^i = 2^{i+1})$
to
avoid super-logarithmic ranks.

(Recall: $\# \text{ of distinct ranks } = \text{cost of MIN}$)

Problem: How to ensure this when nodes
are cut due to **DECREASE-KEY** operations?

Mark. A node is marked if it loses
Warning: A node is marked if it loses a child. Root nodes are always unmarked.
After cutting a node, move up towards the root cutting marked nodes until you reach an unmarked node which you now mark.
The cut nodes are unmarked since they are roots now.

Analysis: $\phi = k \left( \text{# of roots} + 2 \cdot \text{# of marked nodes} \right)$

- **ENQUEUE**: $O(1)$ real work; $\Delta \phi = k = O(1)$
- **DEQUEUE-MIN**: # of roots $= r$; # of new roots $= O(\log n)$

\[
O(r + \log n) \text{ real work}; \quad \Delta \phi = k \left( O(\log n) - r \right)
\]

For large enough $k$, overall $= O(\log n)$
- **DECREASE-KEY**: “cascading cuts”

$O(\text{# of cut nodes})$ real work;

\[
\Delta \phi = -2kc + kc + O(k)
\]

For large enough $k$, overall $= O(k) = O(1)$

What remains? To show that every heap of rank $r$ has $\alpha^r$ nodes where $\alpha$ is some constant $> 1$.

Lemma: The $i$th child to be added to a vertex has rank $\geq i - 2$

Proof: When it was added, it had rank at least $i - 1$ (equal rank roots are merged). It couldn’t have lost more than one child without being cut.
Lower bound on heap size

\[ s_k \geq \sum_{i=1}^{k-2} s_i \]

\[ s_k - s_{k-1} \geq s_{k-2} \]

\[ s_k \geq s_{k-1} + s_{k-2} \]

Now you know why it's called a Fibonacci heap!