CSE 548: Analysis of Algorithms

Lecture 10
(Dijkstra's SSSP & Fibonacci Heaps)

Rezaul A. Chowdhury
Department of Computer Science
SUNY Stony Brook
Fall 2023

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.

Heap Operation	Binary Heap (worst-case)	Binomial Heap (amortized)
MAKE-HEAP	$\Theta(1)$	$\Theta(1)$
INSERT	$O(\log n)$	$\Theta(1)$
MINIMUM	$\Theta(1)$	$\Theta(1)$
EXTRACT-MIN	$O(\log n)$	$O(\log n)$
UNION	$\Theta(n)$	$\Theta(1)$
DECREASE-KEY	$O(\log n)$	$-$
DELETE	$O(\log n)$	$-$

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.

Heap Operation	Binary Heap (worst-case)	Binomial Heap (amortized)
MAKE-HEAP	$\Theta(1)$	$\Theta(1)$
INSERT	$O(\log n)$	$\Theta(1)$
MINIMUM	$\Theta(1)$	$\Theta(1)$
EXTRACT-MIN	$O(\log n)$	$O(\log n)$
UNION	$\Theta(n)$	$\Theta(1)$
DECREASE-KEY	$O(\log n)$	$O(\log n)$ (worst case)
DELETE	$O(\log n)$	$O(\log n)$

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.

Heap Operation	Binary Heap (worst-case)	Binomial Heap (amortized)	Fibonacci Heap (amortized)
MAKE-HEAP	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$
INSERT	$O(\log n)$	$\Theta(1)$	$\Theta(1)$
MINIMUM	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$
EXTRACT-MIN	$O(\log n)$	$O(\log n)$	$O(\log n)$
UNION	$\Theta(n)$	$\Theta(1)$	$\Theta(1)$
DECREASE-KEY	$O(\log n)$	$O(\log n)$ (worst case)	$\Theta(1)$
DELETE	$O(\log n)$	$O(\log n)$	$O(\log n)$

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.

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

    for each v ∈ G[V] do v.d ← ∞
    s.d ← 0
    H ← φ                                { empty min-heap }
    for each v ∈ G[V] do INSERT( H, v )
    while H ≠ ∅ do
        u ← EXTRACT-MIN( H )
        for each v ∈ Adj[u] do
            if v.d > u.d + w_{u,v} then
                DECREASE-KEY( H, v, u.d + w_{u,v} )
                v.d ← 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 )

    for each v ∈ G[V] do v.d ← ∞
    s.d ← 0
    H ← ϕ                                    { empty min-heap }
    for each v ∈ G[V] do INSERT( H, v )
    while H ≠ ∅ do
        u ← EXTRACT-MIN( H )
        for each v ∈ Adj[u] do
            if v.d > u.d + w_{u,v} then
                DECREASE-KEY( H, v, u.d + w_{u,v} )
                v.d ← u.d + w_{u,v}
```

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

\# INSERTS $= n$

\# EXTRACT-MINS $= n$

\# DECREASE-KEYS $\leq m$

Total cost

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

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.

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

    for each v ∈ G[V] do v.d ← ∞
    s.d ← 0
    H ← ϕ                                  { empty min-heap }
    for each v ∈ G[V] do INSERT( H, v )
    while H ≠ ∅ do
        u ← EXTRACT-MIN( H )
        for each v ∈ Adj[u] do
            if v.d > u.d + w_{u,v} then
                DECREASE-KEY( H, v, u.d + w_{u,v} )
                v.d ← u.d + w_{u,v}
```

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

For Binary Heap (worst-case costs):

$$cost_{Insert} = O(\log n)$$
$$cost_{Extract-Min} = O(\log n)$$
$$cost_{Decrease-Key} = O(\log n)$$

