8.3 Fibonacci Heaps

Collection of trees that fulfill the heap property.

Structure is much more relaxed than binomial heaps.
8.3 Fibonacci Heaps

Additional implementation details:

- Every node $x$ stores its degree in a field $x$.degree. Note that this can be updated in constant time when adding a child to $x$.
- Every node stores a boolean value $x$.marked that specifies whether $x$ is marked or not.
8.3 Fibonacci Heaps

The potential function:

- $t(S)$ denotes the number of trees in the heap.
- $m(S)$ denotes the number of marked nodes.
- We use the potential function $\Phi(S) = t(S) + 2m(S)$.

The potential is $\Phi(S) = 5 + 2 \cdot 3 = 11$. 
We assume that one unit of potential can pay for a constant amount of work, where the constant is chosen “big enough” (to take care of the constants that occur).

To make this more explicit we use $c$ to denote the amount of work that a unit of potential can pay for.
8.3 Fibonacci Heaps

S. minimum()

- Access through the min-pointer.
- Actual cost $\Theta(1)$.
- No change in potential.
- Amortized cost $\Theta(1)$. 
8.3 Fibonacci Heaps

S. merge($S'$)

- Merge the root lists.
- Adjust the min-pointer

Running time:

- Actual cost $O(1)$.
- No change in potential.
- Hence, amortized cost is $O(1)$. 
8.3 Fibonacci Heaps

S. merge($S'$)

- Merge the root lists.
- Adjust the min-pointer

Running time:

- Actual cost $O(1)$. 
8.3 Fibonacci Heaps

### S. merge($S'$)

- Merge the root lists.
- Adjust the min-pointer

![Diagram of Fibonacci Heaps]

**Running time:**

- Actual cost $\mathcal{O}(1)$.
- No change in potential.
8.3 Fibonacci Heaps

S. merge($S'$)

- Merge the root lists.
- Adjust the min-pointer

Running time:

- Actual cost $\Theta(1)$.
- No change in potential.
- Hence, amortized cost is $\Theta(1)$. 
8.3 Fibonacci Heaps

S. insert($x$)

- Create a new tree containing $x$.
- Insert $x$ into the root-list.
- Update min-pointer, if necessary.
8.3 Fibonacci Heaps

**S. insert(x)**

- Create a new tree containing \( x \).
- Insert \( x \) into the root-list.
- Update min-pointer, if necessary.

![Diagram of Fibonacci Heap](image-url)
8.3 Fibonacci Heaps

S. insert($x$)

- Create a new tree containing $x$.
- Insert $x$ into the root-list.
- Update min-pointer, if necessary.

Running time:

- Actual cost $\mathcal{O}(1)$.
- Change in potential is $+1$.
- Amortized cost is $c + \mathcal{O}(1) = \mathcal{O}(1)$. 
S. delete-min(x)

### 8.3 Fibonacci Heaps

#### S. delete-min(x)

- **Delete minimum:** Add child-trees to heap; time: $O(\log n)$.
- **Update min-pointer:** Time: $(t + D(\text{min})) \cdot O(1)$.

- **Consolidate root-list** so that no roots have the same degree. Time $t \cdot O(1)$ (see next slide).
8.3 Fibonacci Heaps

S. delete-min(x)

- Delete minimum; add child-trees to heap;
  time: $D(\text{min}) \cdot O(1)$.
8.3 Fibonacci Heaps

S. delete-min(x)

- Delete minimum; add child-trees to heap; time: \( D(\text{min}) \cdot O(1) \).
- Update min-pointer; time: \( (t + D(\text{min})) \cdot O(1) \).

Min consolidation:

- Time \( t \cdot O(1) \) (see next slide).
8.3 Fibonacci Heaps

**S. delete-min(x)**

- Delete minimum; add child-trees to heap; time: $D(\min) \cdot \mathcal{O}(1)$.
- Update min-pointer; time: $(t + D(\min)) \cdot \mathcal{O}(1)$. 

Consolidate root-list so that no roots have the same degree. Time $t \cdot \mathcal{O}(1)$ (see next slide).
8.3 Fibonacci Heaps

S. delete-min\(x\)

- Delete minimum; add child-trees to heap; time: \(D(\text{min}) \cdot \Theta(1)\).
- Update min-pointer; time: \((t + D(\text{min})) \cdot \Theta(1)\).

