Lecture 10
(Dijkstra’s SSSP & Fibonacci Heaps)

Rezaul A. Chowdhury
Department of Computer Science
SUNY Stony Brook
Fall 2017
Fibonacci Heaps  
(Fredman & Tarjan, 1984)

A Fibonacci heap can be viewed as an extension of Binomial heaps which supports DECREASE-KEY and DELETE operations efficiently.

<table>
<thead>
<tr>
<th>Heap Operation</th>
<th>Binary Heap (worst-case)</th>
<th>Binomial Heap (amortized)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MAKE-HEAP</strong></td>
<td>$\Theta(1)$</td>
<td>$\Theta(1)$</td>
</tr>
<tr>
<td><strong>INSERT</strong></td>
<td>$O(\log n)$</td>
<td>$\Theta(1)$</td>
</tr>
<tr>
<td><strong>MINIMUM</strong></td>
<td>$\Theta(1)$</td>
<td>$\Theta(1)$</td>
</tr>
<tr>
<td><strong>EXTRACT-MIN</strong></td>
<td>$O(\log n)$</td>
<td>$O(\log n)$</td>
</tr>
<tr>
<td><strong>UNION</strong></td>
<td>$\Theta(n)$</td>
<td>$\Theta(1)$</td>
</tr>
<tr>
<td><strong>DECREASE-KEY</strong></td>
<td>$O(\log n)$</td>
<td>—</td>
</tr>
<tr>
<td><strong>DELETE</strong></td>
<td>$O(\log n)$</td>
<td>—</td>
</tr>
</tbody>
</table>
**Fibonacci Heaps**  
(*Fredman & Tarjan, 1984*)

A *Fibonacci heap* can be viewed as an extension of Binomial heaps which supports **DECREASE-KEY** and **DELETE** operations efficiently.

<table>
<thead>
<tr>
<th>Heap Operation</th>
<th>Binary Heap (worst-case)</th>
<th>Binominal Heap (amortized)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MAKE-HEAP</strong></td>
<td>$\Theta(1)$</td>
<td>$\Theta(1)$</td>
</tr>
<tr>
<td><strong>INSERT</strong></td>
<td>$O(\log n)$</td>
<td>$\Theta(1)$</td>
</tr>
<tr>
<td><strong>MINIMUM</strong></td>
<td>$\Theta(1)$</td>
<td>$\Theta(1)$</td>
</tr>
<tr>
<td><strong>EXTRACT-MIN</strong></td>
<td>$O(\log n)$</td>
<td>$O(\log n)$</td>
</tr>
<tr>
<td><strong>UNION</strong></td>
<td>$\Theta(n)$</td>
<td>$\Theta(1)$</td>
</tr>
<tr>
<td><strong>DECREASE-KEY</strong></td>
<td>$O(\log n)$</td>
<td>$O(\log n)$ (worst case)</td>
</tr>
<tr>
<td><strong>DELETE</strong></td>
<td>$O(\log n)$</td>
<td>$O(\log n)$ (worst case)</td>
</tr>
</tbody>
</table>
A Fibonacci heap can be viewed as an extension of Binomial heaps which supports `DECREASE-KEY` and `DELETE` operations efficiently.

