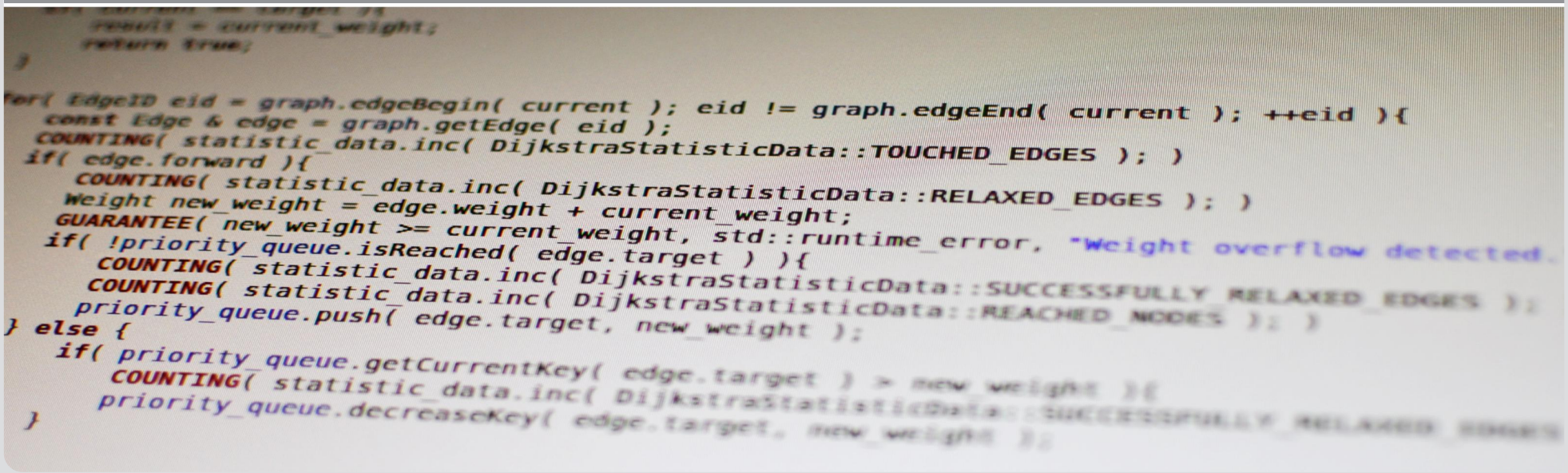
Karlsruher Institut für Technologie

Übung 1 – Algorithmen II

Daniel Seemaier, Tobias Heuer – *daniel.seemaier@kit.edu, tobias.heuer@kit.edu*

`http://algo2.iti.kit.edu/AlgorithmenII_WS20.php`

Institut für Theoretische Informatik - Algorithmik II



Organisatorisches

Vorlesungen:

Mo 10:00–11:30 Online – Q&A mit Prof. Dr. Peter Sanders
Di 16:00–17:30 Online – Übung

Übungsblätter:

14-tägig, jeweils Montags, Musterlösung wird zusammen mit nächsten Übungsblatt veröffentlicht
1. Blatt: 09.11.2020

Ilias Forum:

Fragen zum Vorlesungsinhalt (auch anonym möglich)
Aufzeichnungen der Zoom Online Vorlesungen

Vorlesungsaufzeichnung:

Vorlesung auf Youtube – (Link: *Algorithmen II - Playlist*)
In der Regel werden 2 Vorlesungen gegen Ende jeder Woche hochgeladen

Organisatorisches

Sprechstunden:

Corona bedingt: Ilias Forum oder E-Mail bevorzugt

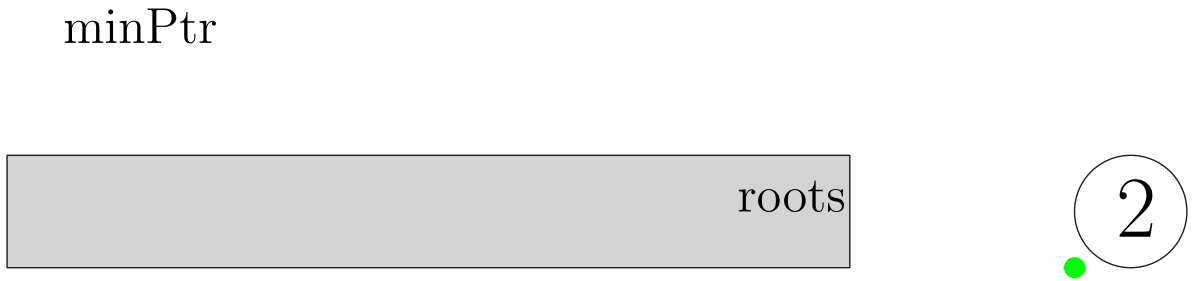
Letzte Vorlesung:

16. Februar 2021

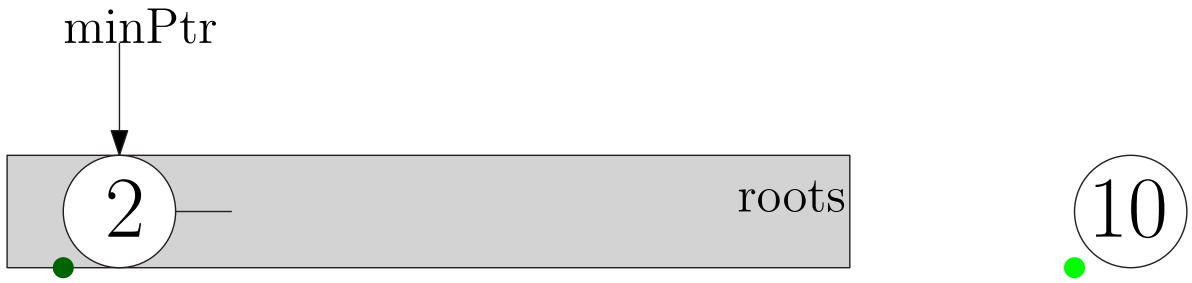
Klausur:

Wird auf der Vorlesung-Homepage noch bekannt gegeben

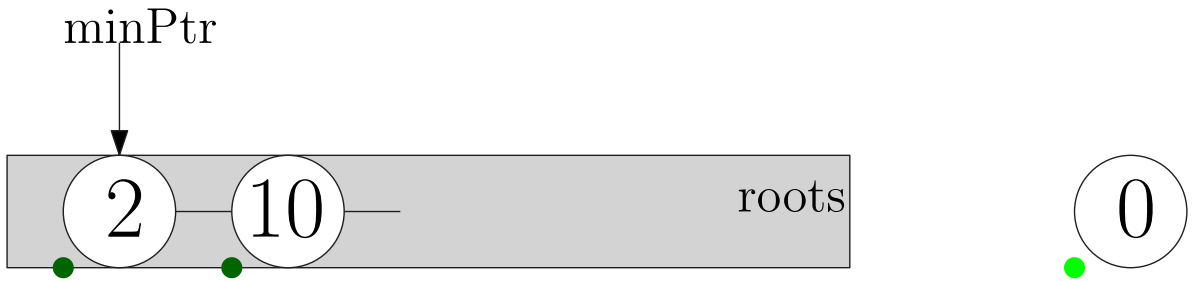
Fibonacci Heaps - Insert



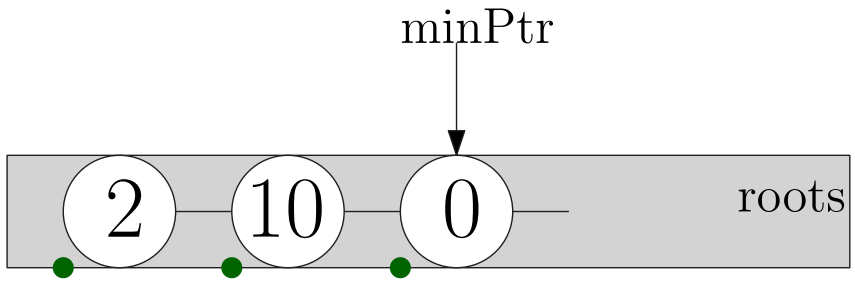
Fibonacci Heaps - Insert



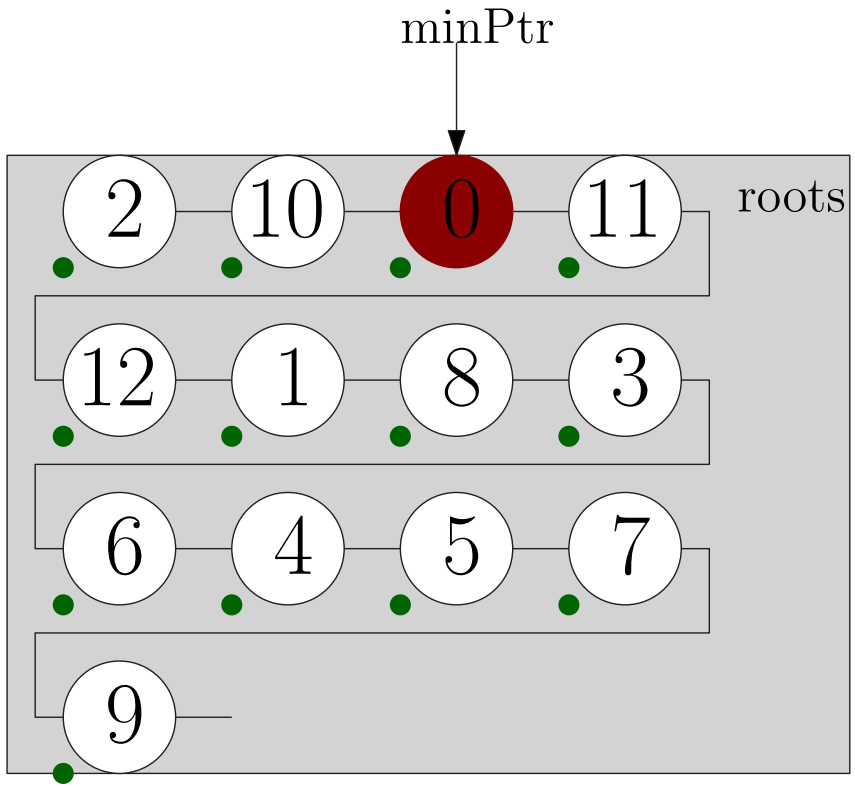
Fibonacci Heaps - Insert



Fibonacci Heaps - Insert



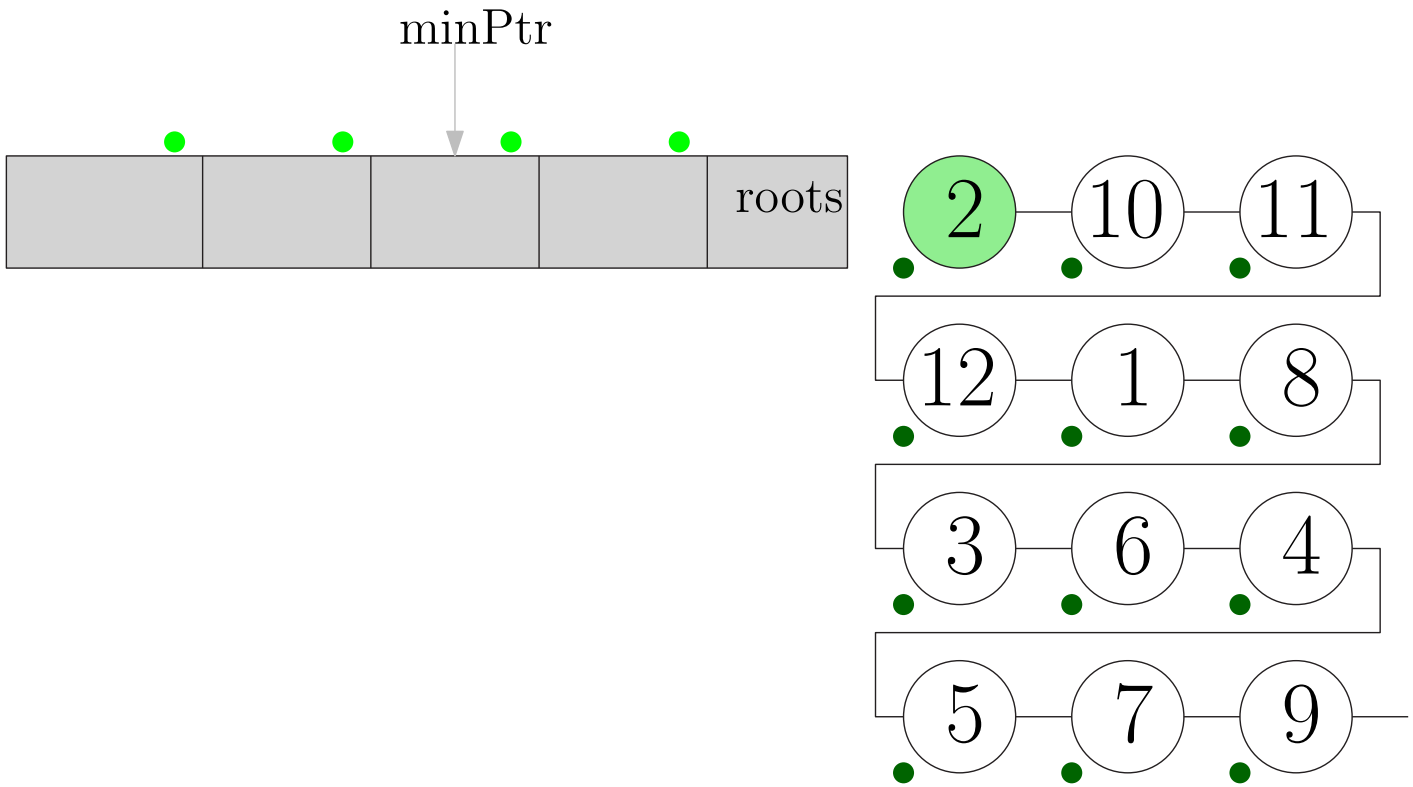
Fibonacci Heaps - Insert



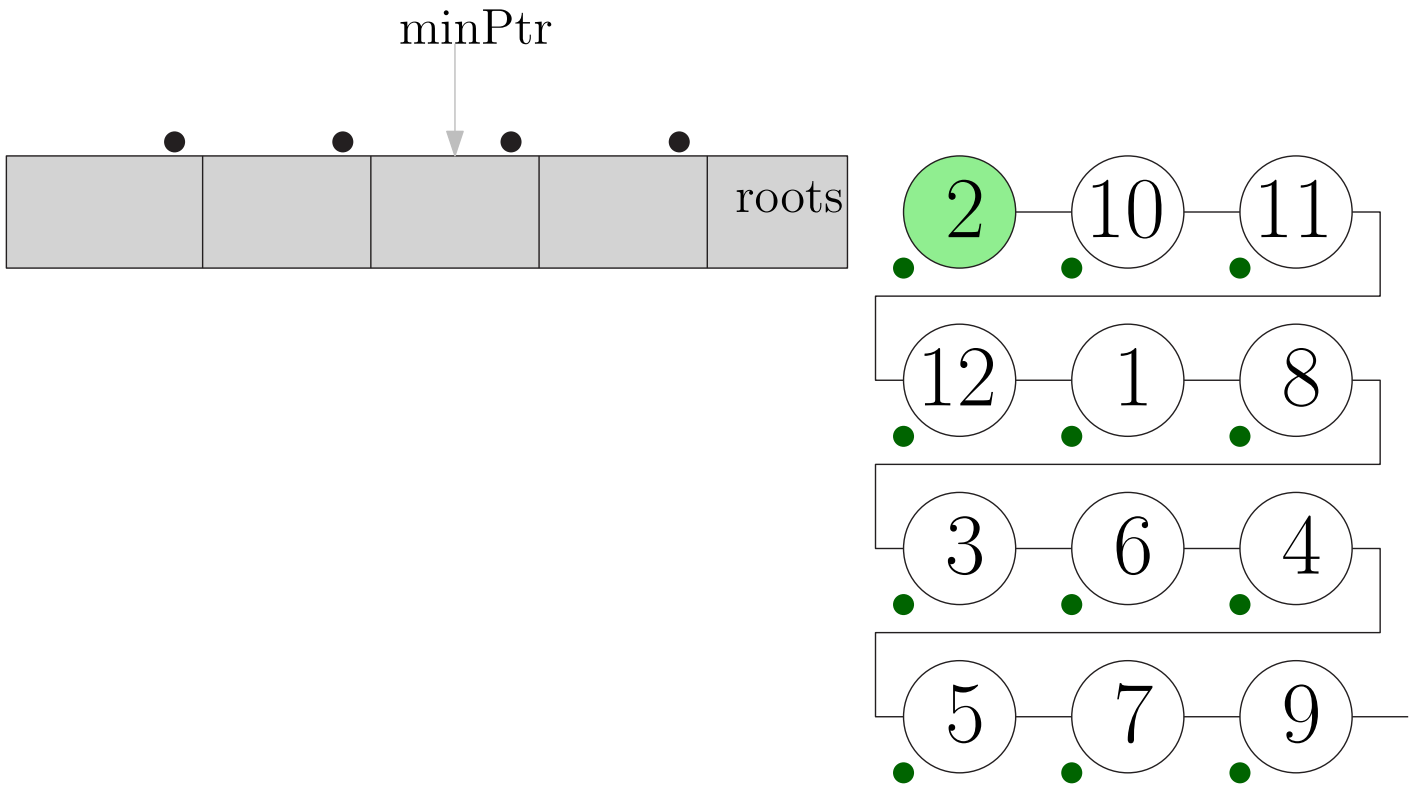
Fibonacci Heaps - Insert

- Insert in konstanter Zeit
- Erzeugt großen Wald einzelner Knoten
- Entspricht ohne **union** nur linearer Liste