- Consolidate root-list so that no roots have the same degree. Time \(t \cdot \Theta(1)\) (see next slide).
8.3 Fibonacci Heaps

Consolidate:
8.3 Fibonacci Heaps

Consolidate:

![Diagram of Fibonacci Heaps]
8.3 Fibonacci Heaps

Consolidate:
8.3 Fibonacci Heaps

Consolidate:

![Diagram of Fibonacci Heaps]

- **min**
- **current**
8.3 Fibonacci Heaps

Consolidate:

current

min

8.3 Fibonacci Heaps
8.3 Fibonacci Heaps

Consolidate:

```
min

current
```
8.3 Fibonacci Heaps

Consolidate:
8.3 Fibonacci Heaps

Consolidate:

![Diagram of Fibonacci Heaps with nodes and edges]
8.3 Fibonacci Heaps

Consolidate:

![Diagram showing consolidation process in Fibonacci Heaps]
8.3 Fibonacci Heaps

Consolidate:

---

current

min

---

min
8.3 Fibonacci Heaps

Consolidate:

![Diagram showing the consolidation process in a Fibonacci heap.](Image)
8.3 Fibonacci Heaps

Consolidate:
8.3 Fibonacci Heaps

Consolidate:
8.3 Fibonacci Heaps

Consolidate:
8.3 Fibonacci Heaps

Consolidate:
8.3 Fibonacci Heaps

Consolidate:
8.3 Fibonacci Heaps

Consolidate:
Consolidate:
8.3 Fibonacci Heaps

Consolidate:
8.3 Fibonacci Heaps

Consolidate:
8.3 Fibonacci Heaps

Actual cost for delete-min()

- At most \( D_n + t \) elements in root-list before consolidate.
8.3 Fibonacci Heaps

Actual cost for `delete-min()`

- At most \( D_n + t \) elements in root-list before consolidate.
- Actual cost for a `delete-min` is at most \( \mathcal{O}(1) \cdot (D_n + t) \).

Hence, there exists \( c_1 \) s.t. actual cost is at most \( c_1 \cdot (D_n + t) \).
8.3 Fibonacci Heaps

Actual cost for delete-min()

- At most $D_n + t$ elements in root-list before consolidate.
- Actual cost for a delete-min is at most $O(1) \cdot (D_n + t)$.
  Hence, there exists $c_1$ s.t. actual cost is at most $c_1 \cdot (D_n + t)$.

Amortized cost for delete-min()

- $t' \leq D_n + 1$ as degrees are different after consolidating.
8.3 Fibonacci Heaps

Actual cost for delete-min()

- At most $D_n + t$ elements in root-list before consolidate.
- Actual cost for a delete-min is at most $O(1) \cdot (D_n + t)$.
  Hence, there exists $c_1$ s.t. actual cost is at most $c_1 \cdot (D_n + t)$.

Amortized cost for delete-min()

- $t' \leq D_n + 1$ as degrees are different after consolidating.
- Therefore $\Delta \Phi \leq D_n + 1 - t$;
8.3 Fibonacci Heaps

Actual cost for delete-min()

- At most $D_n + t$ elements in root-list before consolidate.
- Actual cost for a delete-min is at most $O(1) \cdot (D_n + t)$.
  Hence, there exists $c_1$ s.t. actual cost is at most $c_1 \cdot (D_n + t)$.

Amortized cost for delete-min()

- $t' \leq D_n + 1$ as degrees are different after consolidating.
- Therefore $\Delta \Phi \leq D_n + 1 - t$;
- We can pay $c \cdot (t - D_n - 1)$ from the potential decrease.
8.3 Fibonacci Heaps

Actual cost for delete-min()

- At most $D_n + t$ elements in root-list before consolidate.
- Actual cost for a delete-min is at most $O(1) \cdot (D_n + t)$.
  Hence, there exists $c_1$ s.t. actual cost is at most $c_1 \cdot (D_n + t)$.