<table>
<thead>
<tr>
<th>Heap Operation</th>
<th>Binary Heap (worst-case)</th>
<th>Binomial Heap (amortized)</th>
<th>Fibonacci Heap (amortized)</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>MAKE-HEAP</code></td>
<td>$\Theta(1)$</td>
<td>$\Theta(1)$</td>
<td>$\Theta(1)$</td>
</tr>
<tr>
<td><code>INSERT</code></td>
<td>$O(\log n)$</td>
<td>$\Theta(1)$</td>
<td>$\Theta(1)$</td>
</tr>
<tr>
<td><code>MINIMUM</code></td>
<td>$\Theta(1)$</td>
<td>$\Theta(1)$</td>
<td>$\Theta(1)$</td>
</tr>
<tr>
<td><code>EXTRACT-MIN</code></td>
<td>$O(\log n)$</td>
<td>$O(\log n)$</td>
<td>$O(\log n)$</td>
</tr>
<tr>
<td><code>UNION</code></td>
<td>$\Theta(n)$</td>
<td>$\Theta(1)$</td>
<td>$\Theta(1)$</td>
</tr>
<tr>
<td><code>DECREASE-KEY</code></td>
<td>$O(\log n)$</td>
<td>$O(\log n)$ (worst case)</td>
<td>$\Theta(1)$</td>
</tr>
<tr>
<td><code>DELETE</code></td>
<td>$O(\log n)$</td>
<td>$O(\log n)$ (amortized)</td>
<td>$O(\log n)$</td>
</tr>
</tbody>
</table>
Dijkstra’s SSSP Algorithm with a Min-Heap
(SSSP: Single-Source Shortest Paths)

**Input:** Weighted graph $G = (V, E)$ with vertex set $V$ and edge set $E$, a weight function $w$, and a source vertex $s \in G[V]$.

**Output:** For all $v \in G[V]$, $v.d$ is set to the shortest distance from $s$ to $v$.

```plaintext
Dijkstra-SSSP ( G = (V,E), w, s )
1. for each $v \in G[V]$ do $v.d \leftarrow \infty$
2. $s.d \leftarrow 0$
3. $H \leftarrow \phi$  \{ empty min-heap \}
4. for each $v \in G[V]$ do INSERT( $H$, v )
5. while $H \neq \phi$ do
6. \hspace{1em} $u \leftarrow$ EXTRACT-MIN( $H$ )
7. \hspace{1em} for each $v \in Adj[u]$ do
8. \hspace{2em} if $v.d > u.d + w_{u,v}$ then
9. \hspace{3em} DECREASE-KEY( $H$, $v$, $u.d + w_{u,v}$ )
10. \hspace{1em} $v.d \leftarrow u.d + w_{u,v}$
```
Dijkstra’s SSSP Algorithm with a Min-Heap

(SSSP: Single-Source Shortest Paths)

**Input:** Weighted graph $G = (V, E)$ with vertex set $V$ and edge set $E$, a weight function $w$, and a source vertex $s \in G[V]$.

**Output:** For all $v \in G[V]$, $v.d$ is set to the shortest distance from $s$ to $v$.

**Dijkstra-SSSP** $(G = (V,E), w, s )$

1. for each $v \in G[V]$ do $v.d \leftarrow \infty$
2. $s.d \leftarrow 0$
3. $H \leftarrow \emptyset$ \quad \{ empty min-heap \}
4. for each $v \in G[V]$ do INSERT( $H$, $v$ )
5. while $H \neq \emptyset$ do
6. \quad $u \leftarrow$ EXTRACT-MIN( $H$ )
7. \quad for each $v \in Adj[u]$ do
8. \quad \quad if $v.d > u.d + w_{u,v}$ then
9. \quad \quad \quad DECREASE-KEY( $H$, $v$, $u.d + w_{u,v}$ )
10. \quad \quad $v.d \leftarrow u.d + w_{u,v}$

Let $n = |G[V]|$ and $m = |G[E]|$

- $\#$ INSERTS $= n$
- $\#$ EXTRACT-MINS $= n$
- $\#$ DECREASE-KEYS $\leq m$

Total cost

\[ \leq n(\text{cost}_{\text{Insert}} + \text{cost}_{\text{Extract-Min}}) + m(\text{cost}_{\text{Decrease-Key}}) \]
Dijkstra’s SSSP Algorithm with a Min-Heap

(SSSP: Single-Source Shortest Paths)

Input: Weighted graph \( G = (V, E) \) with vertex set \( V \) and edge set \( E \), a weight function \( w \), and a source vertex \( s \in G[V] \).