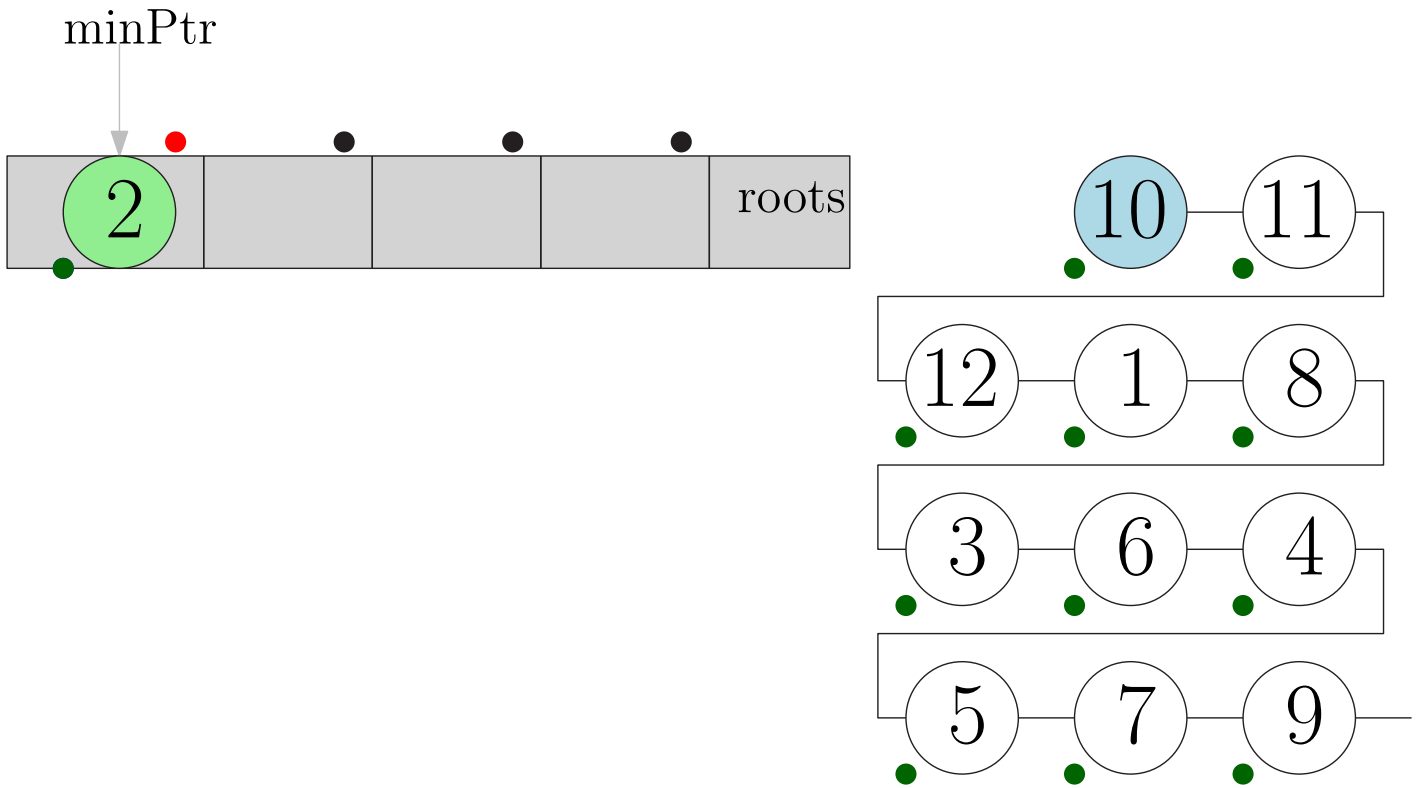
Fibonacci Heaps - Delete Min



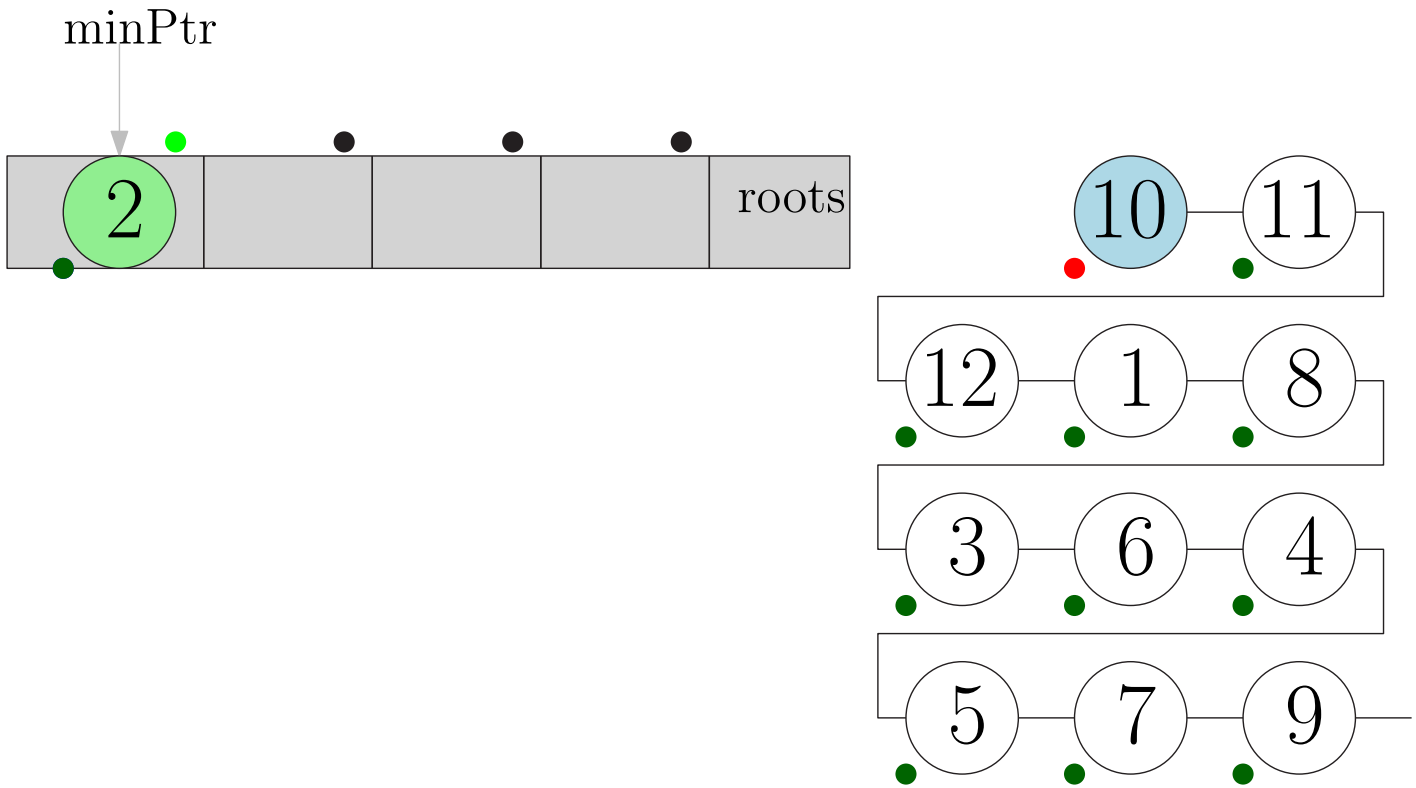
Fibonacci Heaps - Delete Min



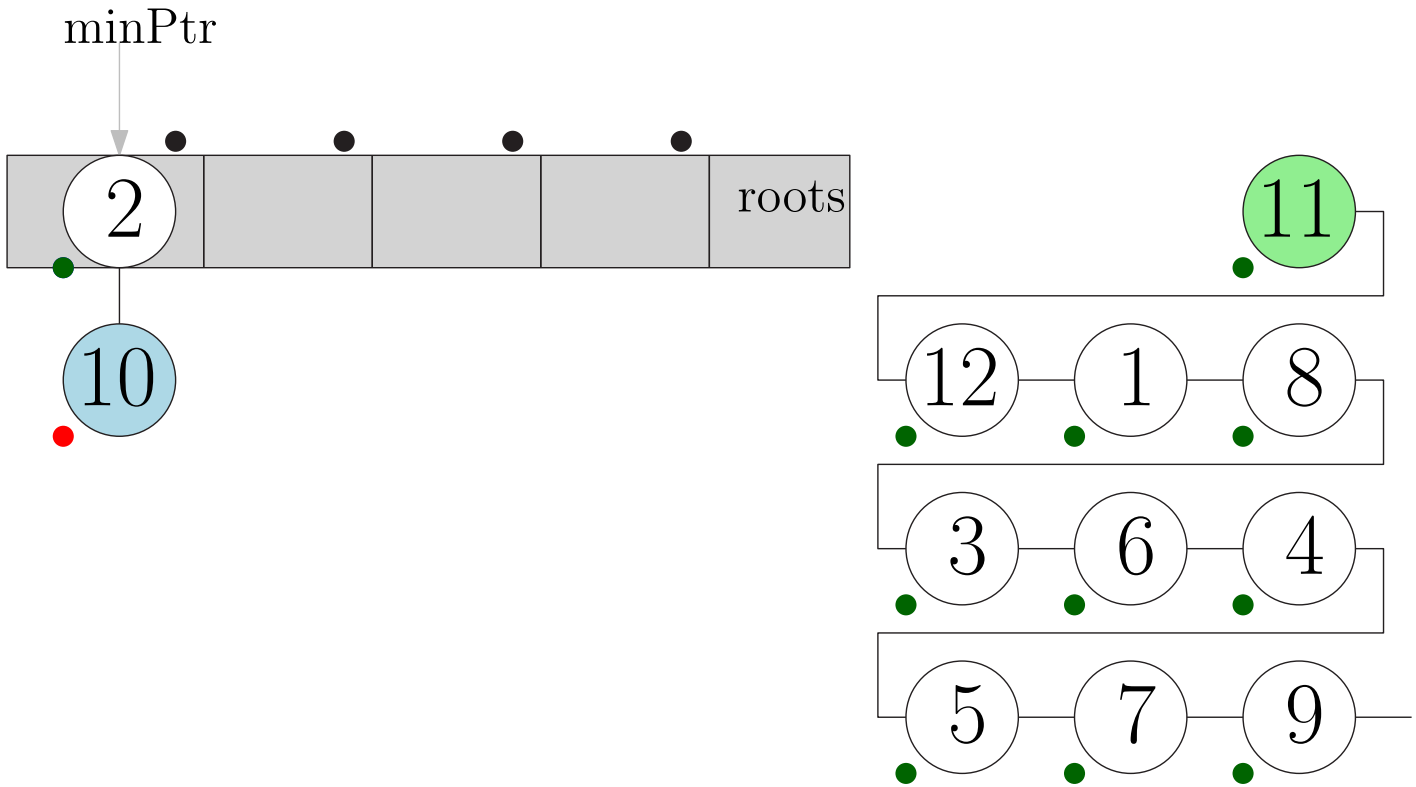
Fibonacci Heaps - Delete Min



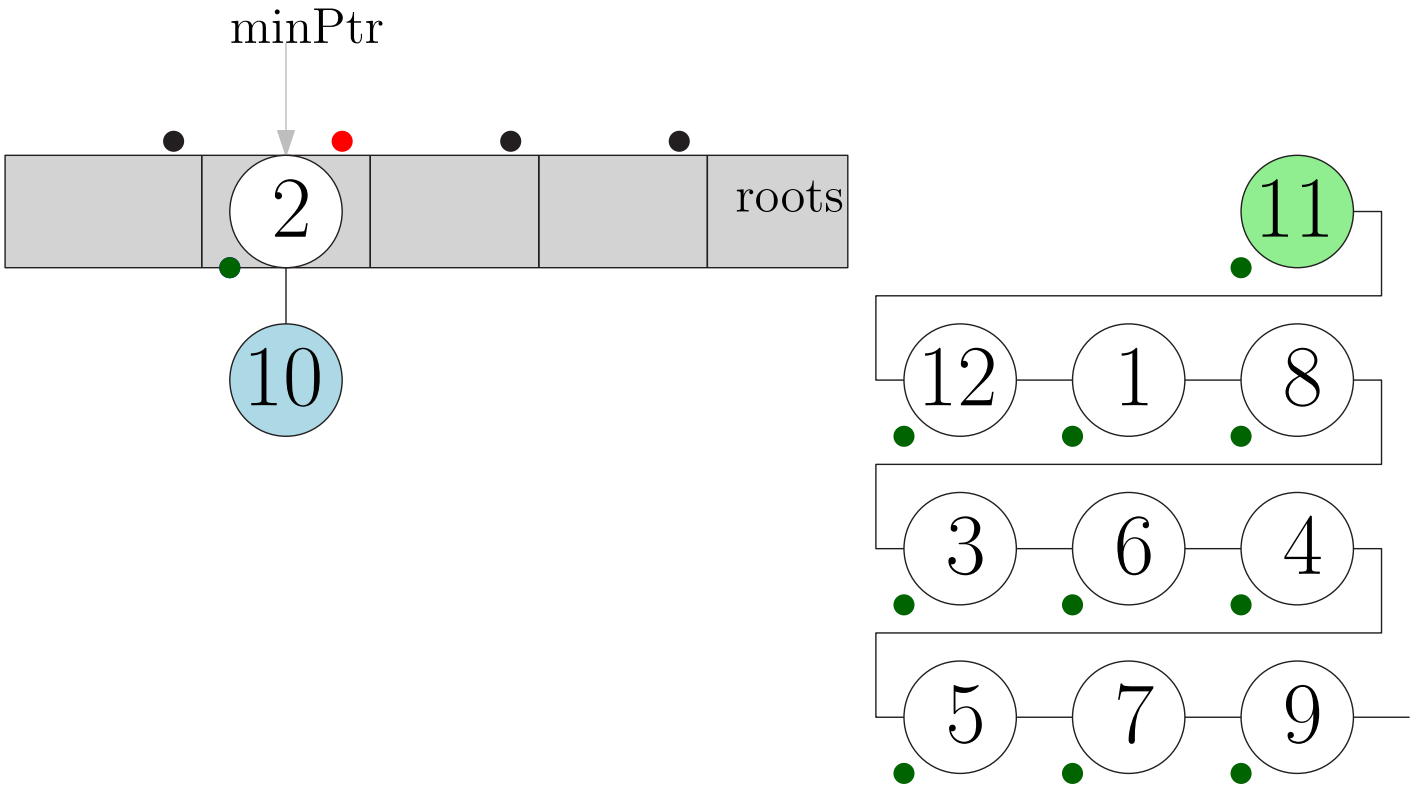
Fibonacci Heaps - Delete Min



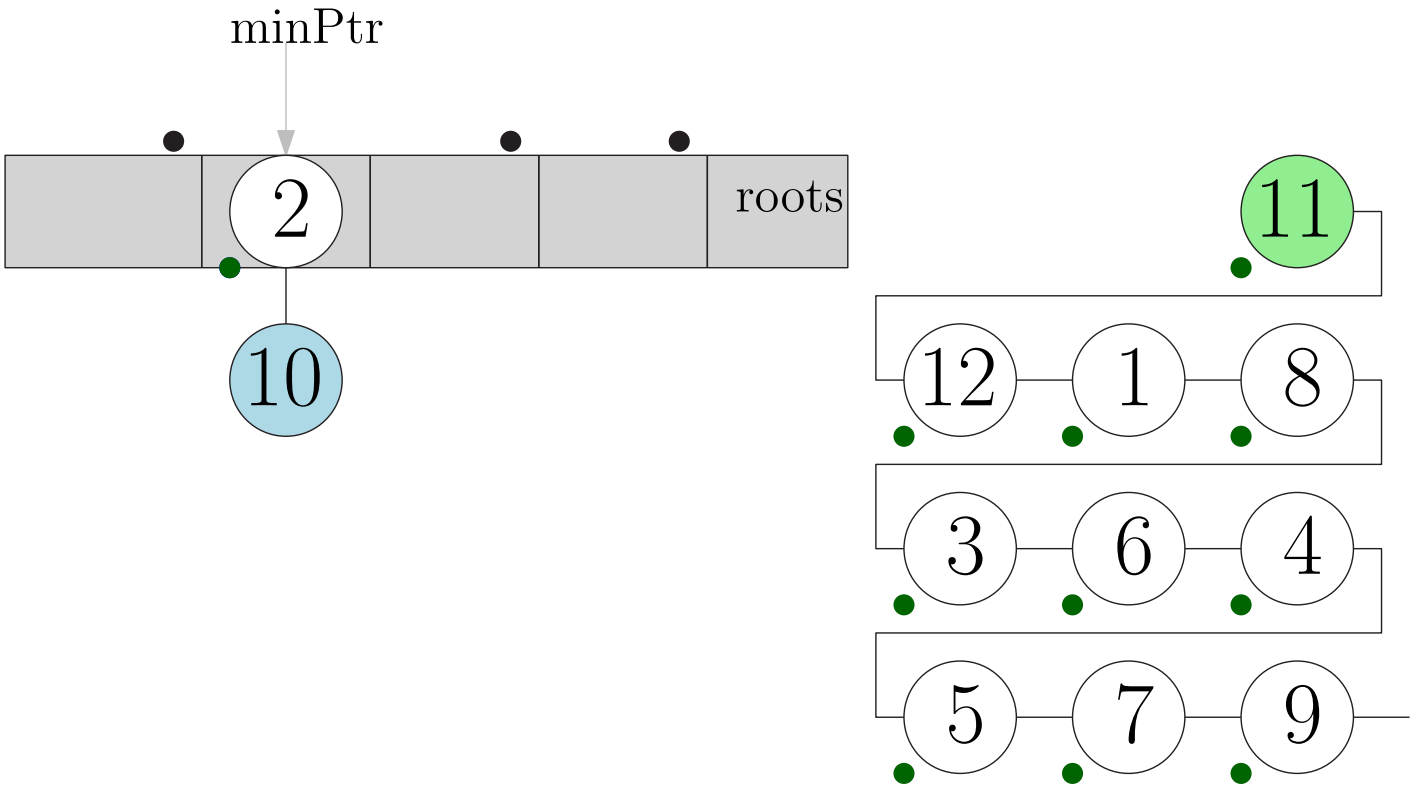
Fibonacci Heaps - Delete Min



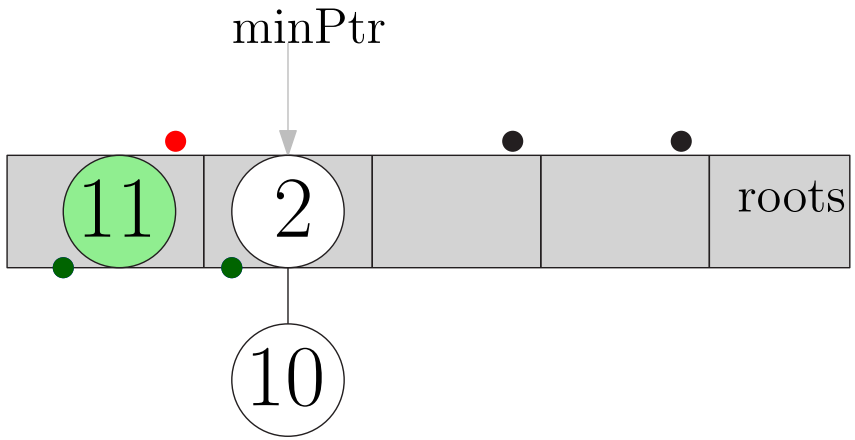
Fibonacci Heaps - Delete Min

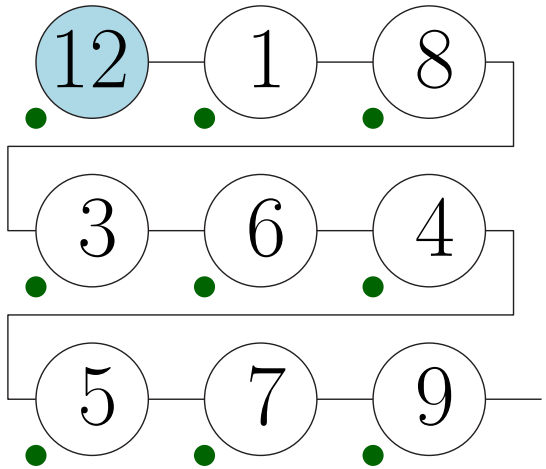


Fibonacci Heaps - Delete Min