Amortized cost for delete-min()

- $t' \leq D_n + 1$ as degrees are different after consolidating.
- Therefore $\Delta \Phi \leq D_n + 1 - t$;
- We can pay $c \cdot (t - D_n - 1)$ from the potential decrease.
- The amortized cost is
8.3 Fibonacci Heaps

Actual cost for delete-min()

- At most $D_n + t$ elements in root-list before consolidate.
- Actual cost for a delete-min is at most $O(1) \cdot (D_n + t)$.
  Hence, there exists $c_1$ s.t. actual cost is at most $c_1 \cdot (D_n + t)$.

Amortized cost for delete-min()

- $t' \leq D_n + 1$ as degrees are different after consolidating.
- Therefore $\Delta \Phi \leq D_n + 1 - t$;
- We can pay $c \cdot (t - D_n - 1)$ from the potential decrease.
- The amortized cost is
  
  $$c_1 \cdot (D_n + t) - c \cdot (t - D_n - 1)$$
8.3 Fibonacci Heaps

Actual cost for delete-min()

▶ At most $D_n + t$ elements in root-list before consolidate.
▶ Actual cost for a delete-min is at most $O(1) \cdot (D_n + t)$.
Hence, there exists $c_1$ s.t. actual cost is at most $c_1 \cdot (D_n + t)$.

Amortized cost for delete-min()

▶ $t' \leq D_n + 1$ as degrees are different after consolidating.
▶ Therefore $\Delta \Phi \leq D_n + 1 - t$;
▶ We can pay $c \cdot (t - D_n - 1)$ from the potential decrease.
▶ The amortized cost is

$$c_1 \cdot (D_n + t) - c \cdot (t - D_n - 1) \leq (c_1 + c)D_n + (c_1 - c)t + c$$
8.3 Fibonacci Heaps

Actual cost for delete-min()

- At most $D_n + t$ elements in root-list before consolidate.
- Actual cost for a delete-min is at most $\mathcal{O}(1) \cdot (D_n + t)$. Hence, there exists $c_1$ s.t. actual cost is at most $c_1 \cdot (D_n + t)$.

Amortized cost for delete-min()

- $t' \leq D_n + 1$ as degrees are different after consolidating.
- Therefore $\Delta \Phi \leq D_n + 1 - t$;
- We can pay $c \cdot (t - D_n - 1)$ from the potential decrease.
- The amortized cost is
  
  $$c_1 \cdot (D_n + t) - c \cdot (t - D_n - 1) \leq (c_1 + c)D_n + (c_1 - c)t + c \leq 2c(D_n + 1)$$
8.3 Fibonacci Heaps

Actual cost for delete-min()

- At most \(D_n + t\) elements in root-list before consolidate.
- Actual cost for a delete-min is at most \(\mathcal{O}(1) \cdot (D_n + t)\).
  Hence, there exists \(c_1\) s.t. actual cost is at most \(c_1 \cdot (D_n + t)\).

Amortized cost for delete-min()

- \(t' \leq D_n + 1\) as degrees are different after consolidating.
- Therefore \(\Delta \Phi \leq D_n + 1 - t\);
- We can pay \(c \cdot (t - D_n - 1)\) from the potential decrease.
- The amortized cost is
  \[
  c_1 \cdot (D_n + t) - c \cdot (t - D_n - 1) \\
  \leq (c_1 + c)D_n + (c_1 - c)t + c \leq 2c(D_n + 1) \leq \mathcal{O}(D_n)
  \]
8.3 Fibonacci Heaps

Actual cost for delete-min()
- At most $D_n + t$ elements in root-list before consolidate.
- Actual cost for a delete-min is at most $\mathcal{O}(1) \cdot (D_n + t)$.
  Hence, there exists $c_1$ s.t. actual cost is at most $c_1 \cdot (D_n + t)$.

Amortized cost for delete-min()
- $t' \leq D_n + 1$ as degrees are different after consolidating.
- Therefore $\Delta \Phi \leq D_n + 1 - t$;
- We can pay $c \cdot (t - D_n - 1)$ from the potential decrease.
- The amortized cost is
  \[
  c_1 \cdot (D_n + t) - c \cdot (t - D_n - 1) 
  \leq (c_1 + c) D_n + (c_1 - c) t + c 
  \leq 2c(D_n + 1) 
  \leq \mathcal{O}(D_n)
  \text{ for } c \geq c_1.
  \]
8.3 Fibonacci Heaps

If the input trees of the consolidation procedure are binomial trees (for example only singleton vertices) then the output will be a set of distinct binomial trees, and, hence, the Fibonacci heap will be (more or less) a Binomial heap right after the consolidation.