Output: For all \( v \in G[V] \), \( v.d \) is set to the shortest distance from \( s \) to \( v \).

Let \( n = |G[V]| \) and \( m = |G[E]| \)

For Binary Heap (worst-case costs):

\[
\begin{align*}
\text{cost}_{\text{Insert}} &= O(\log n) \\
\text{cost}_{\text{Extract-Min}} &= O(\log n) \\
\text{cost}_{\text{Decrease-Key}} &= O(\log n)
\end{align*}
\]

\( \therefore \) Total cost (worst-case)

\[ = O((m + n) \log n) \]
Dijkstra’s SSSP Algorithm with a Min-Heap

(SSSP: Single-Source Shortest Paths)

Input: Weighted graph $G = (V, E)$ with vertex set $V$ and edge set $E$, a weight function $w$, and a source vertex $s \in G[V]$.

Output: For all $v \in G[V]$, $v.d$ is set to the shortest distance from $s$ to $v$.

$
\text{Dijkstra-SSSP} (G = (V, E), w, s)
$

1. for each $v \in G[V]$ do $v.d \leftarrow \infty$
2. $s.d \leftarrow 0$
3. $H \leftarrow \phi$ \{ empty min-heap \}
4. for each $v \in G[V]$ do INSERT$(H, v)$
5. while $H \neq \emptyset$ do
6. \hspace{1em} $u \leftarrow \text{EXTRACT-MIN}(H)$
7. \hspace{1em} for each $v \in \text{Adj}[u]$ do
8. \hspace{2em} if $v.d > u.d + w_{u,v}$ then
9. \hspace{3em} DECREASE-KEY$(H, v, u.d + w_{u,v})$
10. \hspace{2em} $v.d \leftarrow u.d + w_{u,v}$

Let $n = |G[V]|$ and $m = |G[E]|$

For Binomial Heap (amortized costs):

\[\begin{align*}
\text{cost}_{\text{Insert}} &= O(1) \\
\text{cost}_{\text{Extract-Min}} &= O(\log n) \\
\text{cost}_{\text{Decrease-Key}} &= O(\log n) \quad \text{(worst-case)}
\end{align*}\]

$\therefore$ Total cost (worst-case)

\[= O((m + n) \log n)\]
Dijkstra’s SSSP Algorithm with a Min-Heap
(SSSP: Single-Source Shortest Paths)

Input: Weighted graph \( G = (V, E) \) with vertex set \( V \) and edge set \( E \), a weight function \( w \), and a source vertex \( s \in G[V] \).

Output: For all \( v \in G[V] \), \( v.d \) is set to the shortest distance from \( s \) to \( v \).

Let \( n = |G[V]| \) and \( m = |G[E]| \)

Total cost
\[
\leq n(cost_{Insert} + cost_{Extract-Min}) + m(cost_{Decrease-Key})
\]

Observation:
Obtaining a worst-case bound for a sequence of \( n \) INSERTS, \( n \) EXTRACT-MINS and \( m \) DECREASE-KEYS is enough.

\( \therefore \) Amortized bound per operation is sufficient.
Dijkstra’s SSSP Algorithm with a Min-Heap

(SSSP: Single-Source Shortest Paths)

Input: Weighted graph $G = (V, E)$ with vertex set $V$ and edge set $E$, a weight function $w$, and a source vertex $s \in G[V]$.

Output: For all $v \in G[V]$, $v.d$ is set to the shortest distance from $s$ to $v$.

Let $n = |G[V]|$ and $m = |G[E]|$

Total cost

$$\leq n(cost_{\text{Insert}} + cost_{\text{Extract-Min}}) + m(cost_{\text{Decrease-Key}})$$

Observation:

For $n(cost_{\text{Insert}} + cost_{\text{Extract-Min}})$ the best possible bound is $\Theta(n \log n)$. (else violates sorting lower bound)

Perhaps $m(cost_{\text{Decrease-Key}})$ can be improved to $o(m \log n)$. 