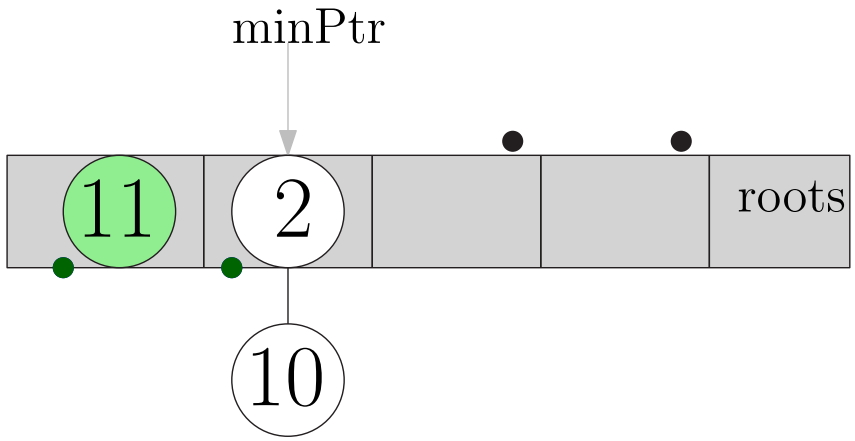


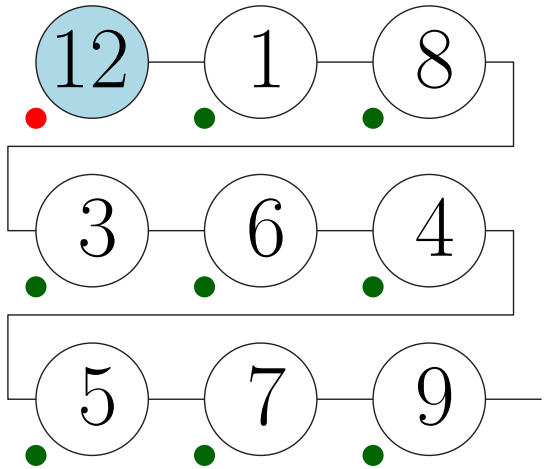
Fibonacci Heaps - Delete Min



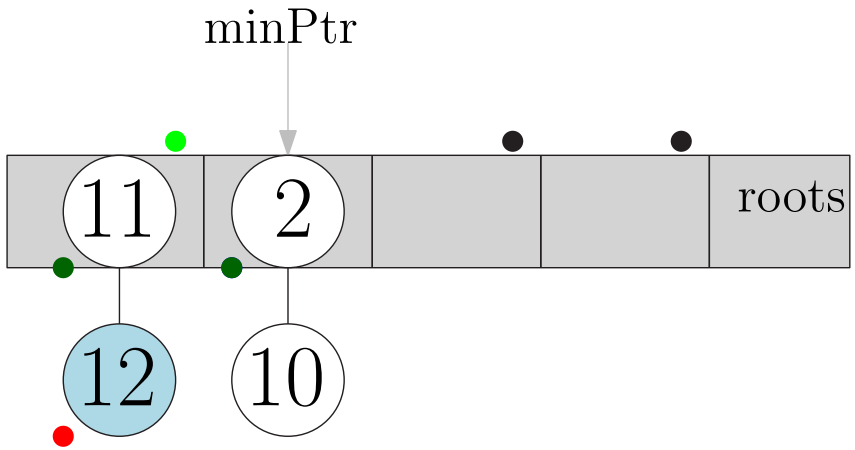


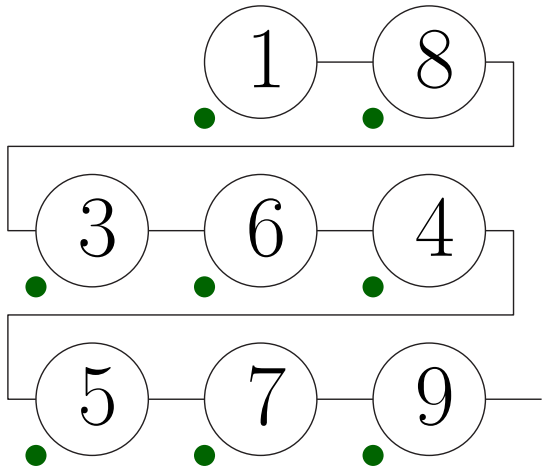
Fibonacci Heaps - Delete Min



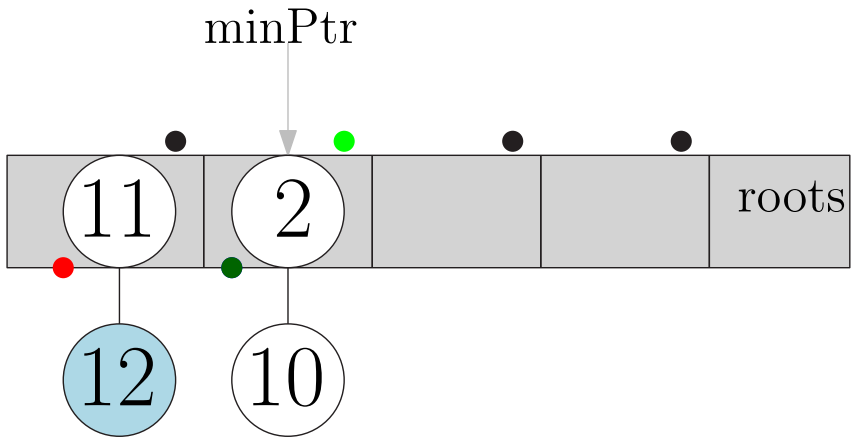


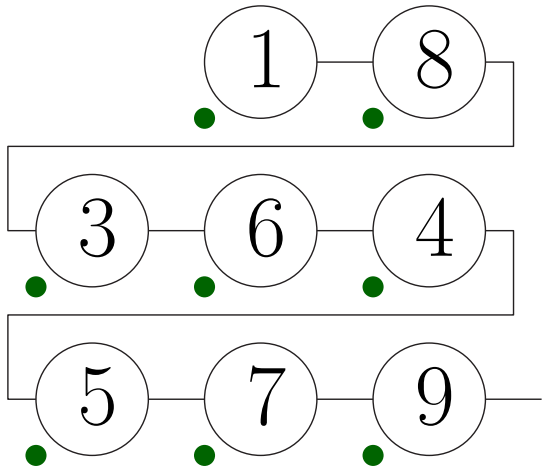
Fibonacci Heaps - Delete Min



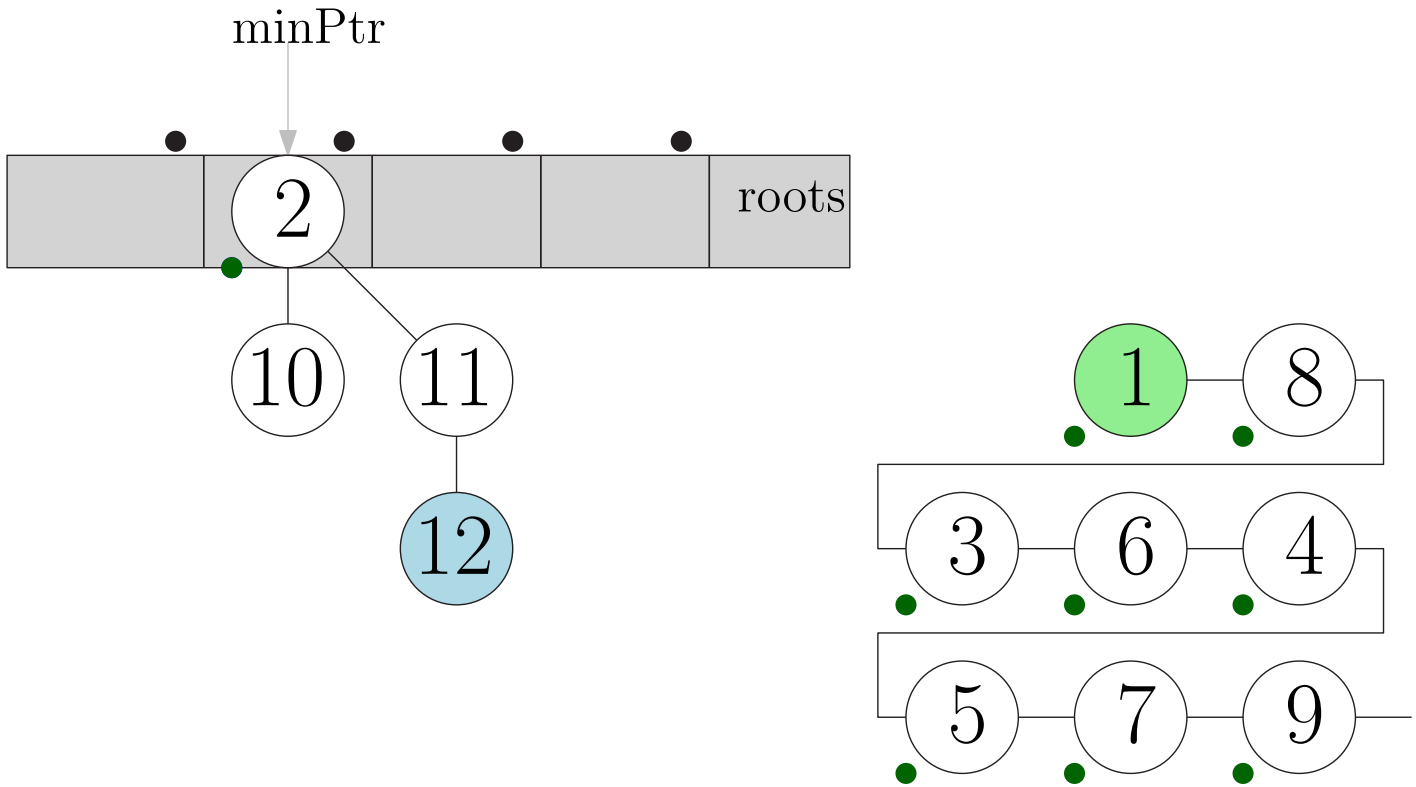


Fibonacci Heaps - Delete Min

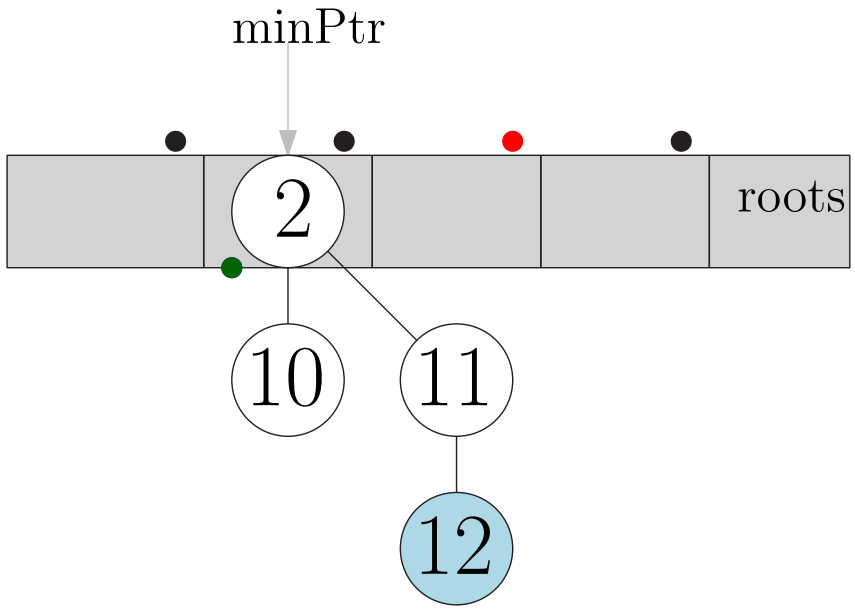


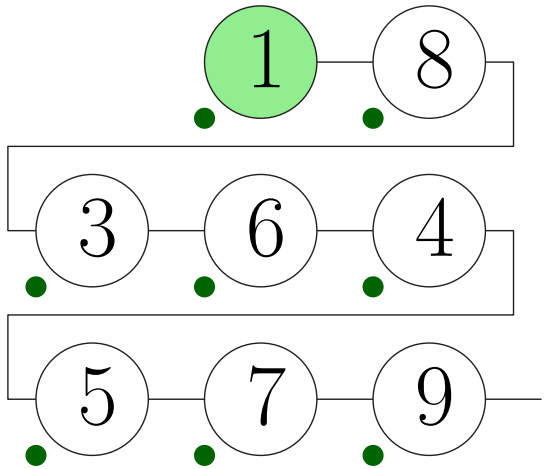