∴ Total cost (worst-case)

$$= O\big((m+n)\log n\big)$$

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 )
    for each v ∈ G[V] do v.d ← ∞
    s.d ← 0
    H ← φ                                   { empty min-heap }
    for each v ∈ G[V] do INSERT( H, v )
    while H ≠ ∅ do
        u ← EXTRACT-MIN( H )
        for each v ∈ Adj[u] do
            if v.d > u.d + w_{u,v} then
                DECREASE-KEY( H, v, u.d + w_{u,v} )
                v.d ← u.d + w_{u,v}
```

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

For Binomial Heap (amortized costs):

$$cost_{Insert} = O(1)$$
$$cost_{Extract-Min} = O(\log n)$$
$$cost_{Decrease-Key} = O(\log n)$$

(worst-case)

$\therefore$ Total cost (worst-case)

$$= O\big((m + n)\log n\big)$$

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 )

    for each v ∈ G[V] do v.d ← ∞
    s.d ← 0
    H ← ϕ                                   { empty min-heap }
    for each v ∈ G[V] do INSERT( H, v )
    while H ≠ ∅ do
        u ← EXTRACT-MIN( H )
        for each v ∈ Adj[u] do
            if v.d > u.d + w_{u,v} then
                DECREASE-KEY( H, v, u.d + w_{u,v} )
                v.d ← u.d + w_{u,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.

∴ 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.

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

    for each v ∈ G[V] do v.d ← ∞
    s.d ← 0
    H ← φ                                  { empty min-heap }
    for each v ∈ G[V] do INSERT( H, v )
    while H ≠ ∅ do
        u ← EXTRACT-MIN( H )
        for each v ∈ Adj[u] do
            if v.d > u.d + w_{u,v} then
                DECREASE-KEY( H, v, u.d + w_{u,v} )
                v.d ← u.d + w_{u,v}
```

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

Total cost

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

Observation:

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

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

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 DECREASE-KEY(H, x, k)

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.

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 EXTRACT-MIN in binomial heaps was highly dependent on this exponential relationship, that analysis will no longer hold.

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*: $f_n = \begin{cases} 0 & if\ n = 0, \\ 1 & if\ n = 1, \\ f_{n-1} + f_{n-2} & otherwise. \end{cases}$

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

where $\phi = \frac{1+\sqrt{5}}{2}$ and $\hat{\phi} = \frac{1-\sqrt{5}}{2}$ are the roots $z^2 - z - 1 = 0$.

Analysis of Fibonacci Heap Operations

f_0	0	<	1	$1 + f_0$
f_1	1	<	2	$1 + f_0 + f_1$
f_2	1	<	3	$1 + f_0 + f_1 + f_2$
f_3	2	<	5	$1 + f_0 + f_1 + f_2 + f_3$
f_4	3	<	8	$1 + f_0 + f_1 + f_2 + f_3 + f_4$
f_5	5	<	13	$1 + f_0 + f_1 + f_2 + f_3 + f_4 + f_5$
f_6	8	<	21	$1 + f_0 + f_1 + f_2 + f_3 + f_4 + f_5 + f_6$
f_7	13	<	34	$1 + f_0 + f_1 + f_2 + f_3 + f_4 + f_5 + f_6 + f_7$
f_8	21	<	55	$1 + f_0 + f_1 + f_2 + f_3 + f_4 + f_5 + f_6 + f_7 + f_8$
f_9	34	<	89	$1 + f_0 + f_1 + f_2 + f_3 + f_4 + f_5 + f_6 + f_7 + f_8 + f_9$
f_{10}	55	<	144	$1 + f_0 + f_1 + f_2 + f_3 + f_4 + f_5 + f_6 + f_7 + f_8 + f_9 + f_{10}$

Analysis of Fibonacci Heap Operations

f_0	0			
f_1	1	$=$	1	$1 + f_0$
f_2	1	$<$	2	$1 + f_0 + f_1$
f_3	2	$<$	3	$1 + f_0 + f_1 + f_2$
f_4	3	$<$	5	$1 + f_0 + f_1 + f_2 + f_3$
f_5	5	$<$	8	$1 + f_0 + f_1 + f_2 + f_3 + f_4$
f_6	8	$<$	13	$1 + f_0 + f_1 + f_2 + f_3 + f_4 + f_5$
f_7	13	$<$	21	$1 + f_0 + f_1 + f_2 + f_3 + f_4 + f_5 + f_6$
f_8	21	$<$	34	$1 + f_0 + f_1 + f_2 + f_3 + f_4 + f_5 + f_6 + f_7$
f_9	34	$<$	55	$1 + f_0 + f_1 + f_2 + f_3 + f_4 + f_5 + f_6 + f_7 + f_8$
f_{10}	55	$<$	89	$1 + f_0 + f_1 + f_2 + f_3 + f_4 + f_5 + f_6 + f_7 + f_8 + f_9$
f_{11}	89	$<$	144	$1 + f_0 + f_1 + f_2 + f_3 + f_4 + f_5 + f_6 + f_7 + f_8 + f_9 + f_{10}$