---

**Dijkstra-SSSP** $(G = (V,E), w, s )$

1. for each $v \in G[V]$ do $v.d \leftarrow \infty$
2. $s.d \leftarrow 0$
3. $H \leftarrow \phi$ \{ empty min-heap \}
4. for each $v \in G[V]$ do INSERT( $H$, $v$ )
5. while $H \neq \phi$ do
6. \hspace{1em} $u \leftarrow \text{EXTRACT-MIN}( H )$
7. \hspace{1em} for each $v \in \text{Adj}[u]$ do
8. \hspace{2em} if $v.d > u.d + w_{u,v}$ then
9. \hspace{3em} $\text{DECREASE-KEY}( H, v, u.d + w_{u,v} )$
10. \hspace{2em} $v.d \leftarrow u.d + w_{u,v}$
Fibonacci Heaps from Binomial Heaps

A *Fibonacci heap* can be viewed as an extension of Binomial heaps which supports `DECREASE-KEY` and `DELETE` operations efficiently.

But the trees in a Fibonacci heap are no longer binomial trees as we will be cutting subtrees out of them.

However, all operations (except `DECREASE-KEY` and `DELETE`) are still performed in the same way as in binomial heaps.

The *rank* of a tree is still defined as the number of children of the root, and we still link two trees if they have the same rank.
Implementing \texttt{DECREASE-KEY}(H, x, k)

\texttt{DECREASE-KEY}(H, x, k): One possible approach is to cut out the subtree rooted at \(x\) from \(H\), reduce the value of \(x\) to \(k\), and insert that subtree into the root list of \(H\).

\textbf{Problem}: If we cut out a lot of subtrees from a tree its size will no longer be exponential in its rank. Since our analysis of \texttt{EXTRACT-MIN} in binomial heaps was highly dependent on this exponential relationship, that analysis will no longer hold.

\textbf{Solution}: Limit #cuts among the children of any node to 2. We will show that the size of each tree will still remain exponential in its rank.

When a 2nd child is cut from a node \(x\), we also cut \(x\) from its parent leading to a possible sequence of cuts moving up towards the root.
Analysis of Fibonacci Heap Operations

Recurrence for Fibonacci numbers: \( f_n = \begin{cases} 0 & \text{if } n = 0, \\ 1 & \text{if } n = 1, \\ f_{n-1} + f_{n-2} & \text{otherwise.} \end{cases} \)

We showed in a previous lecture: \( f_n = \frac{1}{\sqrt{5}} (\phi^n - \hat{\phi}^n) \),

where \( \phi = \frac{1+\sqrt{5}}{2} \) and \( \hat{\phi} = \frac{1-\sqrt{5}}{2} \) are the roots \( z^2 - z - 1 = 0 \).
Lemma 1: For all integers $n \geq 0$, $f_{n+2} = 1 + \sum_{i=0}^{n} f_i$.

Proof: By induction on $n$.

Base case: $f_2 = 1 = 1 + 0 = 1 + f_0 = 1 + \sum_{i=0}^{n} f_i$.

Inductive hypothesis: $f_{k+2} = 1 + \sum_{i=0}^{k} f_i$ for $0 \leq k \leq n - 1$.

Then $f_{n+2} = f_{n+1} + f_n = f_n + (1 + \sum_{i=0}^{n-1} f_i) = 1 + \sum_{i=0}^{n} f_i$. 

**Analysis of Fibonacci Heap Operations**
Lemma 2: For all integers \( n \geq 0 \), \( f_{n+2} \geq \phi^n \).

Proof: By induction on \( n \).

Base case: \( f_2 = 1 = \phi^0 \) and \( f_3 = 2 > \phi^1 \).

Inductive hypothesis: \( f_{k+2} \geq \phi^k \) for \( 0 \leq k \leq n - 1 \).