Fibonacci Heaps - Delete Min

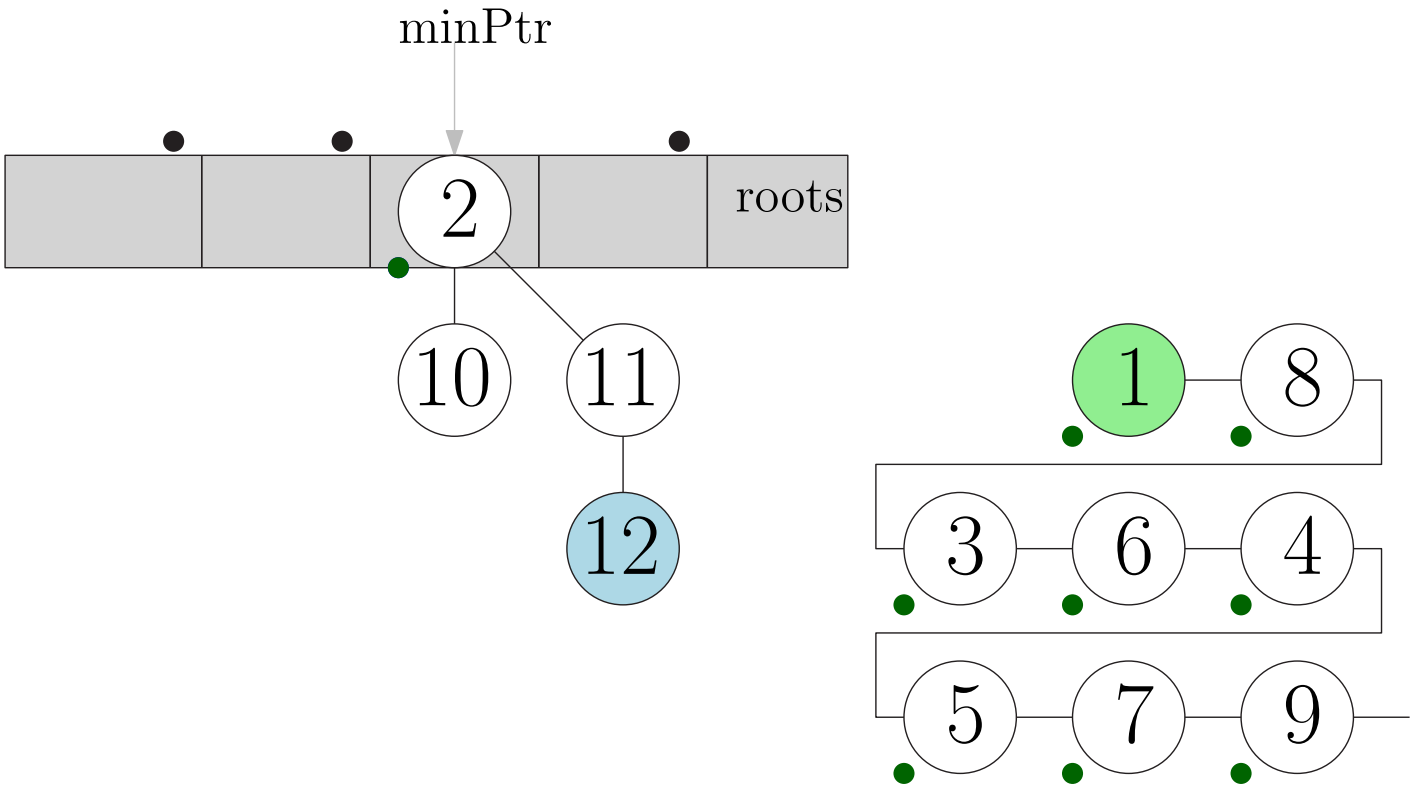


Fibonacci Heaps - Delete Min

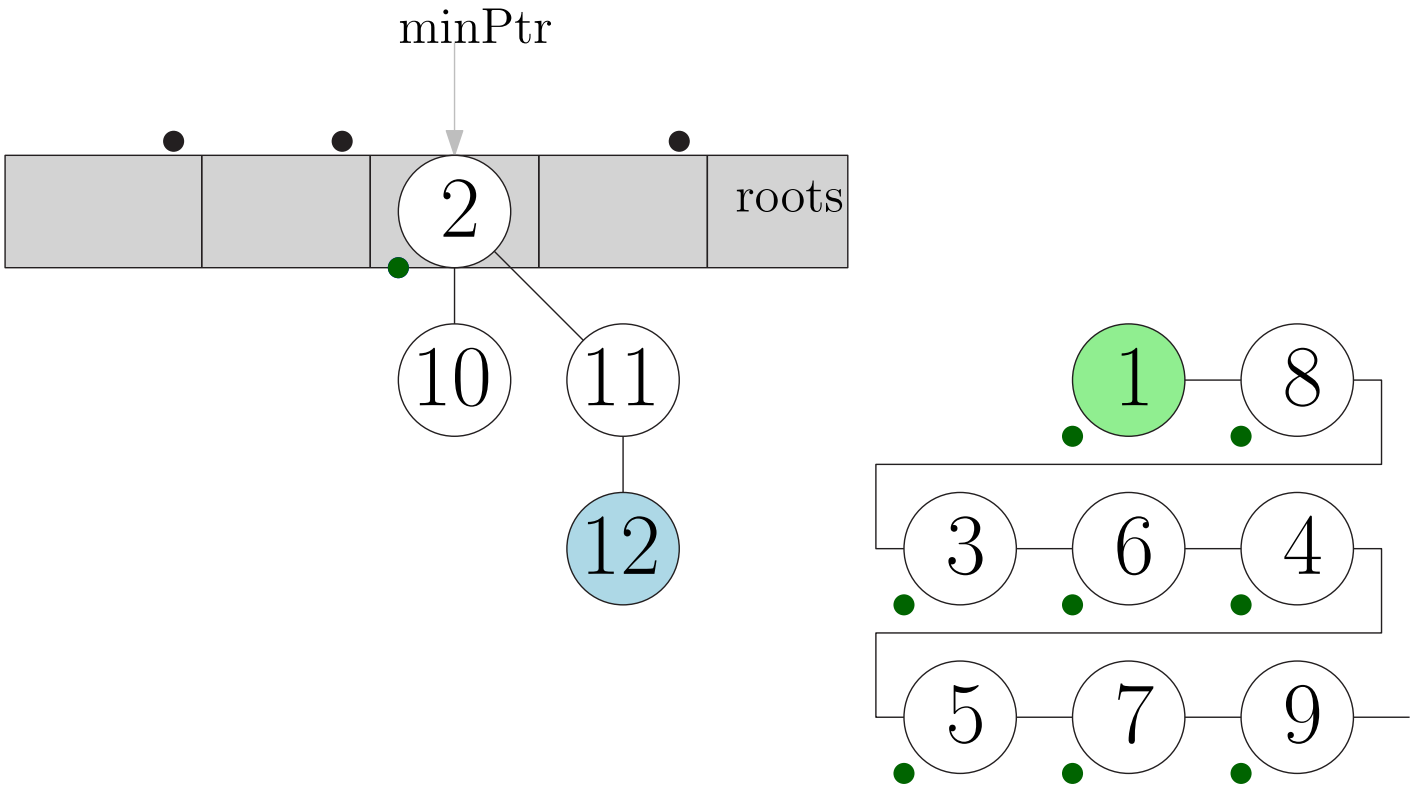




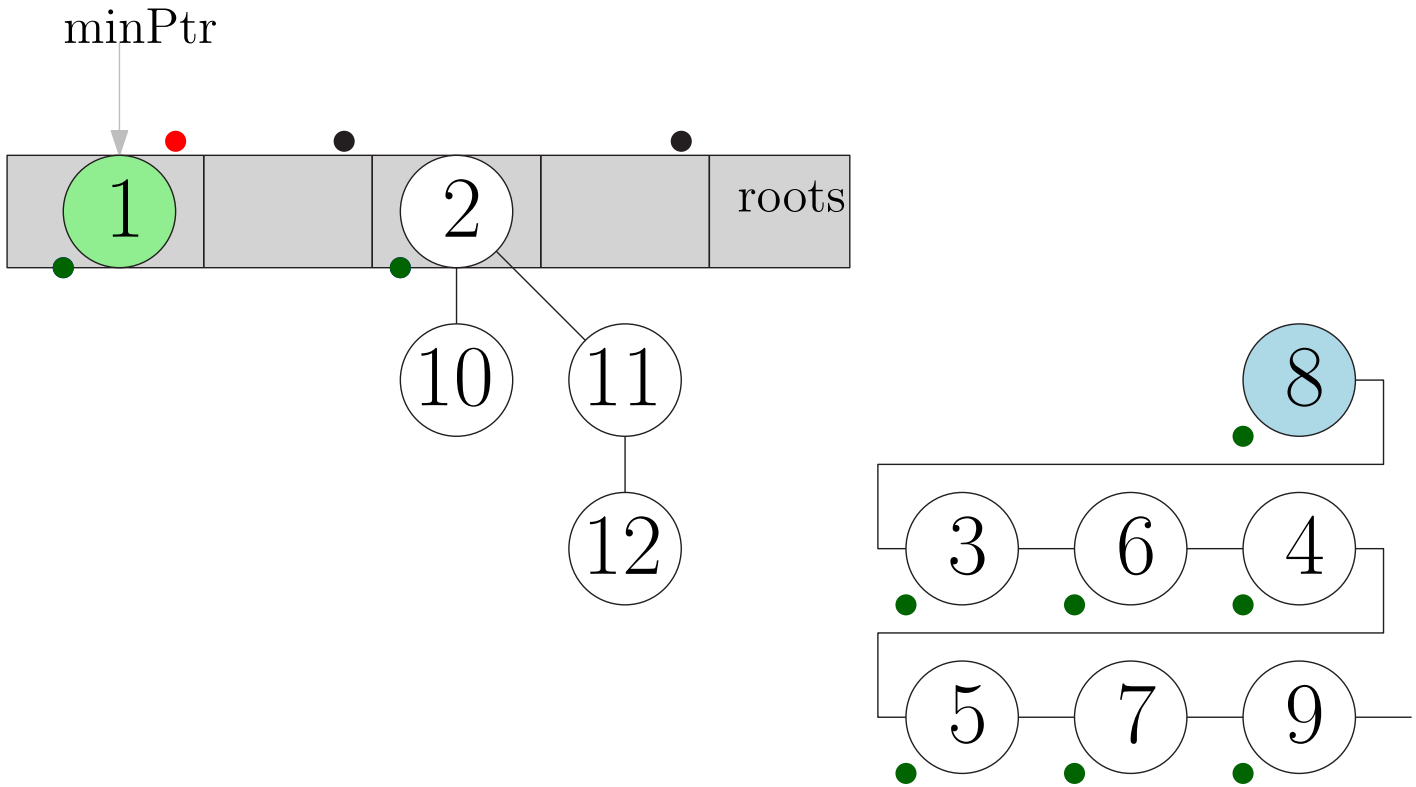
Fibonacci Heaps - Delete Min



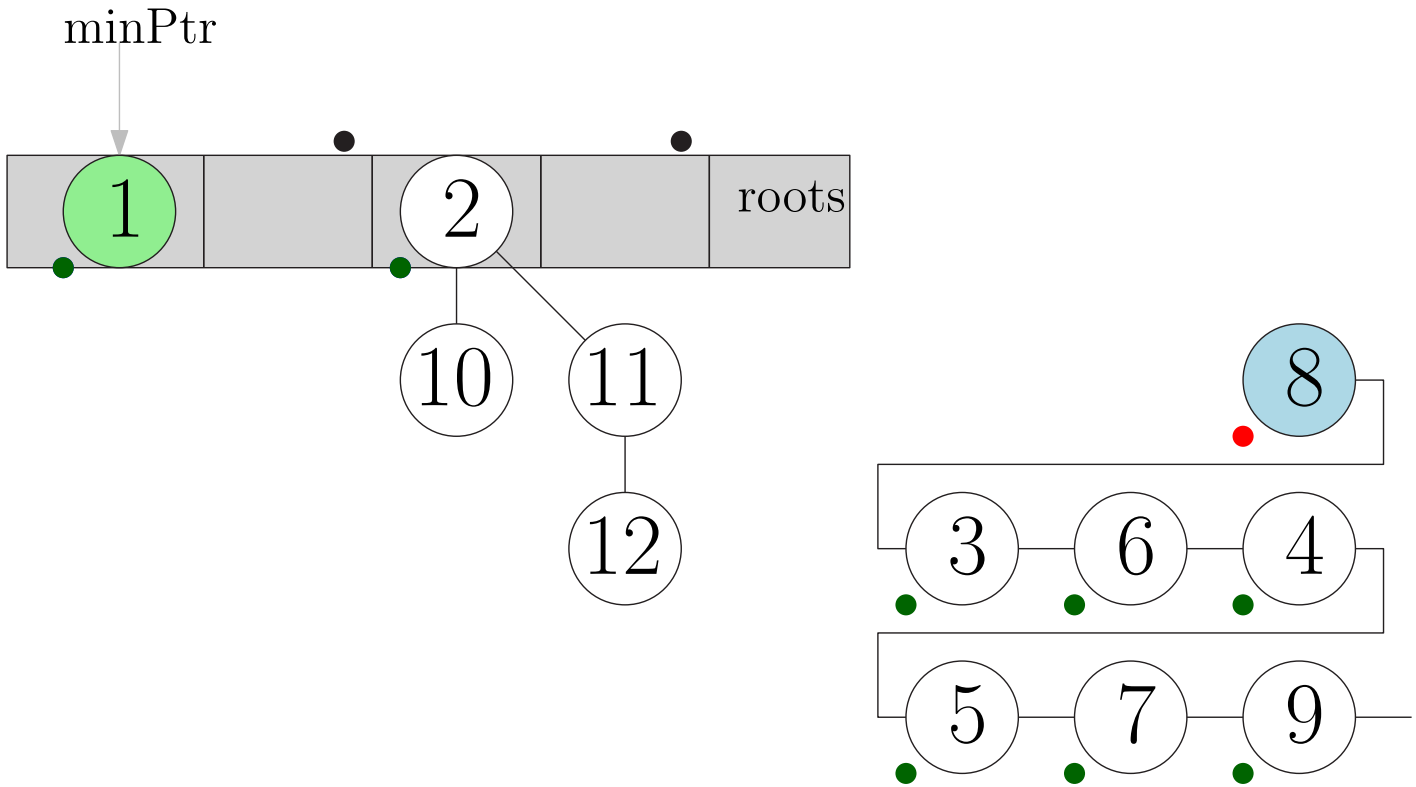
Fibonacci Heaps - Delete Min



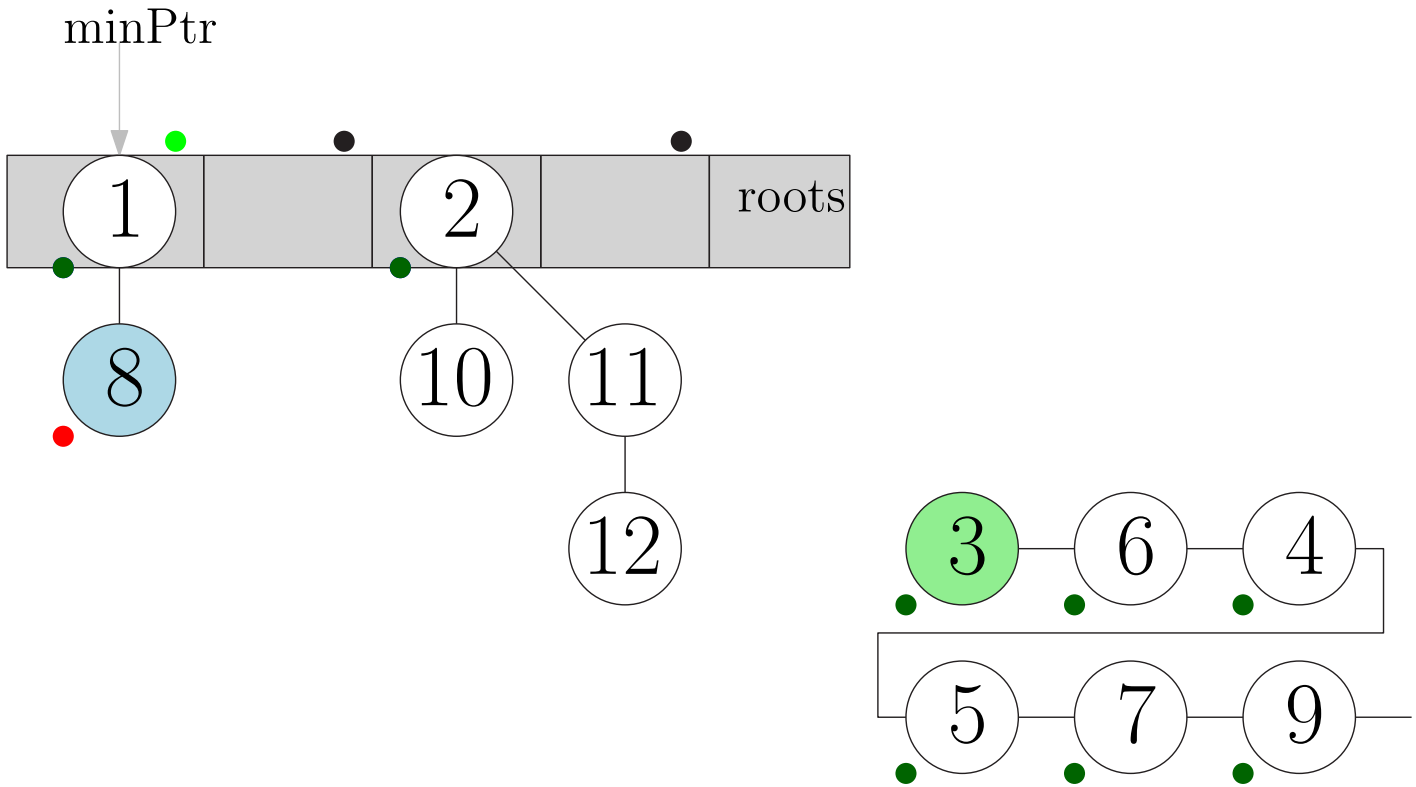
Fibonacci Heaps - Delete Min



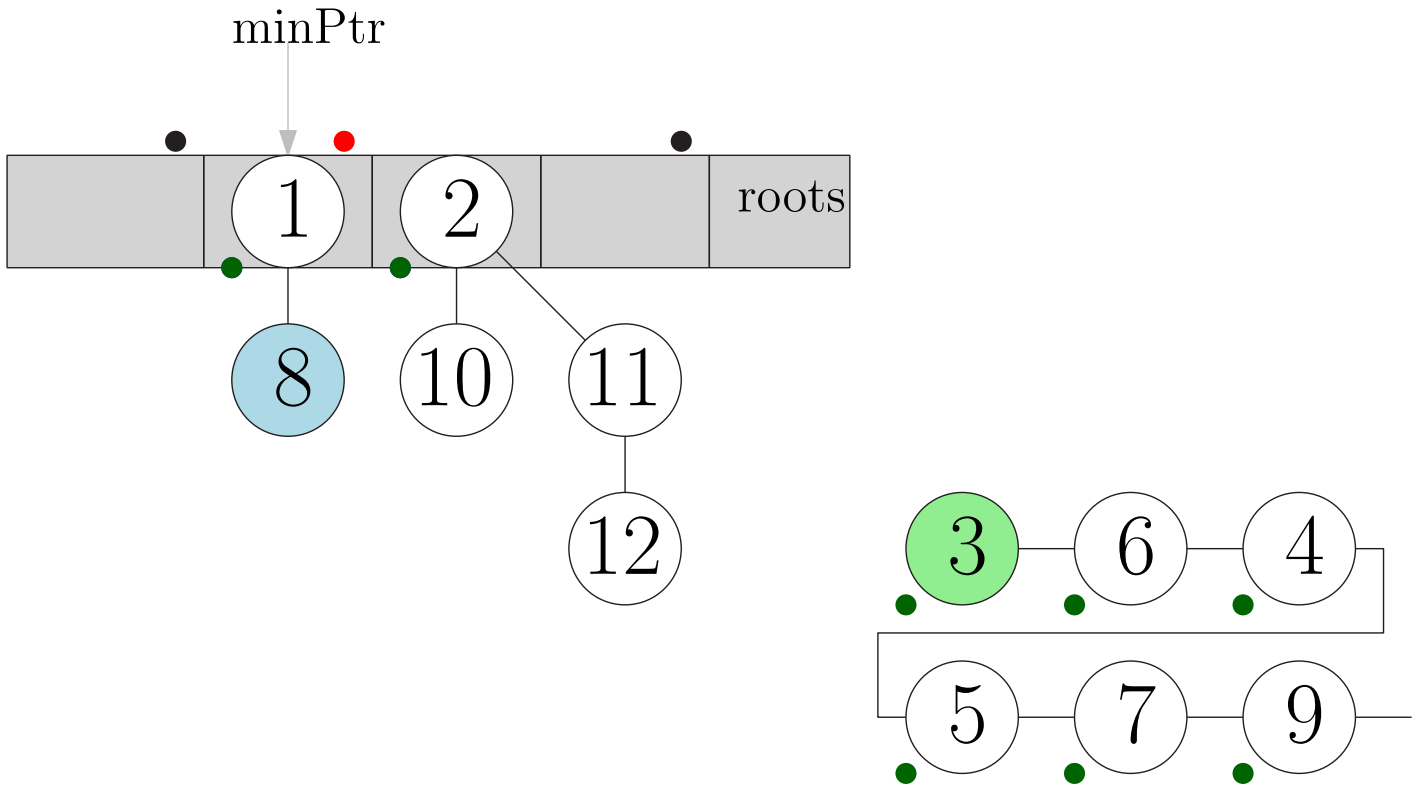
Fibonacci Heaps - Delete Min



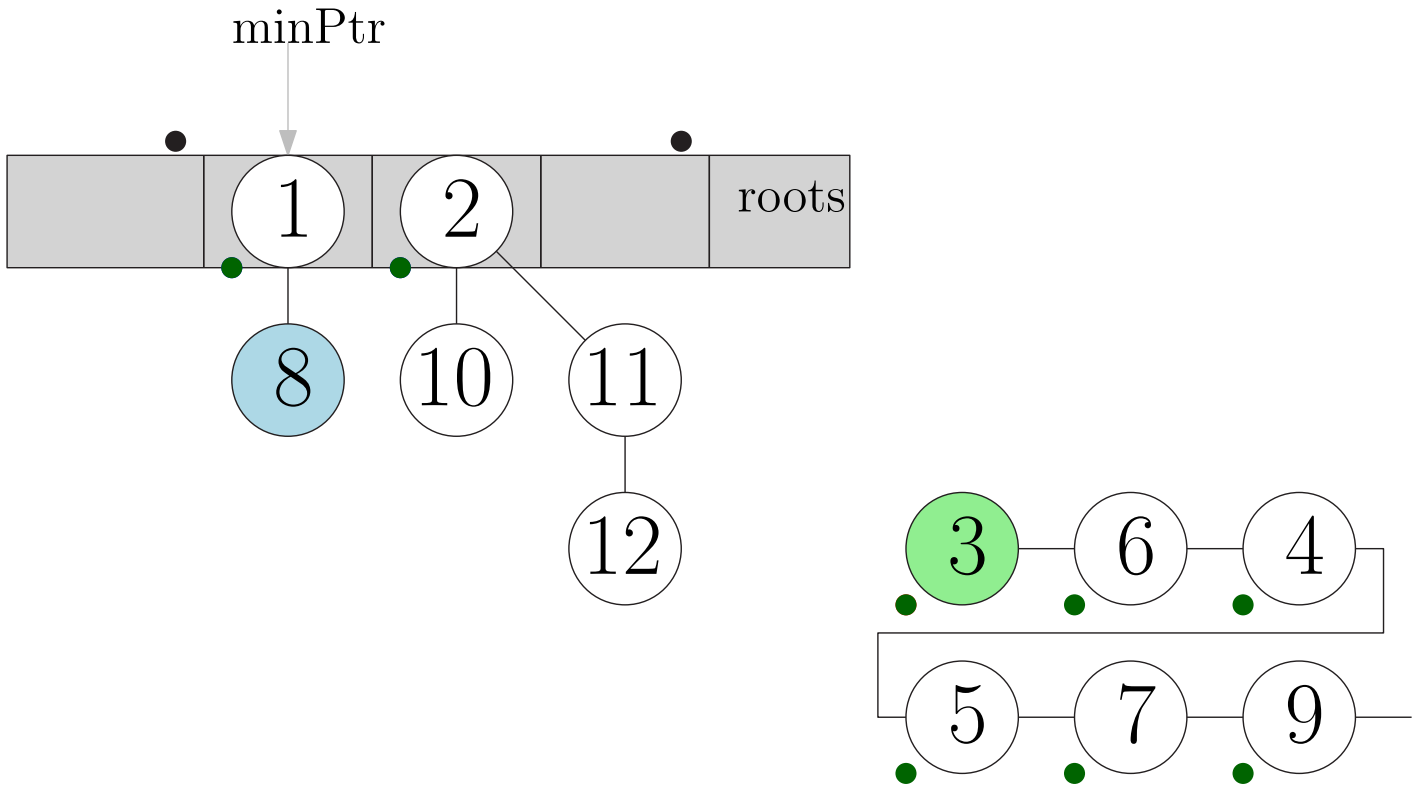
Fibonacci Heaps - Delete Min