Analysis of Fibonacci Heap Operations

f_0	0			
f_1	1			
f_2	1	=	1	$1 + f_0$
f_3	2	=	2	$1 + f_0 + f_1$
f_4	3	=	3	$1 + f_0 + f_1 + f_2$
f_5	5	=	5	$1 + f_0 + f_1 + f_2 + f_3$
f_6	8	=	8	$1 + f_0 + f_1 + f_2 + f_3 + f_4$
f_7	13	=	13	$1 + f_0 + f_1 + f_2 + f_3 + f_4 + f_5$
f_8	21	=	21	$1 + f_0 + f_1 + f_2 + f_3 + f_4 + f_5 + f_6$
f_9	34	=	34	$1 + f_0 + f_1 + f_2 + f_3 + f_4 + f_5 + f_6 + f_7$
f_{10}	55	=	55	$1 + f_0 + f_1 + f_2 + f_3 + f_4 + f_5 + f_6 + f_7 + f_8$
f_{11}	89	=	89	$1 + f_0 + f_1 + f_2 + f_3 + f_4 + f_5 + f_6 + f_7 + f_8 + f_9$
f_{12}	144	=	144	$1 + f_0 + f_1 + f_2 + f_3 + f_4 + f_5 + f_6 + f_7 + f_8 + f_9 + f_{10}$

Lemma 1: For all integers $n \geq 0$, $f_{n+2} = 1 + \sum_{i=0}^{n} f_i$.

Analysis of Fibonacci Heap Operations

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

Analysis of Fibonacci Heap Operations

f_0	0	$<$	1.00	ϕ^0
f_1	1	$<$	1.62	ϕ^1
f_2	1	$<$	2.62	ϕ^2
f_3	2	$<$	4.24	ϕ^3
f_4	3	$<$	6.85	ϕ^4
f_5	5	$<$	11.09	ϕ^5
f_6	8	$<$	17.94	ϕ^6
f_7	13	$<$	29.03	ϕ^7
f_8	21	$<$	46.98	ϕ^8
f_9	34	$<$	76.01	ϕ^9
f_{10}	55	$<$	122.99	ϕ^{10}

Analysis of Fibonacci Heap Operations

f_0	0			
f_1	1	$\geq$	1.00	ϕ^0
f_2	1	$<$	1.62	ϕ^1
f_3	2	$<$	2.62	ϕ^2
f_4	3	$<$	4.24	ϕ^3
f_5	5	$<$	6.85	ϕ^4
f_6	8	$<$	11.09	ϕ^5
f_7	13	$<$	17.94	ϕ^6
f_8	21	$<$	29.03	ϕ^7
f_9	34	$<$	46.98	ϕ^8
f_{10}	55	$<$	76.01	ϕ^9
f_{11}	89	$<$	122.99	ϕ^{10}

Analysis of Fibonacci Heap Operations

f_0	0			
f_1	1			
f_2	1	$\geq$	1.00	ϕ^0
f_3	2	$\geq$	1.62	ϕ^1
f_4	3	$\geq$	2.62	ϕ^2
f_5	5	$\geq$	4.24	ϕ^3
f_6	8	$\geq$	6.85	ϕ^4
f_7	13	$\geq$	11.09	ϕ^5
f_8	21	$\geq$	17.94	ϕ^6
f_9	34	$\geq$	29.03	ϕ^7
f_{10}	55	$\geq$	46.98	ϕ^8
f_{11}	89	$\geq$	76.01	ϕ^9
f_{12}	144	$\geq$	122.99	ϕ^{10}

Lemma 2: For all integers $n \geq 0$, $f_{n+2} \geq \phi^n$.