If we do not have delete or decrease-key operations then $D_n \leq \log n$. 
If the input trees of the consolidation procedure are binomial trees (for example only singleton vertices) then the output will be a set of distinct binomial trees, and, hence, the Fibonacci heap will be (more or less) a Binomial heap right after the consolidation.

If we do not have delete or decrease-key operations then $D_n \leq \log n$. 
Case 1: decrease-key does not violate heap-property

- Just decrease the key-value of element referenced by $h$. Nothing else to do.
Fibonacci Heaps: decrease-key(handle \( h, v \))

**Case 1: decrease-key does not violate heap-property**

- Just decrease the key-value of element referenced by \( h \).
  Nothing else to do.
Fibonacci Heaps: decrease-key(handle $h$, $v$)

Case 1: decrease-key does not violate heap-property

- Just decrease the key-value of element referenced by $h$.
  Nothing else to do.
Fibonacci Heaps: decrease-key(handle $h$, $v$)

Case 1: decrease-key does not violate heap-property

- Just decrease the key-value of element referenced by $h$. Nothing else to do.
Fibonacci Heaps: decrease-key(handle $h, v$)

Case 2: heap-property is violated, but parent is not marked

- Decrease key-value of element $x$ reference by $h$.
- If the heap-property is violated, cut the parent edge of $x$, and make $x$ into a root.
- Adjust min-pointers, if necessary.
- Mark the (previous) parent of $x$ (unless it’s a root).
Case 2: heap-property is violated, but parent is not marked

- Decrease key-value of element $x$ reference by $h$.
- If the heap-property is violated, cut the parent edge of $x$, and make $x$ into a root.
- Adjust min-pointers, if necessary.
- Mark the (previous) parent of $x$ (unless it’s a root).
Case 2: heap-property is violated, but parent is not marked

- Decrease key-value of element $x$ reference by $h$.
- If the heap-property is violated, cut the parent edge of $x$, and make $x$ into a root.
- Adjust min-pointers, if necessary.
- Mark the (previous) parent of $x$ (unless it’s a root).
Case 2: heap-property is violated, but parent is not marked

- Decrease key-value of element $x$ reference by $h$.
- If the heap-property is violated, cut the parent edge of $x$, and make $x$ into a root.
- Adjust min-pointers, if necessary.
- Mark the (previous) parent of $x$ (unless it’s a root).
Case 2: heap-property is violated, but parent is not marked

- Decrease key-value of element $x$ reference by $h$.
- If the heap-property is violated, cut the parent edge of $x$, and make $x$ into a root.
- Adjust min-pointers, if necessary.
- Mark the (previous) parent of $x$ (unless it’s a root).
Case 3: heap-property is violated, and parent is marked

- Decrease key-value of element $x$ reference by $h$.
- Cut the parent edge of $x$, and make $x$ into a root.
- Adjust min-pointers, if necessary.
- Continue cutting the parent until you arrive at an unmarked node.
Fibonacci Heaps: decrease-key(handle \( h, v \))

Case 3: heap-property is violated, and parent is marked

- Decrease key-value of element \( x \) reference by \( h \).
- Cut the parent edge of \( x \), and make \( x \) into a root.
- Adjust min-pointers, if necessary.
- Continue cutting the parent until you arrive at an unmarked node.
Fibonacci Heaps: decrease-key\((\text{handle } h, v)\)

Case 3: heap-property is violated, and parent is marked

- Decrease key-value of element \(x\) reference by \(h\).
- Cut the parent edge of \(x\), and make \(x\) into a root.
- Adjust min-pointers, if necessary.
- Continue cutting the parent until you arrive at an unmarked node.
Fibonacci Heaps: decrease-key(handle $h, v$)

Case 3: heap-property is violated, and parent is marked

- Decrease key-value of element $x$ reference by $h$.
- Cut the parent edge of $x$, and make $x$ into a root.
- Adjust min-pointers, if necessary.
- Continue cutting the parent until you arrive at an unmarked node.
Case 3: heap-property is violated, and parent is marked