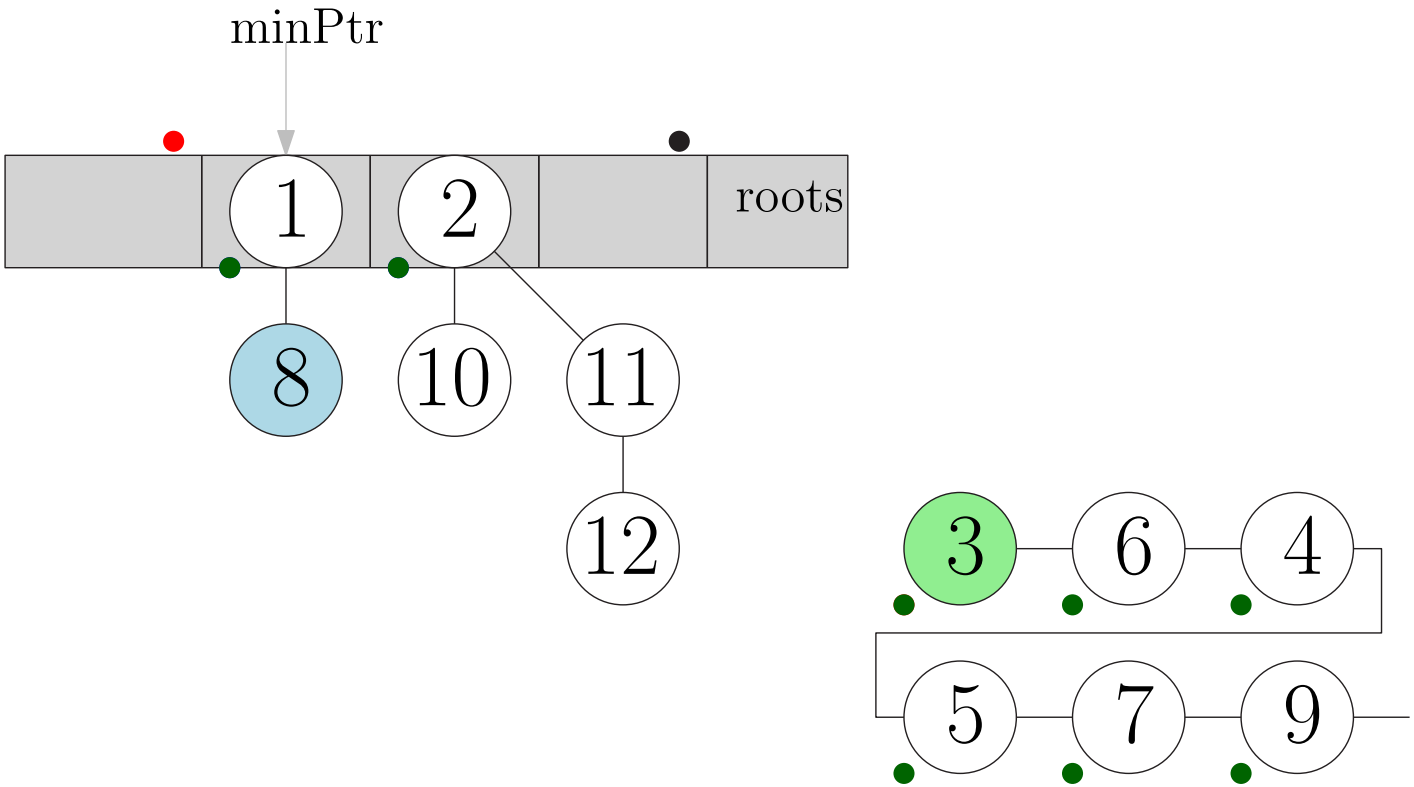
Fibonacci Heaps - Delete Min



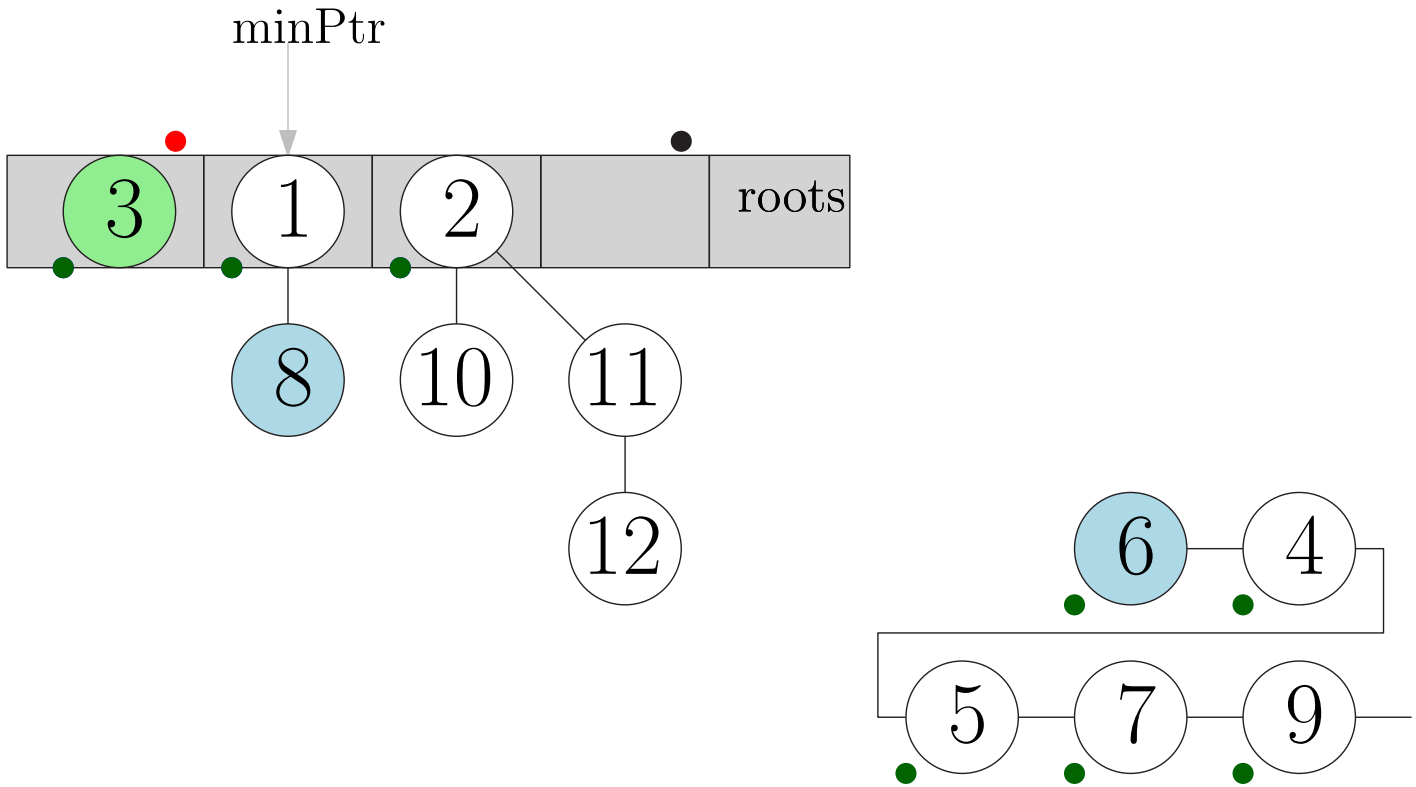
Fibonacci Heaps - Delete Min



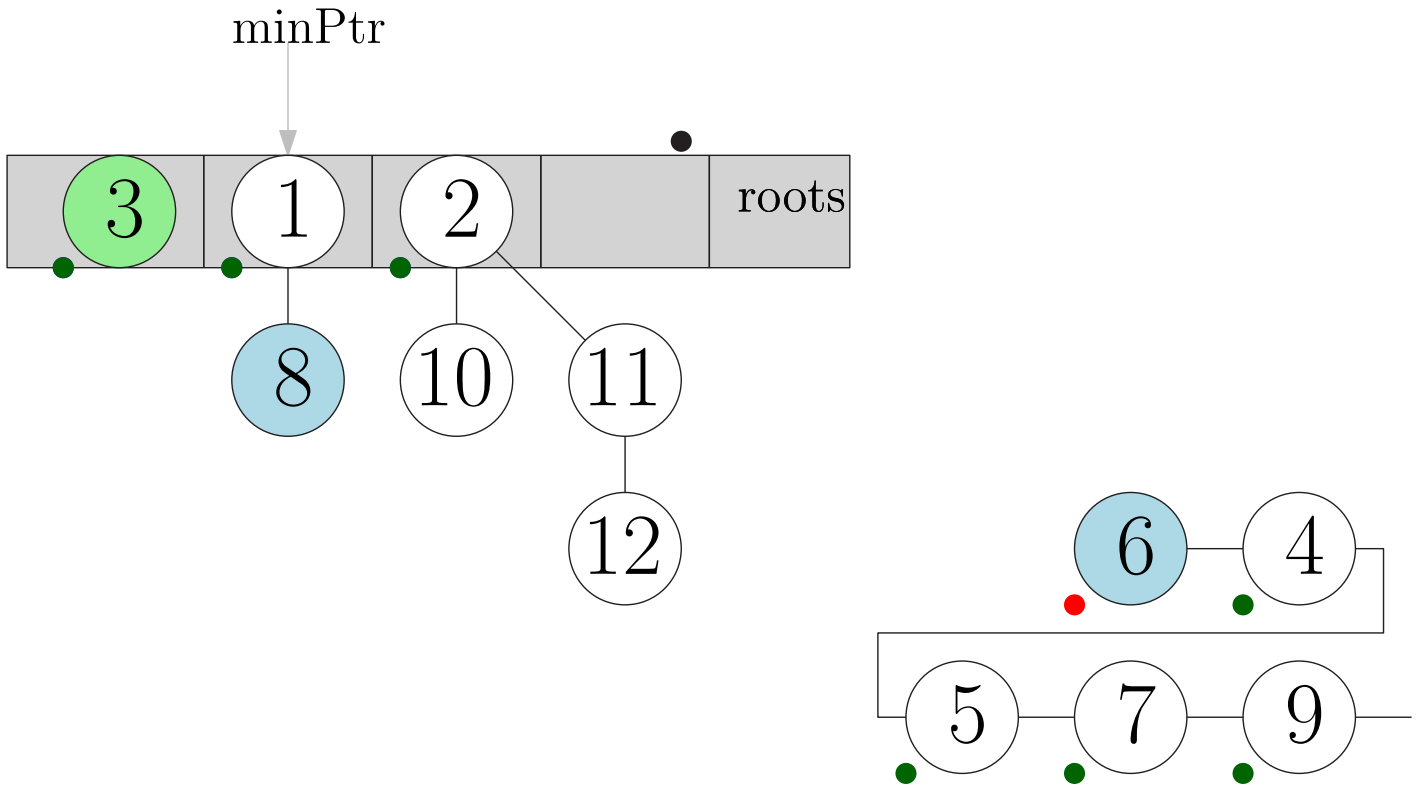
Fibonacci Heaps - Delete Min



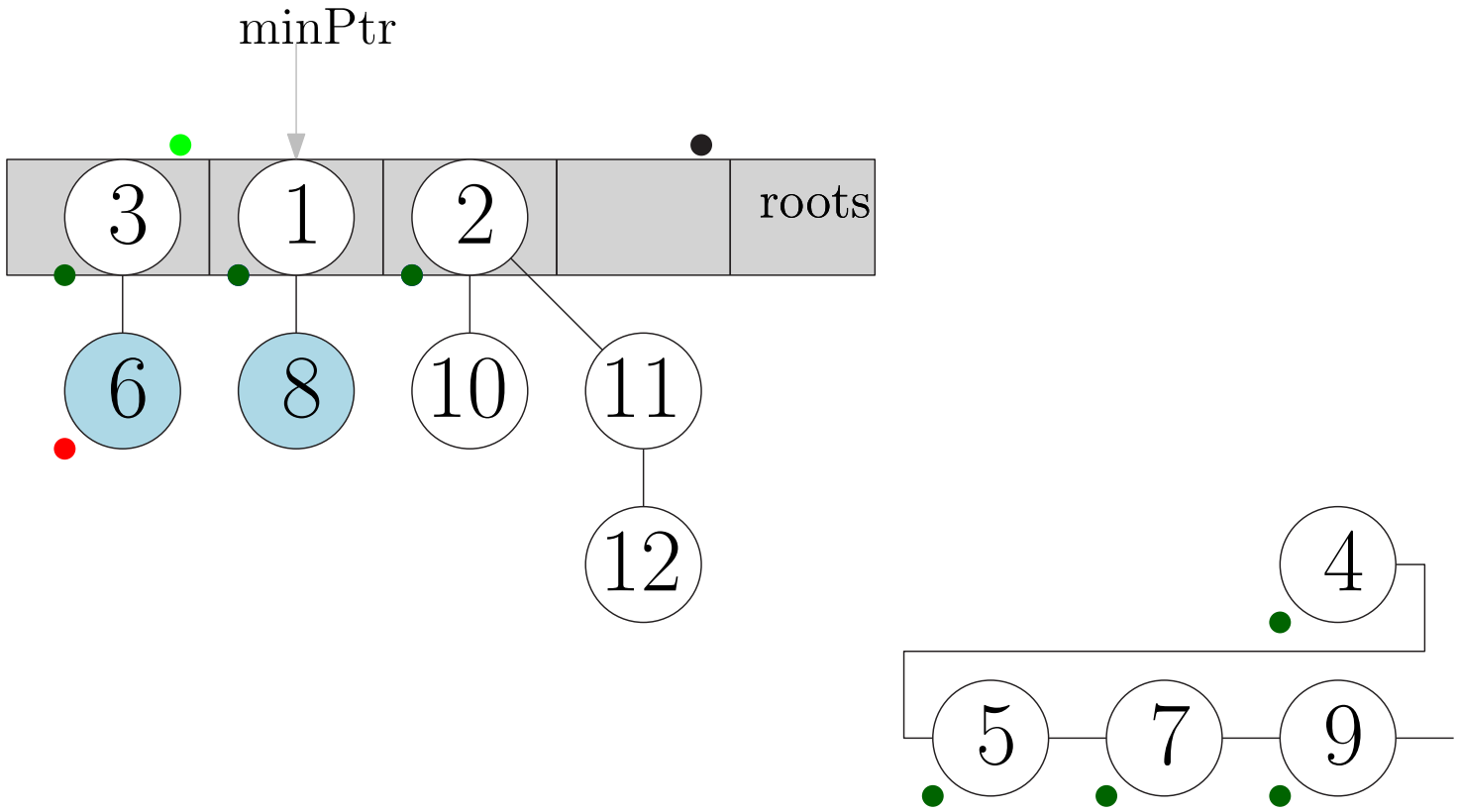
Fibonacci Heaps - Delete Min



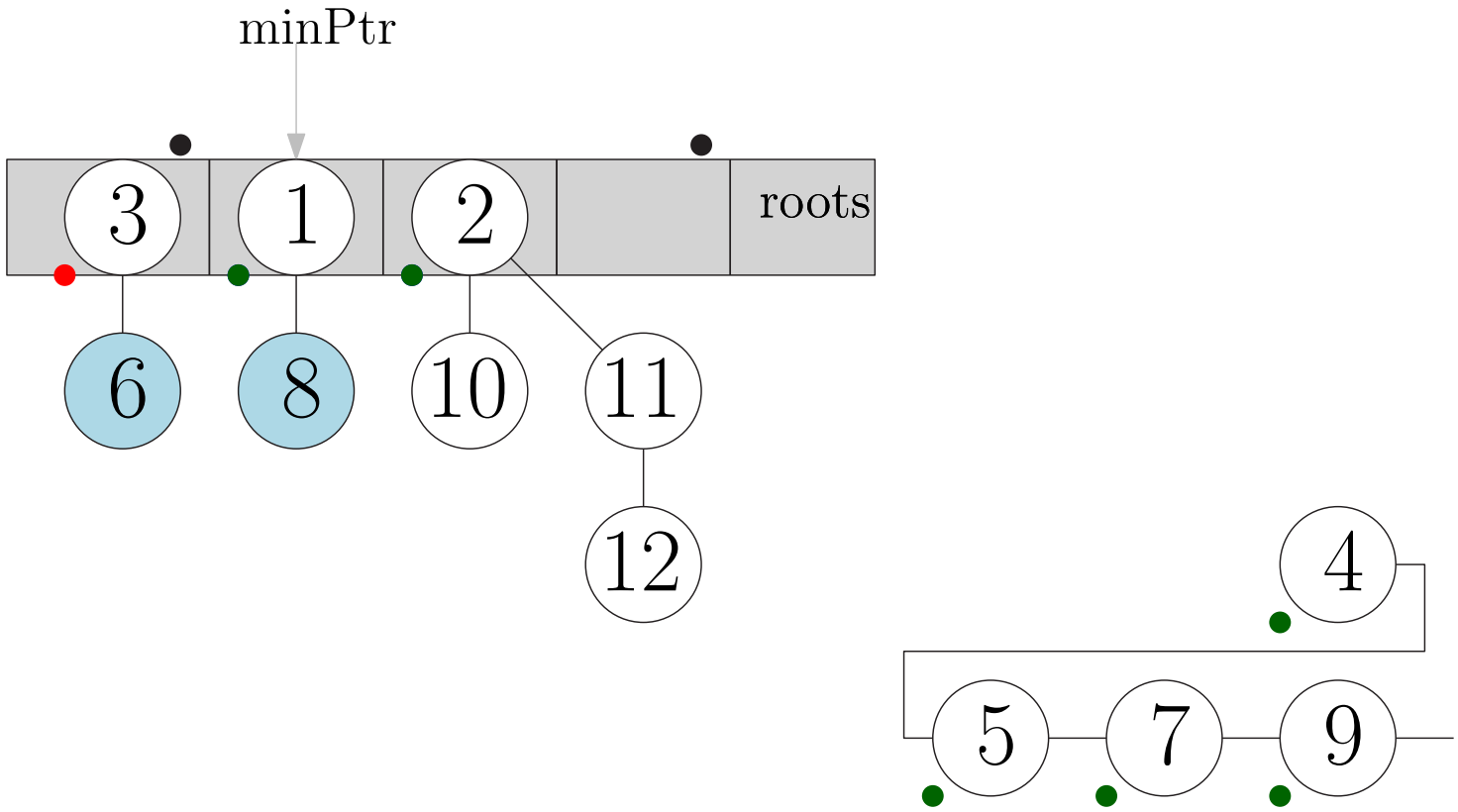
Fibonacci Heaps - Delete Min



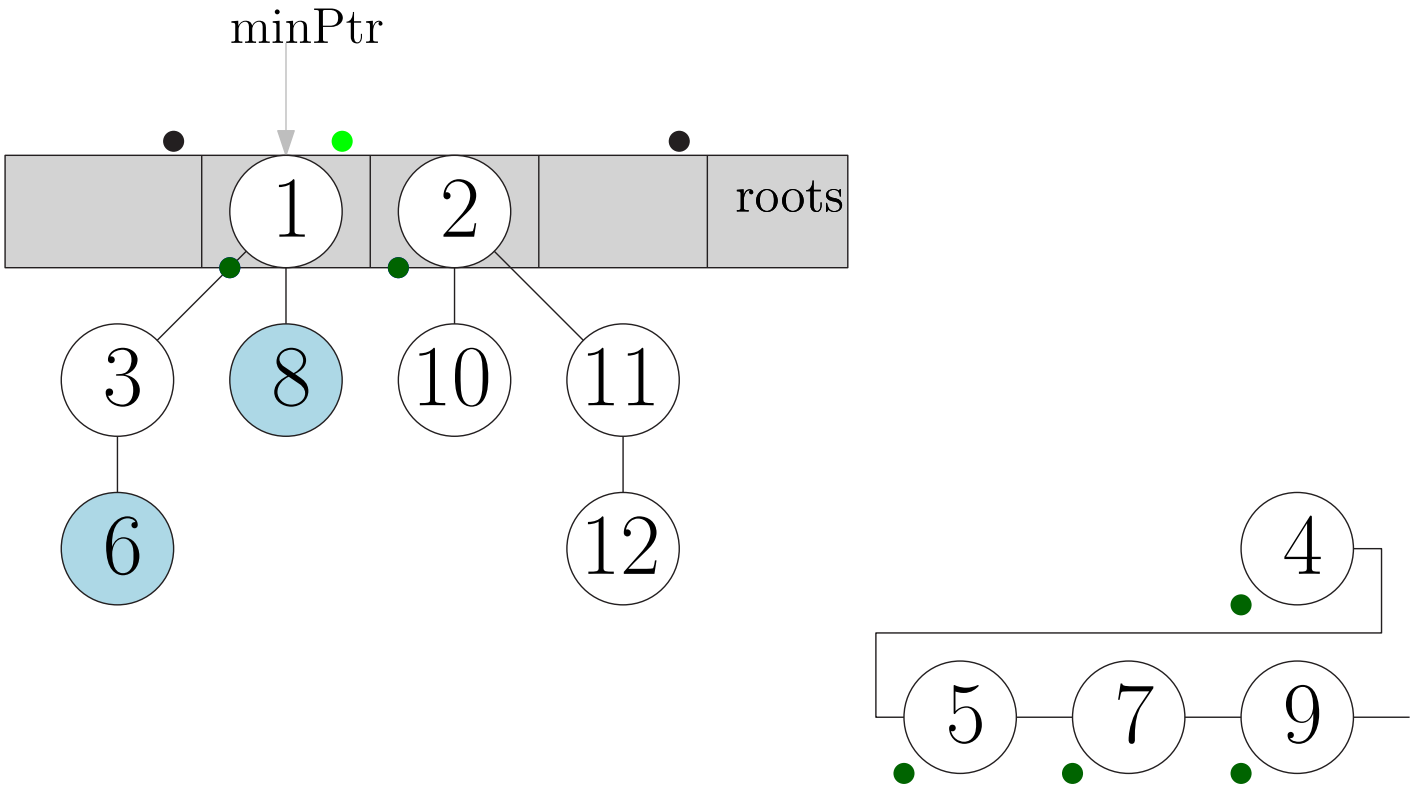
Fibonacci Heaps - Delete Min



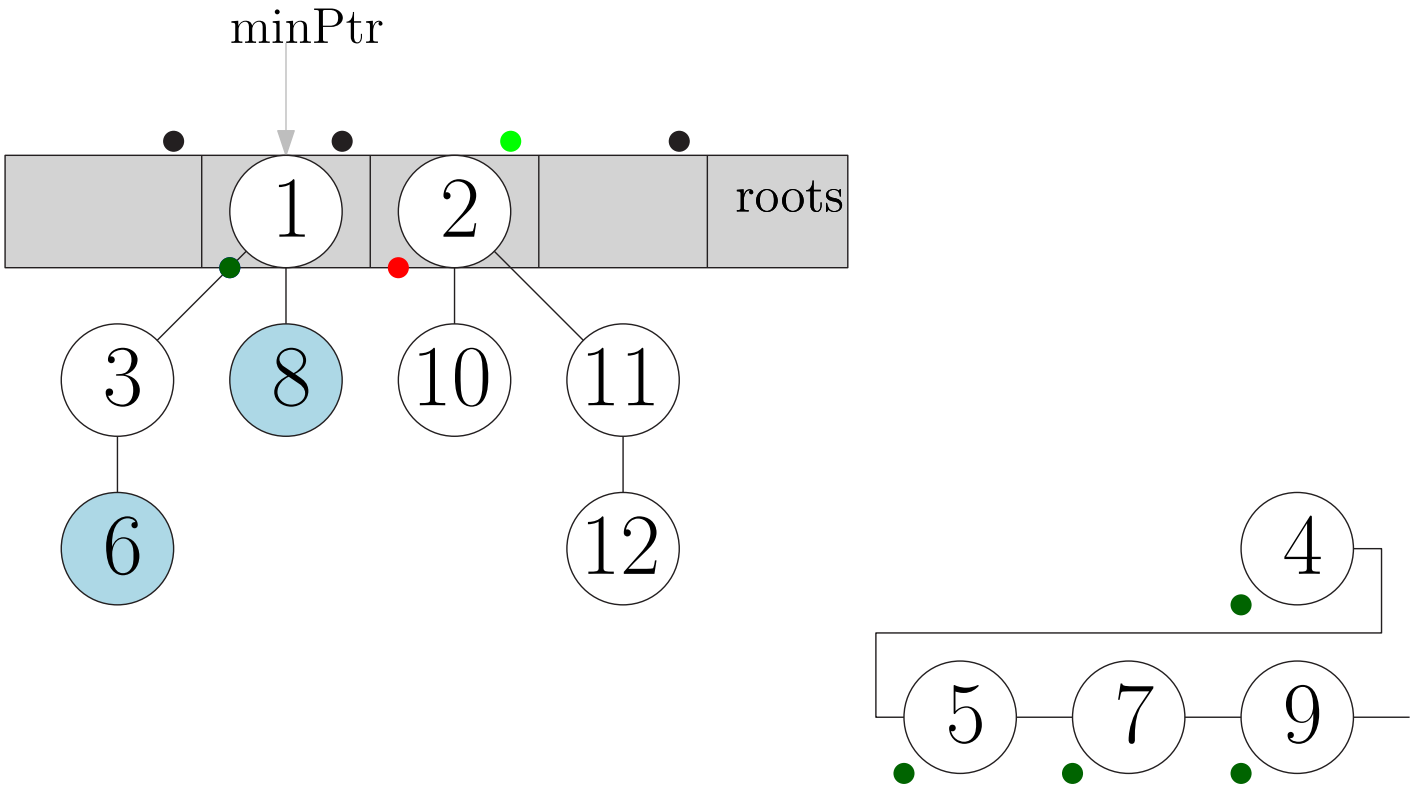
Fibonacci Heaps - Delete Min