Analysis of Fibonacci Heap Operations

Lemma 2: For all integers $n \geq 0$, $f_{n+2} \geq \phi^n$.

Proof: By induction on n.

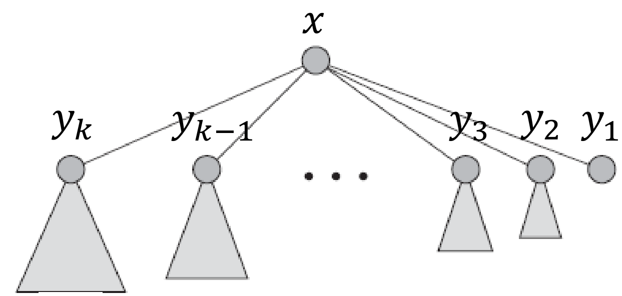
Base case: $f_2 = 1 = \phi^0$ and $f_3 = 2 > \phi^1$.

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

Then
$$
\begin{aligned}
f_{n+2} &= f_{n+1} + f_n \\
&\geq \phi^{n-1} + \phi^{n-2} \\
&= (\phi + 1)\phi^{n-2} \\
&= \phi^2 \phi^{n-2} \\
&= \phi^n
\end{aligned}
$$

Analysis of Fibonacci Heap Operations

Lemma 3: Let x be any node in a Fibonacci heap, and suppose that $k = rank(x)$. Let $y_1, y_2, \ldots, y_k$ 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 $y_1, y_2, \ldots, y_{i-1}$ 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, at most one child has been removed from y_i, and hence $rank(y_i) \geq i - 2$.

Analysis of Fibonacci Heap Operations

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

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

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 $\mathrm{O}(\log n)$.

Proof: Let z be any node in the heap.

Then from Lemma 4,

$$degree(z) = rank(z) \leq \log_{\phi}\big(size(z)\big) \leq \log_{\phi} n = \mathrm{O}(\log n).$$

Analysis of Fibonacci Heap Operations

All nodes are initially unmarked.

We mark a node when

- a child is removed from it for the first time

We unmark a node when

- a child is removed from it for the second time, or
- becomes the child of another node (e.g., LINKed)

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($\boldsymbol{H}, \boldsymbol{x}, \boldsymbol{k_x}$): Let $k =$ #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.

∴ 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($\boldsymbol{H, x, k_x}$):

New trees: 1 tree rooted at x, and

1 tree produced by each of the k cascading cuts.

$\therefore t(D_i) - t(D_{i-1}) = 1 + k$

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

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

$\therefore m(D_i) - m(D_{i-1}) \leq -k + 1$

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

$$= 2\big(t(D_i) - t(D_{i-1})\big) + 3\big(m(D_i) - m(D_{i-1})\big)$$

$$\leq 2(1 + k) + 3(-k + 1)$$

$$= -k + 5$$

Fibonacci Heaps from Binomial Heaps

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

DECREASE-KEY(H, x, k_x):

$$
\begin{aligned}
\text{Amortized cost, } \hat{c}_i &= c_i + \Delta_i \\
&\leq (1 + k) + (-k + 5) \\
&= 6 \\
&= \mathrm{O}(1)
\end{aligned}
$$

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.

The amortized cost of performing EXTRACT-MIN on the array version and that of converting from the array version to the doubly linked list version both can be easily shown to be $\mathrm{O}(d_n) = \mathrm{O}(\log n)$.

Hence, here we will only analyze the amortized cost of converting from the doubly linked list version to the array version.

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.

Fibonacci Heaps from Binomial Heaps

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

EXTRACT-MIN(H):

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

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

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

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

So, $\hat{c}_i \leq 2d_n + 2 = \mathrm{O}(\log n)$.

Fibonacci Heaps from Binomial Heaps

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

DELETE($\boldsymbol{H, x}$):

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

STEP 2: EXTRACT-MIN(H)

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

+ amortized cost of EXTRACT-MIN

$= \mathrm{O}(1) + \mathrm{O}(\log n)$

$= \mathrm{O}(\log n)$