- Decrease key-value of element $x$ reference by $h$.
- Cut the parent edge of $x$, and make $x$ into a root.
- Adjust min-pointers, if necessary.
- Continue cutting the parent until you arrive at an unmarked node.
**Fibonacci Heaps: decrease-key(handle \( h, v \))**

Case 3: heap-property is violated, and parent is marked

- Decrease key-value of element \( x \) reference by \( h \).
- Cut the parent edge of \( x \), and make \( x \) into a root.
- Adjust min-pointers, if necessary.
- Continue cutting the parent until you arrive at an unmarked node.
Fibonacci Heaps: decrease-key\((\text{handle } h, v)\)

Case 3: heap-property is violated, and parent is marked

- Decrease key-value of element \(x\) reference by \(h\).
- Cut the parent edge of \(x\), and make \(x\) into a root.
- Adjust min-pointers, if necessary.
- Continue cutting the parent until you arrive at an unmarked node.
Fibonacci Heaps: decrease-key(handle $h, v$)

Case 3: heap-property is violated, and parent is marked

- Decrease key-value of element $x$ reference by $h$.
- Cut the parent edge of $x$, and make $x$ into a root.
- Adjust min-pointers, if necessary.
- Continue cutting the parent until you arrive at an unmarked node.
Fibonacci Heaps: decrease-key(handle $h, v$)

Case 3: heap-property is violated, and parent is marked

- Decrease key-value of element $x$ reference by $h$.
- Cut the parent edge of $x$, and make $x$ into a root.
- Adjust min-pointers, if necessary.
- Continue cutting the parent until you arrive at an unmarked node.
Fibonacci Heaps: decrease-key(handle $h, v$)

Case 3: heap-property is violated, and parent is marked

- Decrease key-value of element $x$ reference by $h$.
- Cut the parent edge of $x$, and make $x$ into a root.
- Adjust min-pointers, if necessary.
- Execute the following:
  
  \[
  p \leftarrow \text{parent}[x];
  \]
  
  while ($p$ is marked)
  
  \[
  pp \leftarrow \text{parent}[p];
  \]
  
  cut of $p$; make it into a root; unmark it;
  
  \[
  p \leftarrow pp;
  \]
  
  if $p$ is unmarked and not a root mark it;
Fibonacci Heaps: decrease-key(handle \( h, v \))

**Actual cost:**

- Constant cost for decreasing the value.
- Constant cost for each of \( \ell \) cuts.
- Hence, cost is at most \( c_2 \cdot (\ell + 1) \), for some constant \( c_2 \).

**Amortized cost:**

- \( t' = t + \ell \), as every cut creates one new root.
- \( m' \leq m - (\ell - 1) + 1 = m - \ell + 2 \), since all but the first cut unmarks a node; the last cut may mark a node.
- \( \Delta \Phi \leq \ell + 2 \cdot (\ell - 1 + 2) \).
- Amortized cost is at most 
  \[ c_2 \cdot (\ell + 1) + c(4 - \ell) \leq (c_2 - c)\ell + 4c \]
  if \( c \geq c_2 \).
Fibonacci Heaps: \( \text{decrease-key}(\text{handle } h, v) \)

**Actual cost:**
- Constant cost for decreasing the value.
- Constant cost for each of \( l \) cuts.
- Hence, cost is at most \( c_2 \cdot (l + 1) \), for some constant \( c_2 \).

**Amortized cost:**
- \( t' = t + l \), as every cut creates one new root.
- \( m' = m - (l - 1) + 1 = m - l + 2 \), since all but the first cut unmarks a node; the last cut may mark a node.
- Amortized cost is at most \( c_2 \cdot (l + 1) + c(4 - l) \leq (c_2 - c)l + 4c + c_2 = O(1) \), if \( c \geq c_2 \).
Fibonacci Heaps: decrease-key(handle \( h, v \))

**Actual cost:**
- Constant cost for decreasing the value.
- Constant cost for each of \( \ell \) cuts.
- Hence, cost is at most \( c_2 \cdot (\ell + 1) \), for some constant \( c_2 \).