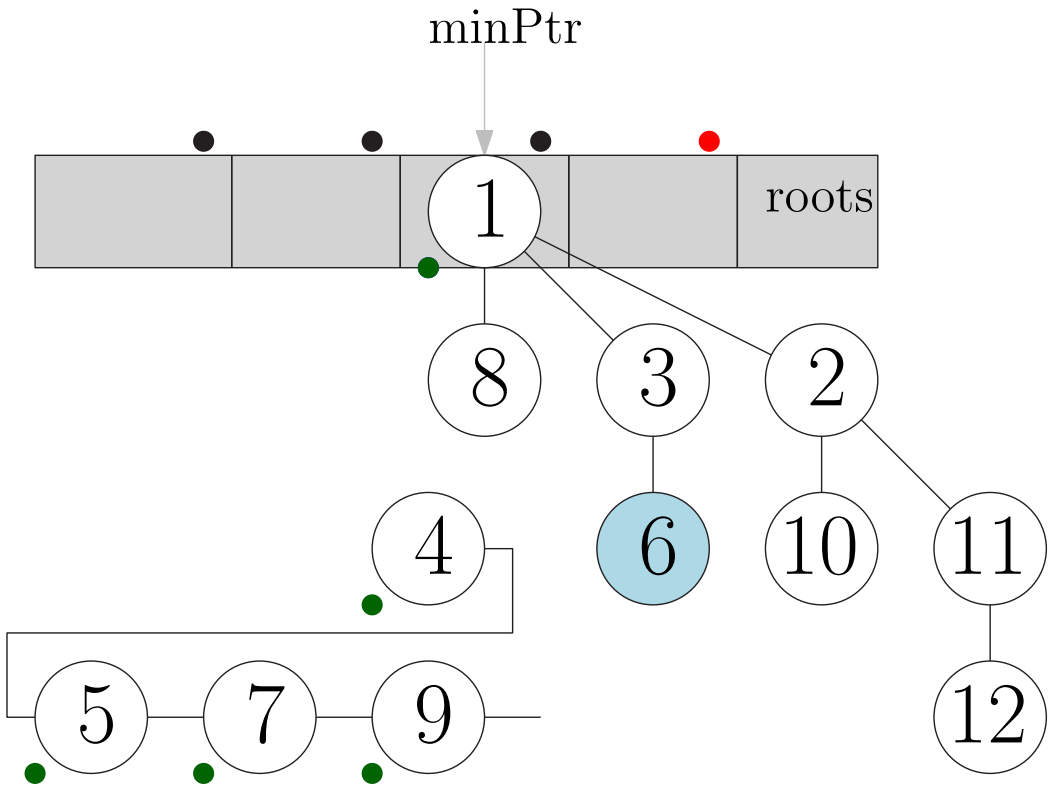
Fibonacci Heaps - Delete Min



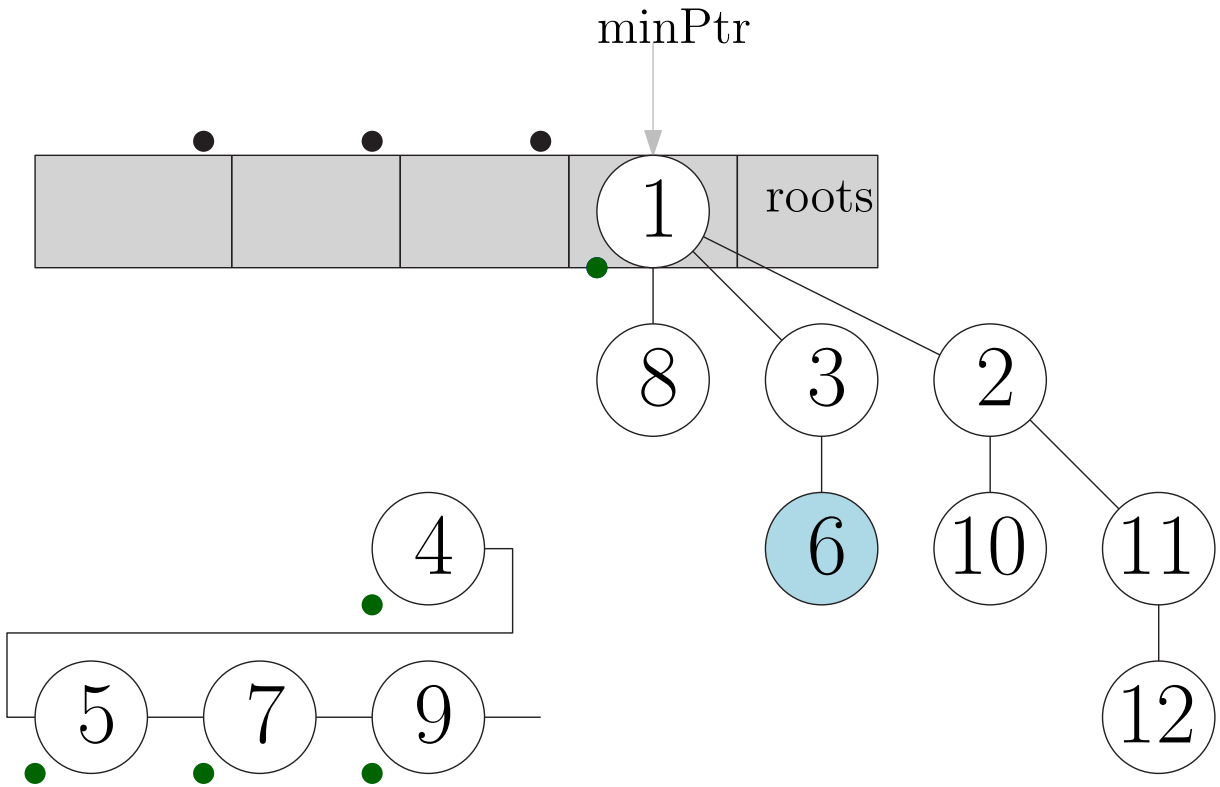
Fibonacci Heaps - Delete Min



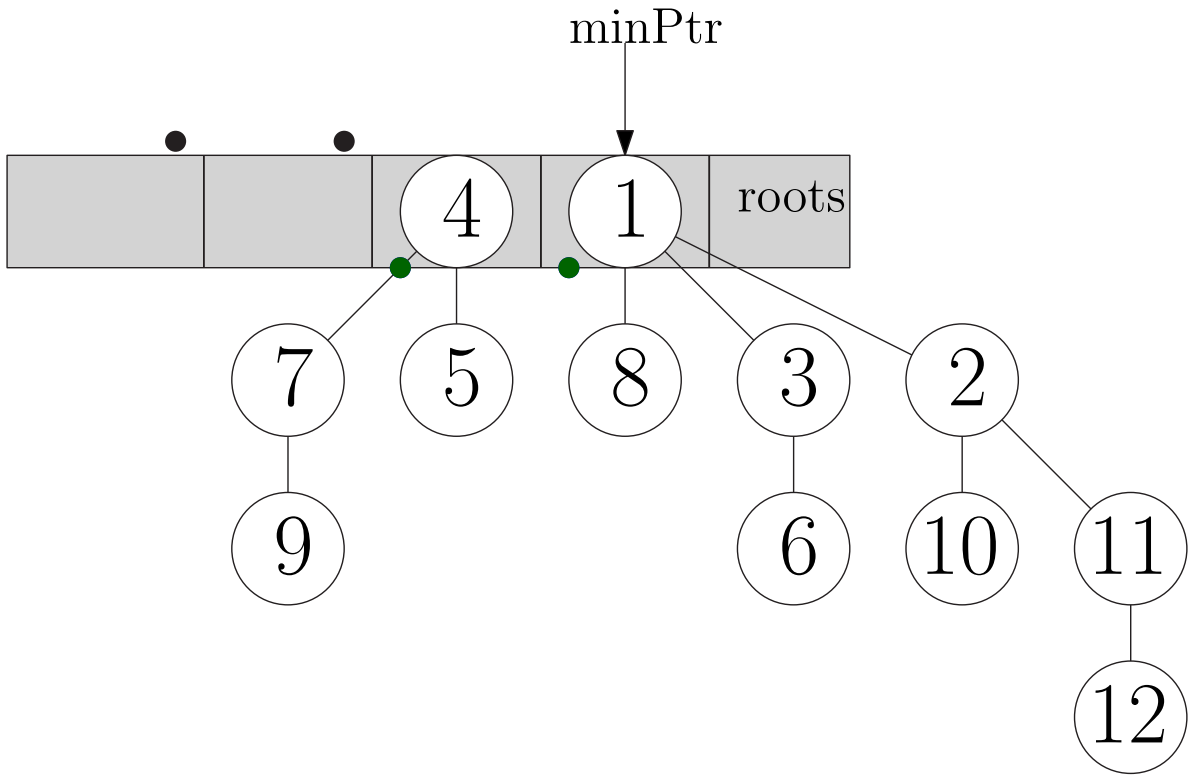
Fibonacci Heaps - Delete Min



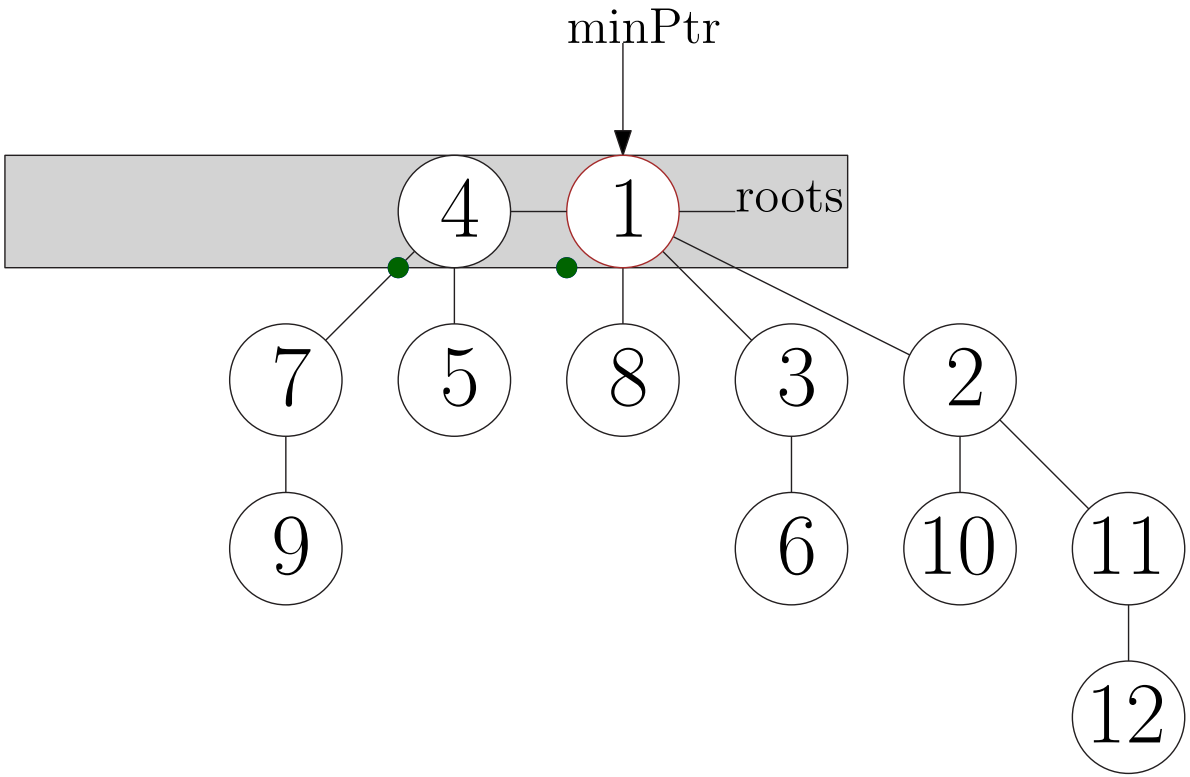
Fibonacci Heaps - Delete Min



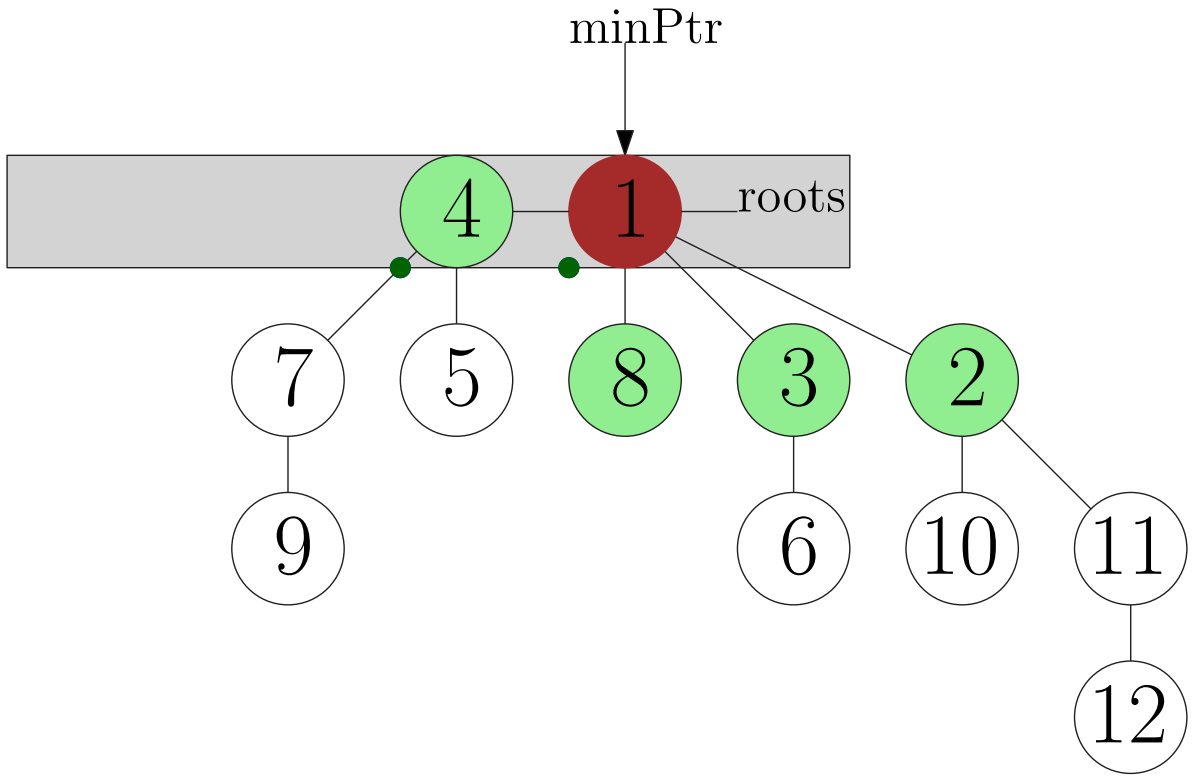
Fibonacci Heaps - Delete Min



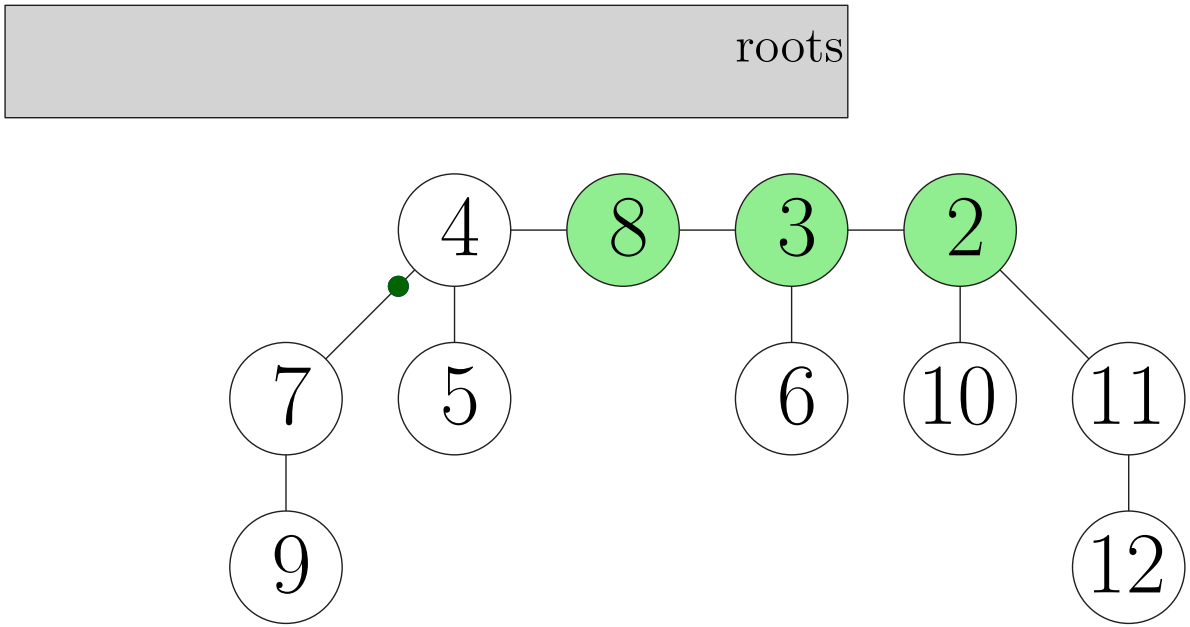
Fibonacci Heaps - Delete Min



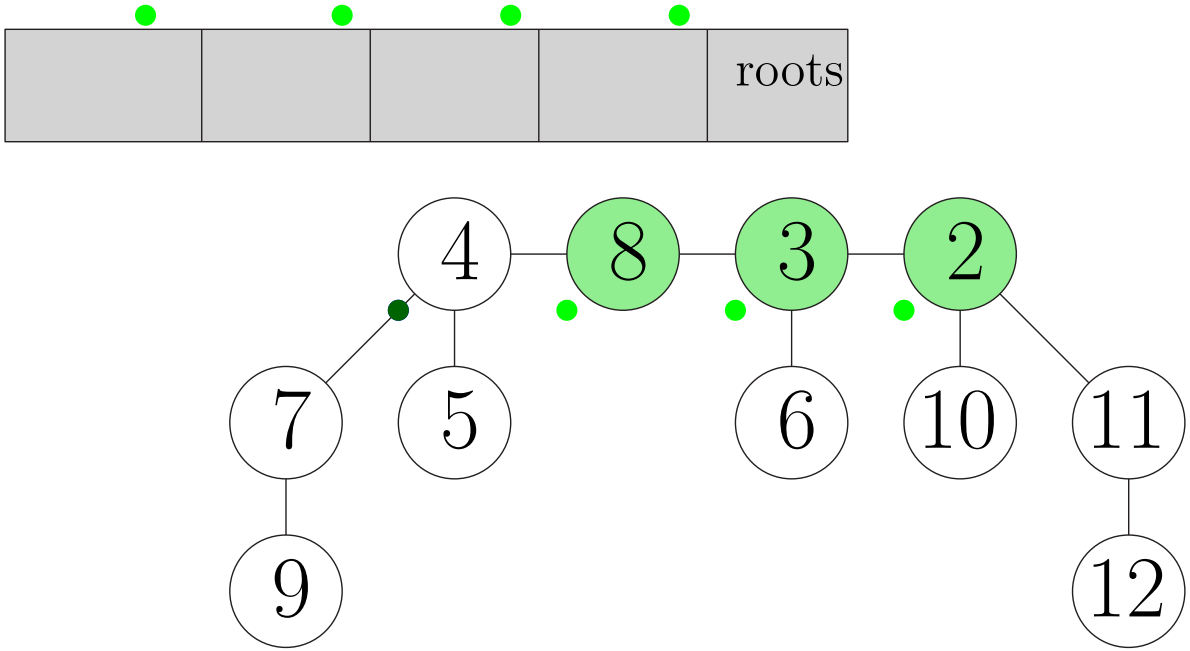
Fibonacci Heaps - Delete Min