Then \( f_{n+2} = f_{n+1} + f_n \)
\[ \geq \phi^{n-1} + \phi^{n-2} \]
\[ = (\phi + 1)\phi^{n-2} \]
\[ = \phi^2\phi^{n-2} \]
\[ = \phi^n \]
Lemma 3: Let $x$ be any node in a Fibonacci heap, and suppose that $k = rank(x)$. Let $y_1, y_2, ..., y_k$ be the children of $x$ in the order in which they were linked to $x$, from the earliest to the latest. Then $rank(y_i) \geq \max\{0, i - 2\}$ for $1 \leq i \leq k$.

Proof: Obviously, $rank(y_1) \geq 0$.

For $i > 1$, when $y_i$ was linked to $x$, all of $y_1, y_2, ..., y_{i-1}$ were children of $x$. So, $rank(x) \geq i - 1$.

Because $y_i$ is linked to $x$ only if $rank(y_i) = rank(x)$, we must have had $rank(y_i) \geq i - 1$ at that time.

Since then, $y_i$ has lost at most one child, and hence $rank(y_i) \geq i - 2$. 
Lemma 4: Let $z$ be any node in a Fibonacci heap with $n = size(z)$ and $r = rank(z)$. Then $r \leq \log_\phi n$.

Proof: Let $s_k$ be the minimum possible size of any node of rank $k$ in any Fibonacci heap.

Trivially, $s_0 = 1$ and $s_1 = 2$.

Since adding children to a node cannot decrease its size, $s_k$ increases monotonically with $k$.

Let $x$ be a node in any Fibonacci heap with $rank(x) = r$ and $size(x) = s_r$. 

Analysis of Fibonacci Heap Operations
Analysis of Fibonacci Heap Operations

Lemma 4: Let \( z \) be any node in a Fibonacci heap with \( n = \text{size}(z) \) and \( r = \text{rank}(z) \). Then \( r \leq \log_{\phi} n \).

Proof (continued): Let \( y_1, y_2, \ldots, y_r \) be the children of \( x \) in the order in which they were linked to \( x \), from the earliest to the latest.

Then \( s_r \geq 1 + \sum_{i=1}^{r} s_{\text{rank}(y_i)} \geq 1 + \sum_{i=1}^{r} s_{\max\{0,i-2\}} = 2 + \sum_{i=2}^{r} s_{i-2} \)

We now show by induction on \( r \) that \( s_r \geq f_{r+2} \) for all integer \( r \geq 0 \).

Base case: \( s_0 = 1 = f_2 \) and \( s_1 = 2 = f_3 \).

Inductive hypothesis: \( s_k \geq f_{k+2} \) for \( 0 \leq k \leq r - 1 \).

Then \( s_r \geq 2 + \sum_{i=2}^{r} s_{i-2} \geq 2 + \sum_{i=2}^{r} f_i = 1 + \sum_{i=1}^{r} f_i = f_{r+2} \).

Hence \( n \geq s_r \geq f_{r+2} \geq \phi^r \Rightarrow r \leq \log_{\phi} n \).
Analysis of Fibonacci Heap Operations

Corollary: The maximum degree of any node in an $n$ node Fibonacci heap is $O(\log n)$.

Proof: Let $z$ be any node in the heap.

Then from Lemma 4,

$$degree(z) = rank(z) \leq \log_\phi(size(z)) \leq \log_\phi n = O(\log n).$$
We extend the potential function used for binomial heaps:

$$\Phi(D_i) = 2t(D_i) + 3m(D_i),$$

where $D_i$ is the state of the data structure after the $i^{th}$ operation, $t(D_i)$ is the number of trees in the root list, and $m(D_i)$ is the number of marked nodes.
Analysis of Fibonacci Heap Operations

We extend the potential function used for binomial heaps:

\[ \Phi(D_i) = 2t(D_i) + 3m(D_i), \]

where \( D_i \) is the state of the data structure after the \( i^{th} \) operation, \( t(D_i) \) is the number of trees in the root list, and \( m(D_i) \) is the number of marked nodes.