**Amortized cost:**
- \( t' = t + \ell \), as every cut creates one new root.
- \( m' \leq m - (\ell - 1) + 1 = m - \ell + 2 \), since all but the first cut unmarks a node, the last cut may mark a node.
- \( \Delta \Phi \leq \ell + 2 (4 - \ell + 2) = 4 - \ell \).
- Amortized cost is at most \( c_2 (\ell + 1) + c (4 - \ell) \leq (c_2 - c) \ell + 4c + c_2 = O(1) \), if \( c \geq c_2 \).
Fibonacci Heaps: decrease-key(handle $h, v$)

Actual cost:

- Constant cost for decreasing the value.
- Constant cost for each of $\ell$ cuts.
- Hence, cost is at most $c_2 \cdot (\ell + 1)$, for some constant $c_2$.

Amortized cost:

- $t' = t + \ell$, as every cut creates one new root.
- $m' \leq m - (\ell - 1) + 1 = m - \ell + 2$, since all but the first cut unmarks a node, the last cut may mark a node.
- Amortized cost is at most $O(1)$.
Fibonacci Heaps: decrease-key\((\text{handle } h, v)\)

**Actual cost:**
- Constant cost for decreasing the value.
- Constant cost for each of \(\ell\) cuts.
- Hence, cost is at most \(c_2 \cdot (\ell + 1)\), for some constant \(c_2\).

**Amortized cost:**
- \(t' = t + \ell\), as every cut creates one new root.
- \(m' \leq m - (\ell - 1) + 1 = m - \ell + 2\), since all but the first cut unmarks a node; the last cut may mark a node.
- \(\Delta \Phi \leq \ell + 2(-\ell + 2) = 4 - \ell\)
- Amortized cost is at most 4 - \(\ell\).
Fibonacci Heaps: `decrease-key(handle h, v)`

**Actual cost:**
- Constant cost for decreasing the value.
- Constant cost for each of \( \ell \) cuts.
- Hence, cost is at most \( c_2 \cdot (\ell + 1) \), for some constant \( c_2 \).

**Amortized cost:**
- \( t' = t + \ell \), as every cut creates one new root.
- \( m' \leq m - (\ell - 1) + 1 = m - \ell + 2 \), since all but the first cut unmarks a node; the last cut may mark a node.
- \( \Delta \Phi \leq \ell + 2(-\ell + 2) = 4 - \ell \)
- Amortized cost is at most...
Fibonacci Heaps: decrease-key(handle \(h, v\))

**Actual cost:**
- Constant cost for decreasing the value.
- Constant cost for each of \(\ell\) cuts.
- Hence, cost is at most \(c_2 \cdot (\ell + 1)\), for some constant \(c_2\).

**Amortized cost:**
- \(t' = t + \ell\), as every cut creates one new root.
- \(m' \leq m - (\ell - 1) + 1 = m - \ell + 2\), since all but the first cut unmarks a node; the last cut may mark a node.
- \(\Delta \Phi \leq \ell + 2(-\ell + 2) = 4 - \ell\)

Amortized cost is at most...
Fibonacci Heaps: \texttt{decrease-key}(handle \( h, v \))

\textbf{Actual cost:}

- Constant cost for decreasing the value.
- Constant cost for each of \( \ell \) cuts.
- Hence, cost is at most \( c_2 \cdot (\ell + 1) \), for some constant \( c_2 \).

\textbf{Amortized cost:}

- \( t' = t + \ell \), as every cut creates one new root.
- \( m' \leq m - (\ell - 1) + 1 = m - \ell + 2 \), since all but the first cut unmarks a node; the last cut may mark a node.
- \( \Delta \Phi \leq \ell + 2(-\ell + 2) = 4 - \ell \)
- Amortized cost is at most
  
  \[
  c_2(\ell + 1) + c(4 - \ell) \leq (c_2 - c) \ell + 4c + c_2 = \Theta(1),
  \]
  
  if \( c \geq c_2 \).
Fibonacci Heaps: decrease-key(handle $h, v$)

**Actual cost:**
- Constant cost for decreasing the value.
- Constant cost for each of $\ell$ cuts.
- Hence, cost is at most $c_2 \cdot (\ell + 1)$, for some constant $c_2$.