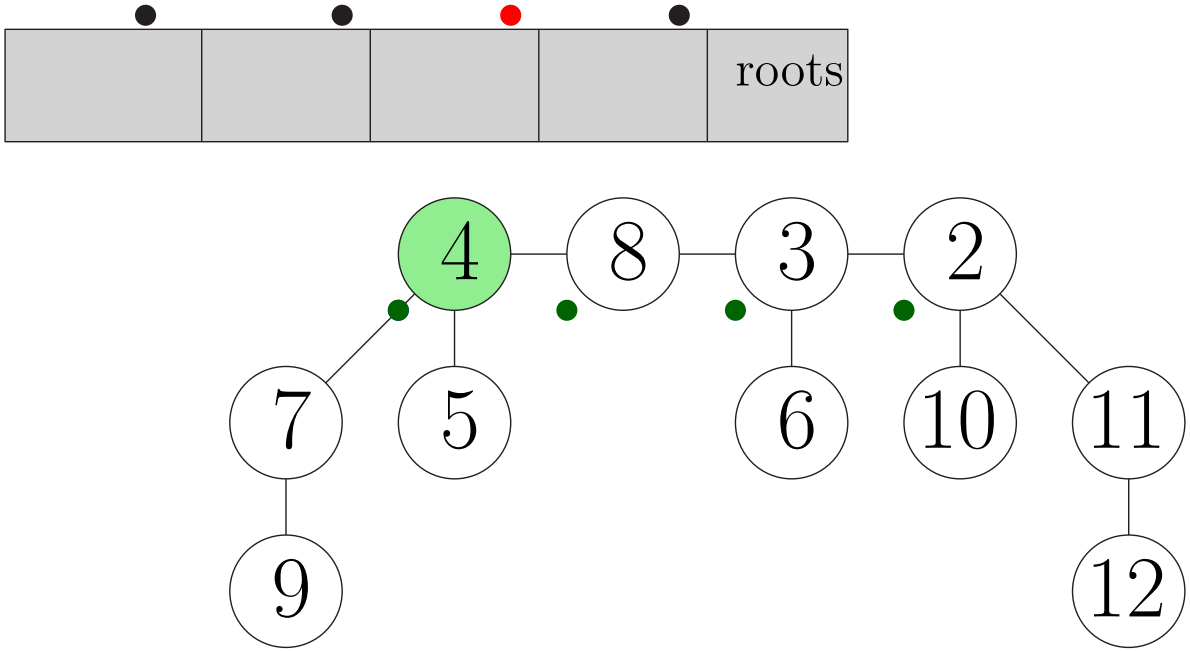
Fibonacci Heaps - Delete Min



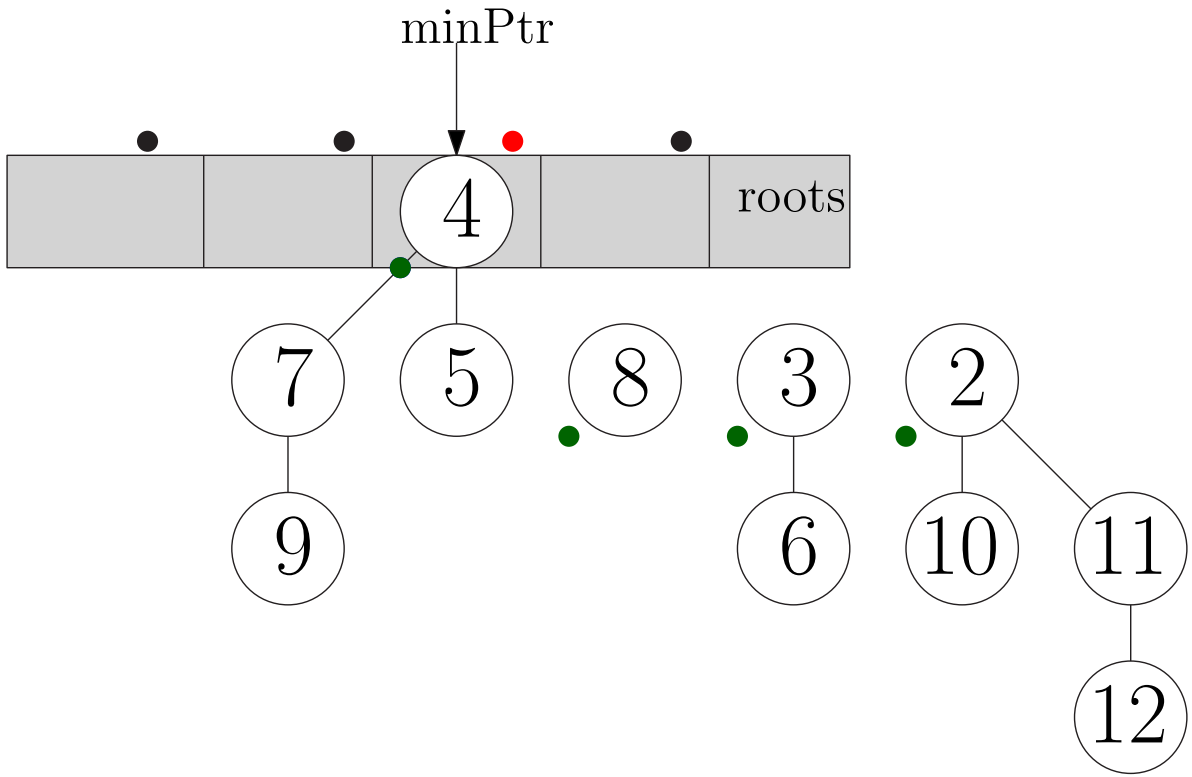
Fibonacci Heaps - Delete Min



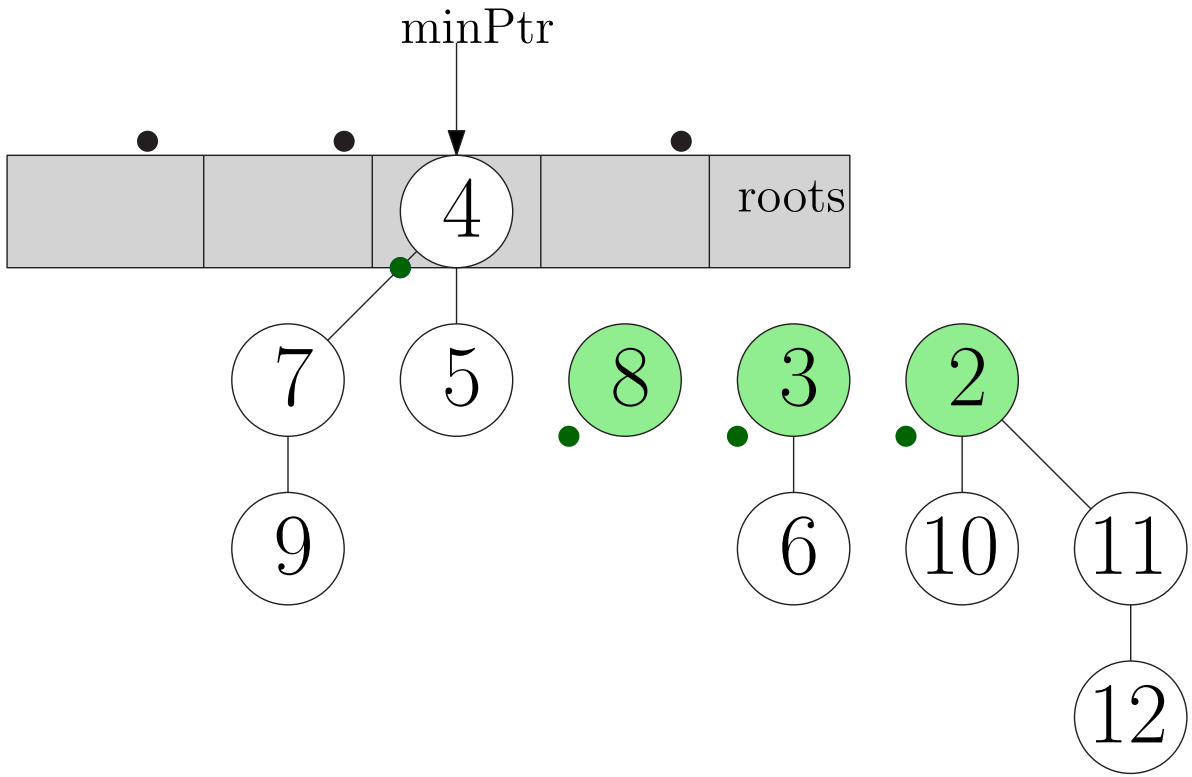
Fibonacci Heaps - Delete Min



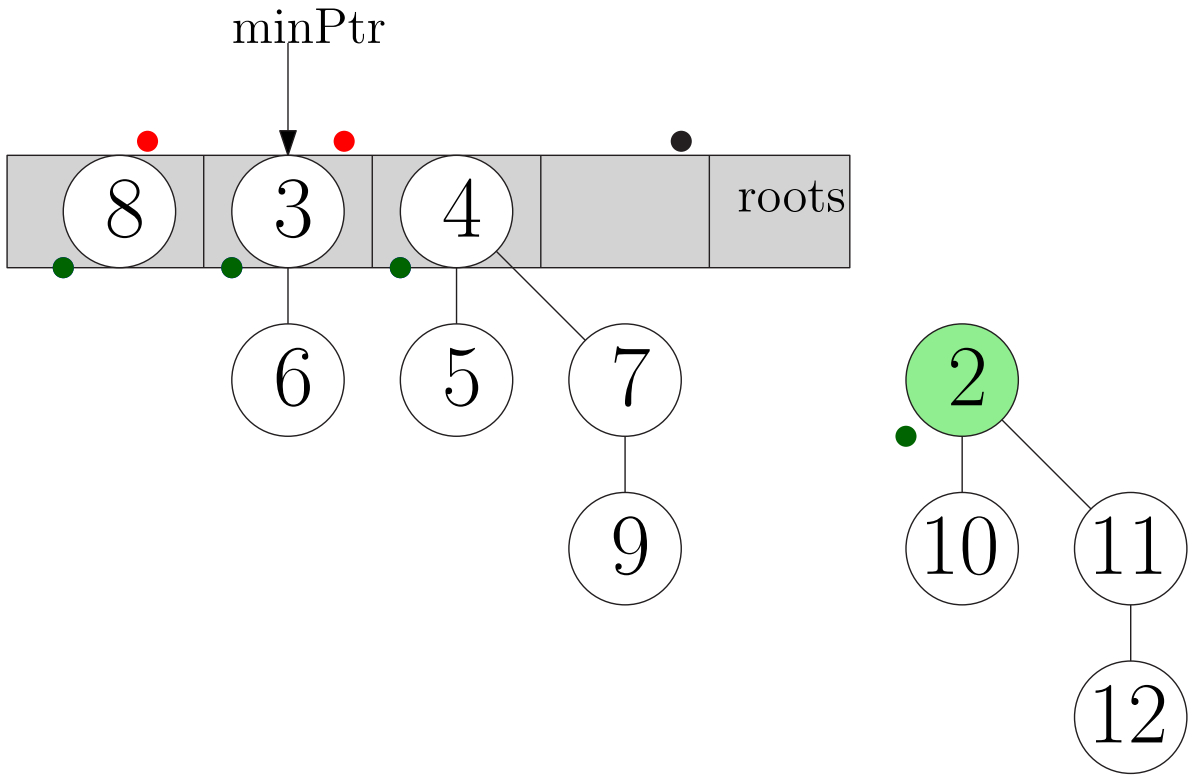
Fibonacci Heaps - Delete Min



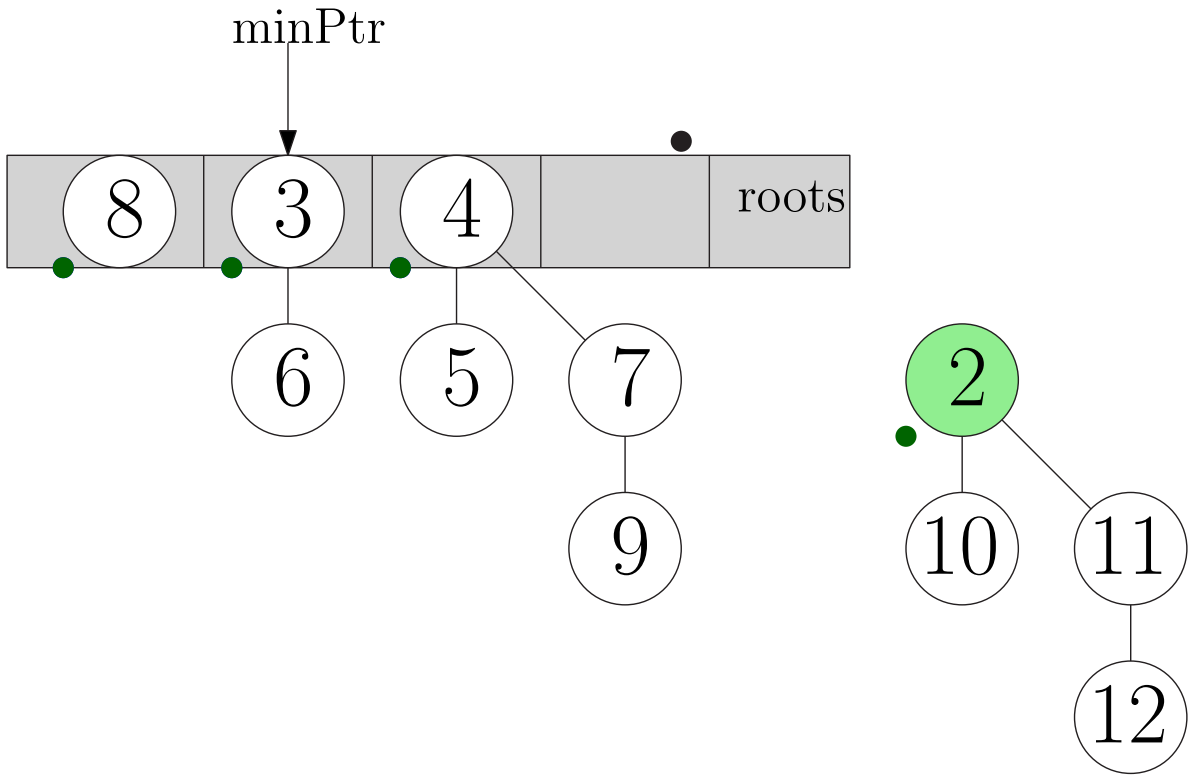
Fibonacci Heaps - Delete Min



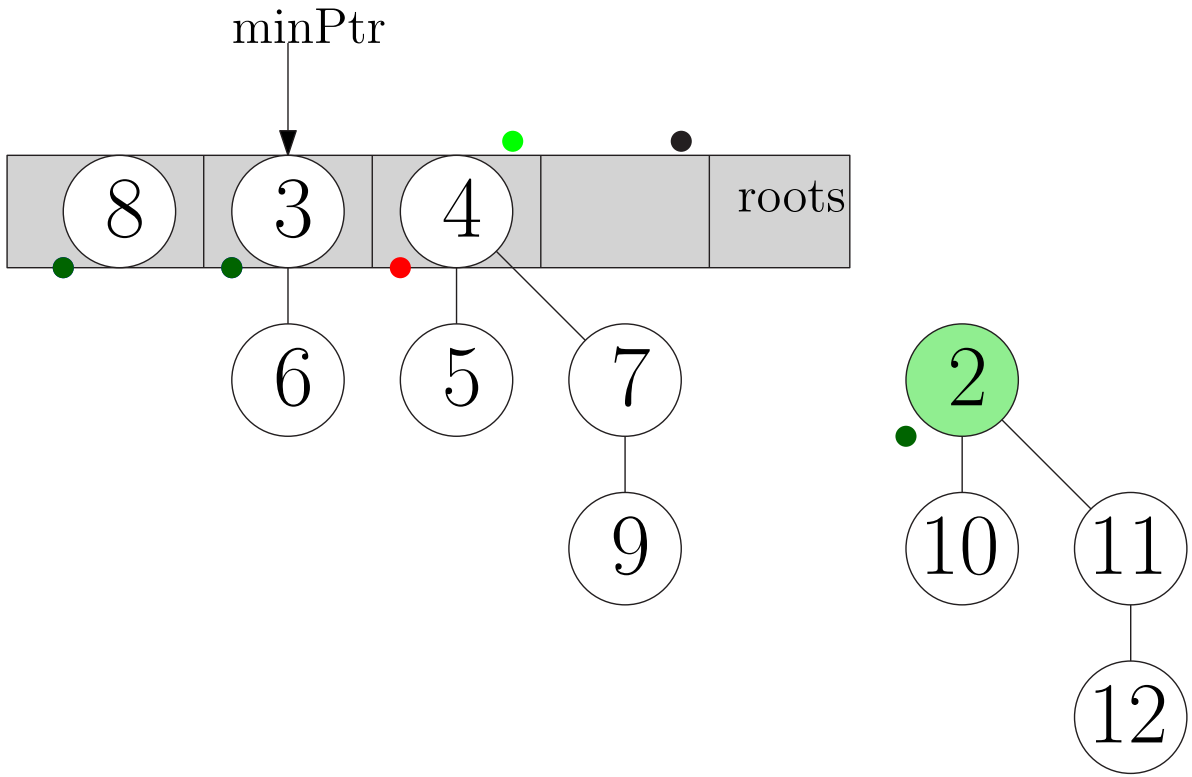
Fibonacci Heaps - Delete Min



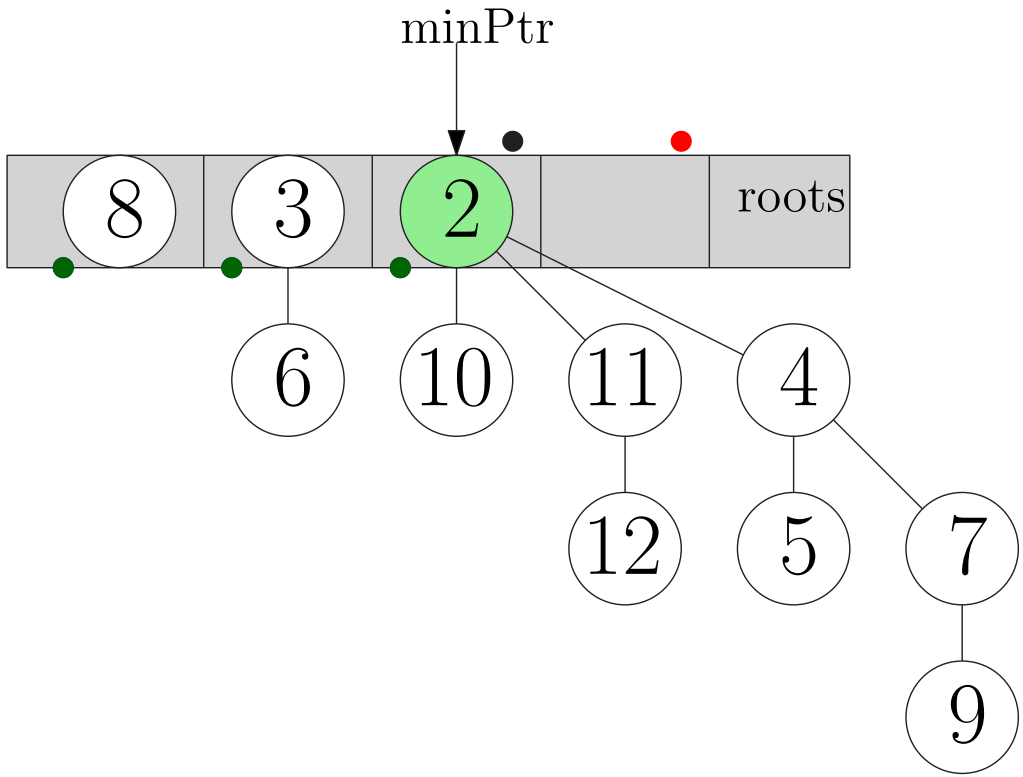
Fibonacci Heaps - Delete Min