**DECREASE-KEY( \( H, x, k_x \) ):** Let \( k = \#\text{cascading cuts performed} \).

Then the actual cost of cutting the tree rooted at \( x \) is 1, and the actual cost of each of the cascading cuts is also 1.

\[ : \text{overall actual cost, } c_i = 1 + k \]
Fibonacci Heaps from Binomial Heaps

Potential function: $\Phi(D_i) = 2t(D_i) + 3m(D_i)$

**DECREASE-KEY**($H, x, k_x$):

New trees: 1 tree rooted at $x$, and

1 tree produced by each of the $k$ cascading cuts.

\[ t(D_i) - t(D_{i-1}) = 1 + k \]

Marked nodes: 1 node unmarked by each cascading cut, and

at most 1 node marked by the last cut/cascading cut.

\[ m(D_i) - m(D_{i-1}) \leq -k + 1 \]

Potential drop, $\Delta_i = \Phi(D_i) - \Phi(D_{i-1})$

\[
\begin{align*}
\Delta_i &= 2(t(D_i) - t(D_{i-1})) + 3(m(D_i) - m(D_{i-1})) \\
&\leq 2(1 + k) + 3(-k + 1) \\
&= -k + 5
\end{align*}
\]
Potential function: \( \Phi(D_i) = 2t(D_i) + 3m(D_i) \)

\textsc{Decrease-Key} (\( H, x, k_x \)):

Amortized cost, \( \hat{c}_i = c_i + \Delta_i \)

\[ \leq (1 + k) + (-k + 5) \]

\[ = 6 \]

\[ = O(1) \]
Fibonacci Heaps from Binomial Heaps

Potential function: \( \Phi(D_i) = 2t(D_i) + 3m(D_i) \)

**EXTRACT-MIN(\( H \))**: 

Let \( d_n \) be the max degree of any node in an \( n \)-node Fibonacci heap.

Cost of creating the array of pointers is \( \leq d_n + 1 \).

Suppose we start with \( k \) trees in the doubly linked list, and perform \( l \) link operations during the conversion from linked list to array version. So we perform \( k + l \) work, and end up with \( k - l \) trees.

Cost of converting to the linked list version is \( k - l \).

Actual cost, \( c_i \leq d_n + 1 + (k + l) + (k - l) = 2k + d_n + 1 \)

Since no node is marked, and each link reduces the #trees by 1, 

potential change, \( \Delta_i = \Phi(D_i) - \Phi(D_{i-1}) \geq -2l \)
Fibonacci Heaps from Binomial Heaps

Potential function: \( \Phi(D_i) = 2t(D_i) + 3m(D_i) \)

**EXTRACT-MIN( H ):**

actual cost, \( c_i \leq d_n + 1 + (k + l) + (k - l) = 2k + d_n + 1 \)

potential change, \( \Delta_i = \Phi(D_i) - \Phi(D_{i-1}) \geq -2l \)

amortized cost, \( \hat{c}_i = c_i + \Delta_i \leq 2(k - l) + d_n + 1 \)

But \( k - l \leq d_n + 1 \) (as we have at most one tree of each rank)

So, \( \hat{c}_i \leq 3d_n + 3 = O(\log n) \).
Fibonacci Heaps from Binomial Heaps

Potential function: $\Phi(D_i) = 2t(D_i) + 3m(D_i)$

**DELETE**($H, x$):

**STEP 1:** **DECREASE-KEY**($H, x, -\infty$)

**STEP 2:** **EXTRACT-MIN**($H$)

amortized cost, $\hat{c}_i = \text{amortized cost of **DECREASE-KEY**}$

+ amortized cost of **EXTRACT-MIN**

$= O(1) + O(\log n)$

$= O(\log n)$