**Amortized cost:**
- $t' = t + \ell$, as every cut creates one new root.
- $m' \leq m - (\ell - 1) + 1 = m - \ell + 2$, since all but the first cut unmarks a node; the last cut may mark a node.
- $\Delta \Phi \leq \ell + 2(-\ell + 2) = 4 - \ell$
- Amortized cost is at most
  $$c_2(\ell + 1) + c(4 - \ell) \leq (c_2 - c)\ell + 4c + c_2 = \Theta(1),$$
  if $c \geq c_2$. 

Fibonacci Heaps: \texttt{decrease-key}(\texttt{handle} \ h, v)

Actual cost:

- Constant cost for decreasing the value.
- Constant cost for each of \( \ell \) cuts.
- Hence, cost is at most \( c_2 \cdot (\ell + 1) \), for some constant \( c_2 \).

Amortized cost:

- \( t' = t + \ell \), as every cut creates one new root.
- \( m' \leq m - (\ell - 1) + 1 = m - \ell + 2 \), since all but the first cut unmarks a node; the last cut may mark a node.
- \( \Delta \Phi \leq \ell + 2(-\ell + 2) = 4 - \ell \)
- Amortized cost is at most
  \[
  c_2(\ell + 1) + c(4 - \ell) \leq (c_2 - c)\ell + 4c + c_2 = \Theta(1),
  \]
  if \( c \geq c_2 \).
Fibonacci Heaps: decrease-key\((\text{handle } h, v)\)

**Actual cost:**
- Constant cost for decreasing the value.
- Constant cost for each of \(\ell\) cuts.
- Hence, cost is at most \(c_2 \cdot (\ell + 1)\), for some constant \(c_2\).

**Amortized cost:**
- \(t' = t + \ell\), as every cut creates one new root.
- \(m' \leq m - (\ell - 1) + 1 = m - \ell + 2\), since all but the first cut unmarks a node; the last cut may mark a node.
- \(\Delta \Phi \leq \ell + 2(-\ell + 2) = 4 - \ell\)
- Amortized cost is at most
  \[
  c_2(\ell + 1) + c(4 - \ell) \leq (c_2 - c) \ell + 4c + c_2 = \Theta(1),
  \]
  if \(c \geq c_2\).
Delete node

\[ H. \text{ delete}(\mathbf{x}): \]
\begin{itemize}
\item decrease value of \( \mathbf{x} \) to \(-\infty\).
\item delete-min.
\end{itemize}

Amortized cost: \( \Theta(D_n) \)
\begin{itemize}
\item \( \Theta(1) \) for decrease-key.
\item \( \Theta(D_n) \) for delete-min.
\end{itemize}
Lemma 1

Let $x$ be a node with degree $k$ and let $y_1, \ldots, y_k$ denote the children of $x$ in the order that they were linked to $x$. Then

$$\text{degree}(y_i) \geq \begin{cases} 0 & \text{if } i = 1 \\ i - 2 & \text{if } i > 1 \end{cases}$$
8.3 Fibonacci Heaps

Proof

- When $\gamma_i$ was linked to $x$, at least $\gamma_1, \ldots, \gamma_{i-1}$ were already linked to $x$.

- Hence, at this time $\deg(x) \geq i - 1$, and therefore also $\deg(\gamma_i) \geq i - 1$ as the algorithm links nodes of equal degree only.

- Since, then $\gamma_i$ has lost at most one child.

- Therefore, $\deg(\gamma_i) \geq i - 2$. 
8.3 Fibonacci Heaps

Proof

- When $y_i$ was linked to $x$, at least $y_1, \ldots, y_{i-1}$ were already linked to $x$.

- Hence, at this time $\text{degree}(x) \geq i - 1$, and therefore also $\text{degree}(y_i) \geq i - 1$ as the algorithm links nodes of equal degree only.

- Since, then $y_i$ has lost at most one child.

- Therefore, $\text{degree}(y_i) \geq i - 2$. 
8.3 Fibonacci Heaps

Proof

- When $y_i$ was linked to $x$, at least $y_1, \ldots, y_{i-1}$ were already linked to $x$.

- Hence, at this time $\text{degree}(x) \geq i - 1$, and therefore also $\text{degree}(y_i) \geq i - 1$ as the algorithm links nodes of equal degree only.

- Since, then $y_i$ has lost at most one child.

- Therefore, $\text{degree}(y_i) \geq i - 2$. 
8.3 Fibonacci Heaps

Proof

- When $y_i$ was linked to $x$, at least $y_1, \ldots, y_{i-1}$ were already linked to $x$.
- Hence, at this time $\text{degree}(x) \geq i - 1$, and therefore also $\text{degree}(y_i) \geq i - 1$ as the algorithm links nodes of equal degree only.
- Since, then $y_i$ has lost at most one child.
- Therefore, $\text{degree}(y_i) \geq i - 2$. 
8.3 Fibonacci Heaps

Let $s_k$ be the minimum possible size of a sub-tree rooted at a node of degree $k$ that can occur in a Fibonacci heap.
8.3 Fibonacci Heaps

- Let $s_k$ be the minimum possible size of a sub-tree rooted at a node of degree $k$ that can occur in a Fibonacci heap.
- $s_k$ monotonically increases with $k$.
8.3 Fibonacci Heaps

- Let $s_k$ be the minimum possible size of a sub-tree rooted at a node of degree $k$ that can occur in a Fibonacci heap.
- $s_k$ monotonically increases with $k$.
- $s_0 = 1$ and $s_1 = 2$. 
8.3 Fibonacci Heaps

- Let $s_k$ be the minimum possible size of a sub-tree rooted at a node of degree $k$ that can occur in a Fibonacci heap.
- $s_k$ monotonically increases with $k$.
- $s_0 = 1$ and $s_1 = 2$.

Let $x$ be a degree $k$ node of size $s_k$ and let $y_1, \ldots, y_k$ be its children.

$$s_k = 2 + \sum_{i=2}^{k} \text{size}(y_i)$$
Let $s_k$ be the minimum possible size of a sub-tree rooted at a node of degree $k$ that can occur in a Fibonacci heap.

- $s_k$ monotonically increases with $k$
- $s_0 = 1$ and $s_1 = 2$.

Let $x$ be a degree $k$ node of size $s_k$ and let $y_1, \ldots, y_k$ be its children.

\[
s_k = 2 + \sum_{i=2}^{k} \text{size}(y_i) \geq 2 + \sum_{i=2}^{k} s_{i-2}
\]
8.3 Fibonacci Heaps

Let $s_k$ be the minimum possible size of a sub-tree rooted at a node of degree $k$ that can occur in a Fibonacci heap.

- $s_k$ monotonically increases with $k$
- $s_0 = 1$ and $s_1 = 2$.

Let $x$ be a degree $k$ node of size $s_k$ and let $y_1, \ldots, y_k$ be its children.

$$s_k = 2 + \sum_{i=2}^{k} \text{size}(y_i)$$

$$\geq 2 + \sum_{i=2}^{k} s_{i-2}$$

$$= 2 + \sum_{i=0}^{k-2} s_i$$
8.3 Fibonacci Heaps

Definition 2
Consider the following non-standard Fibonacci type sequence:

\[
F_k = \begin{cases} 
1 & \text{if } k = 0 \\
2 & \text{if } k = 1 \\
F_{k-1} + F_{k-2} & \text{if } k \geq 2 
\end{cases}
\]

Facts:

1. \( F_k \geq \phi^k \).
2. For \( k \geq 2 \): \( F_k = 2 + \sum_{i=0}^{k-2} F_i \).

The above facts can be easily proved by induction. From this it follows that \( s_k \geq F_k \geq \phi^k \), which gives that the maximum degree in a Fibonacci heap is logarithmic.
\begin{align*}
k=0: & \quad 1 = F_0 \geq \Phi^0 = 1 \\
k=1: & \quad 2 = F_1 \geq \Phi^1 \approx 1.61 \\
k-2,k-1 \rightarrow k: & \quad F_k = F_{k-1} + F_{k-2} \geq \Phi^{k-1} + \Phi^{k-2} = \Phi^{k-2}(\Phi + 1) = \Phi^k \\
k=2: & \quad 3 = F_2 = 2 + 1 = 2 + F_0 \\
k-1 \rightarrow k: & \quad F_k = F_{k-1} + F_{k-2} = 2 + \sum_{i=0}^{k-3} F_i + F_{k-2} = 2 + \sum_{i=0}^{k-2} F_i
\end{align*}