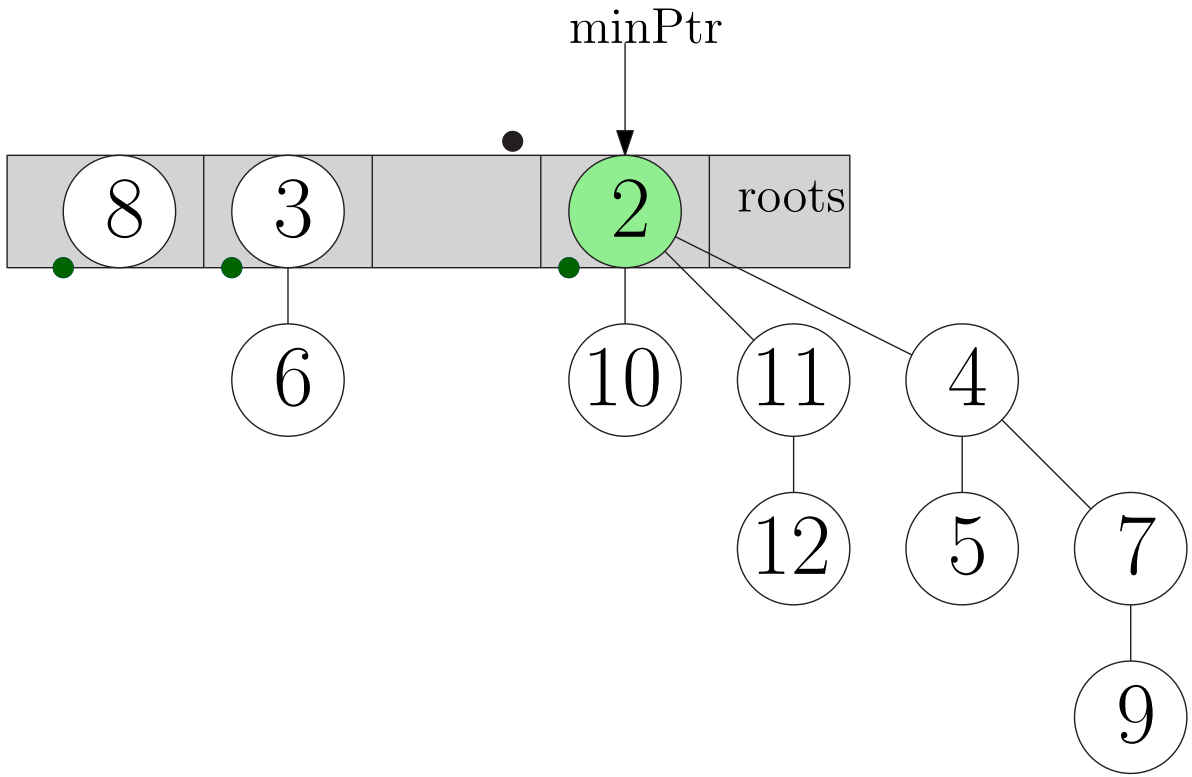
Fibonacci Heaps - Delete Min



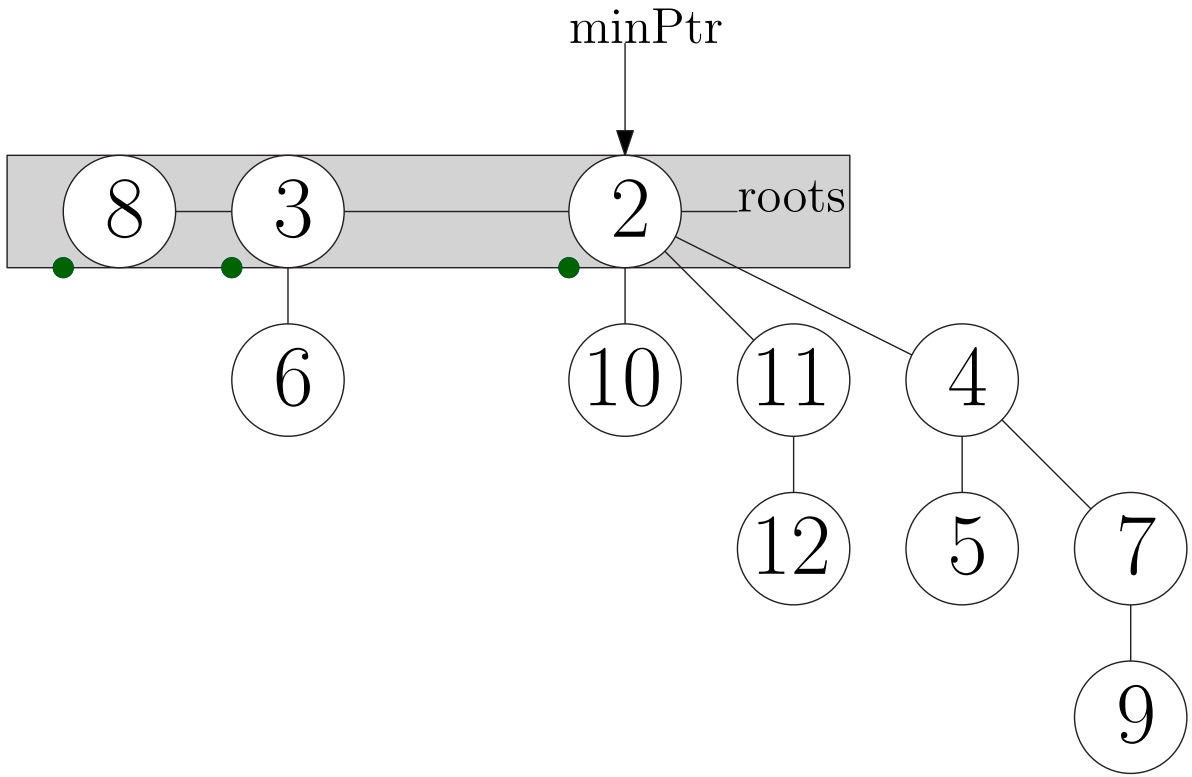
Fibonacci Heaps - Delete Min



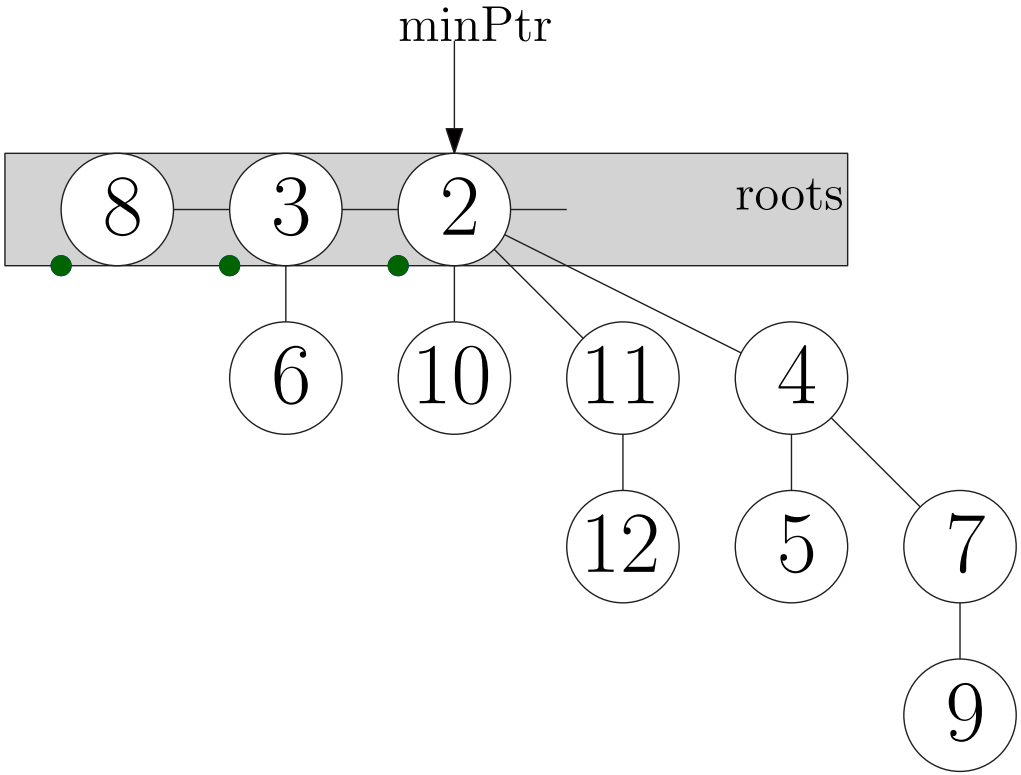
Fibonacci Heaps - Delete Min



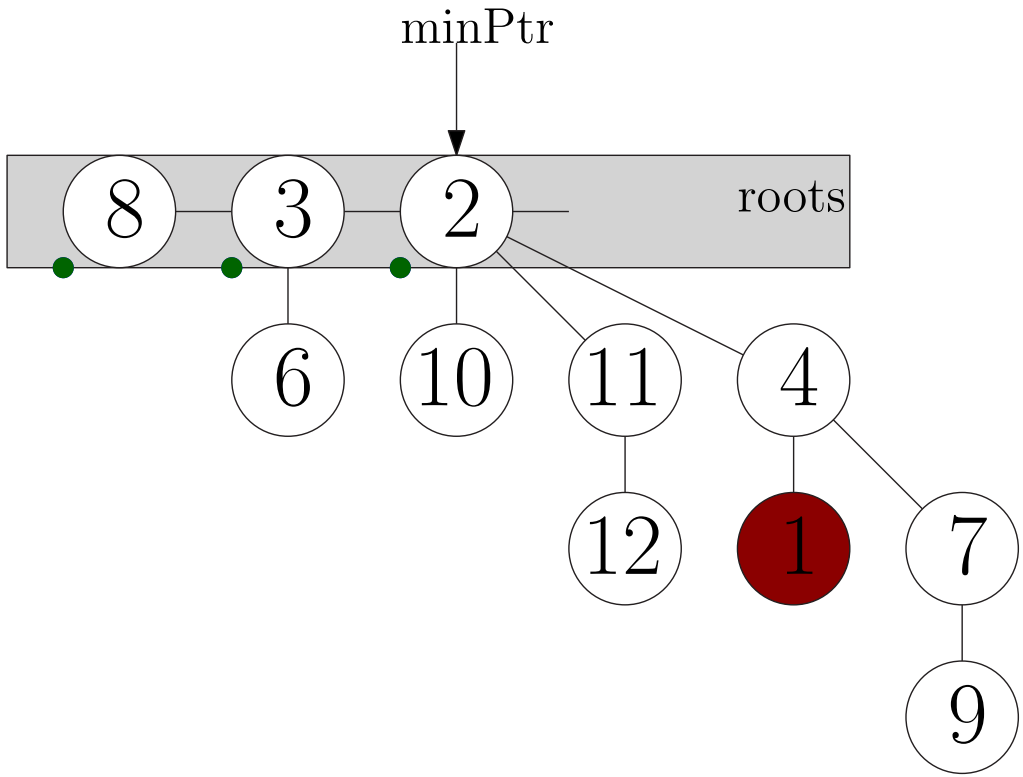
Fibonacci Heaps - Delete Min



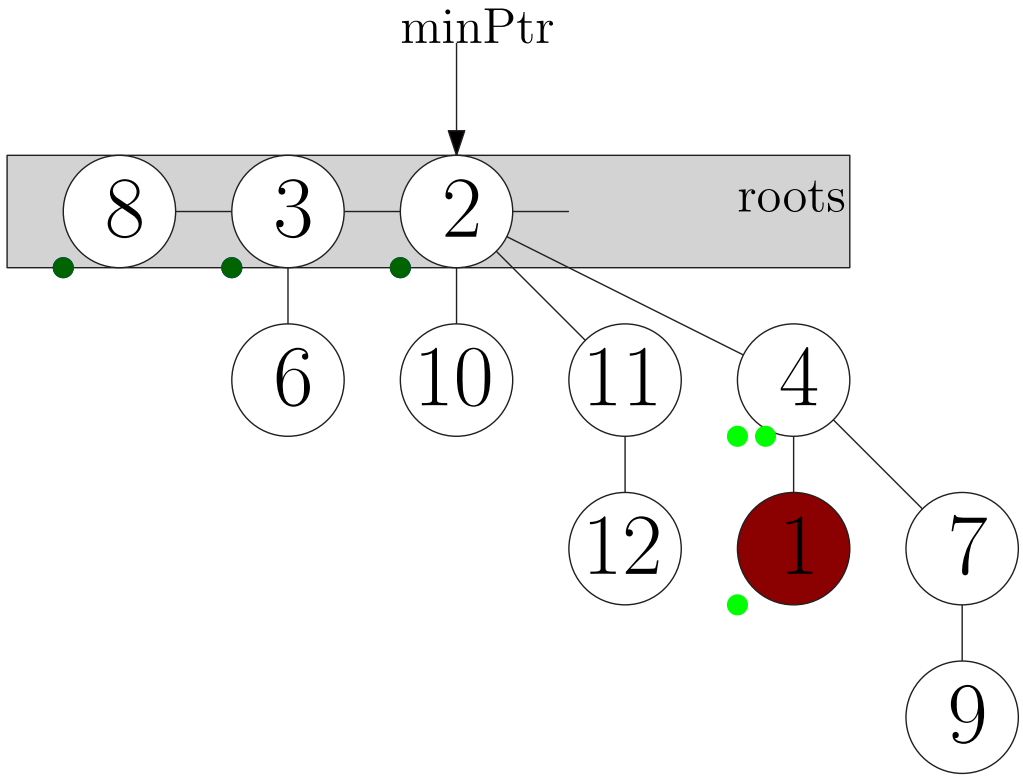
Fibonacci Heaps - Delete Min



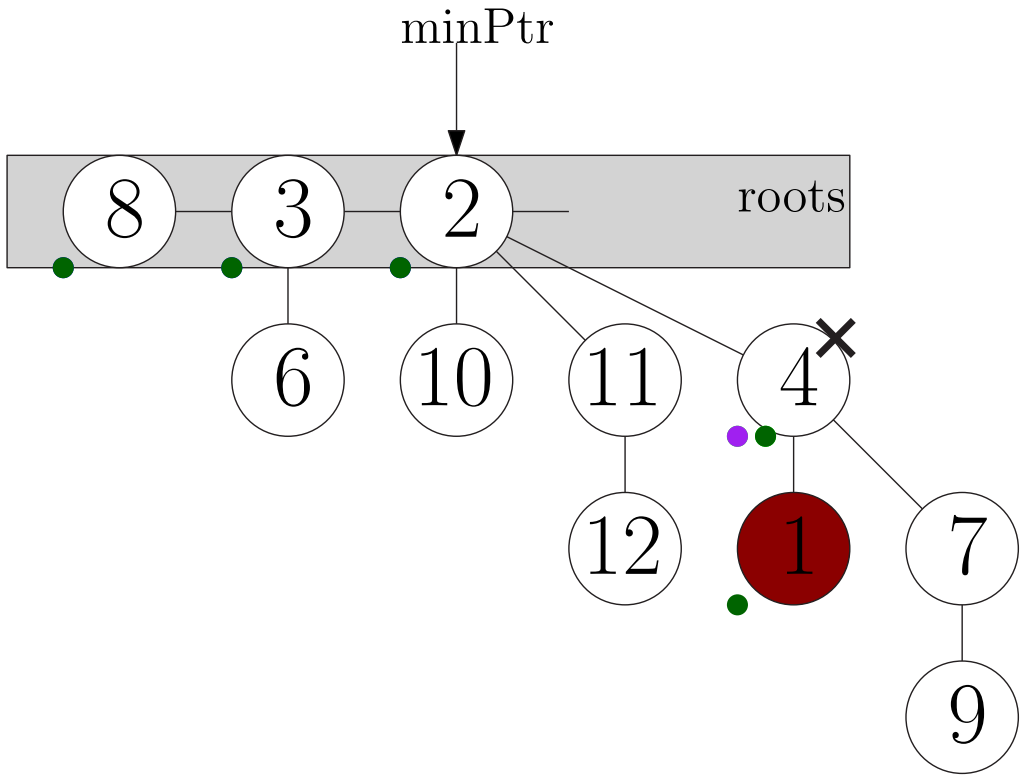
Fibonacci Heaps - Decrease Key



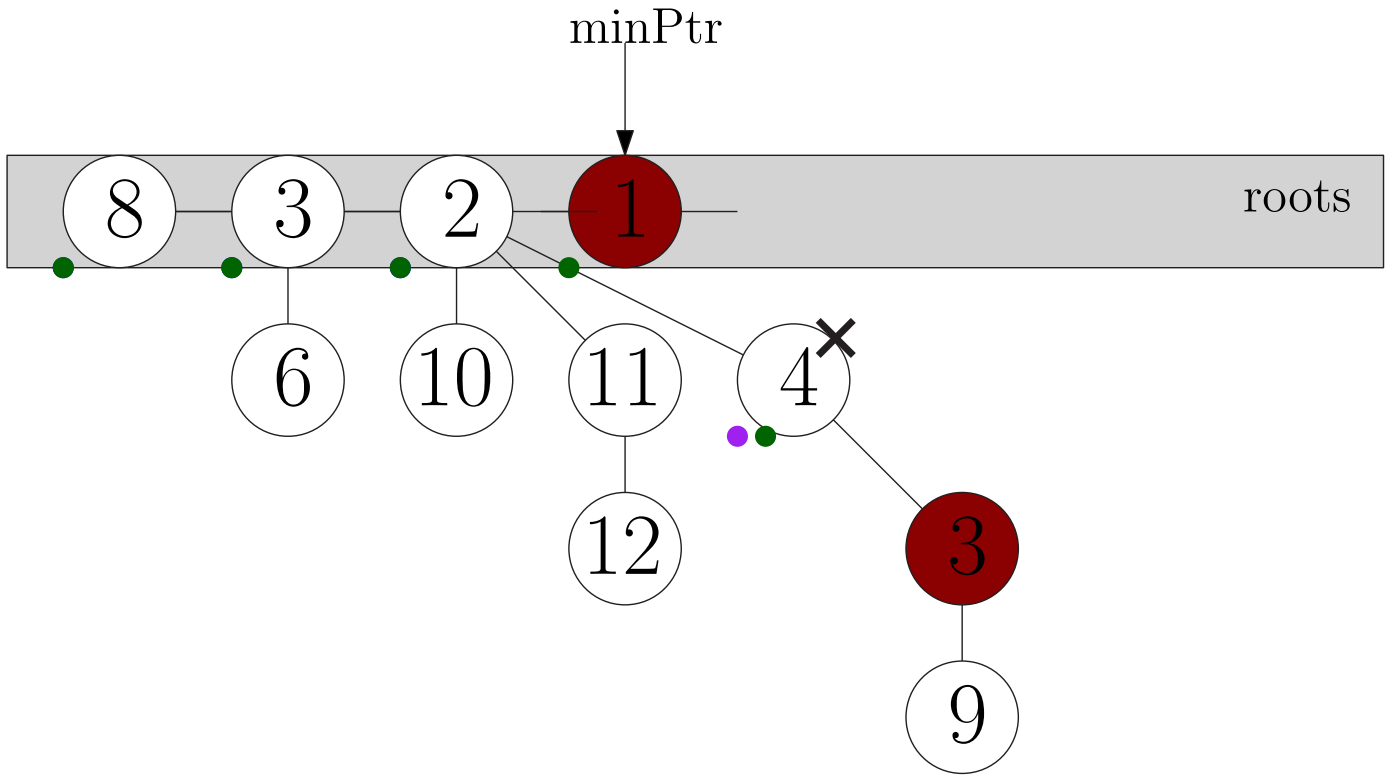
Fibonacci Heaps - Decrease Key



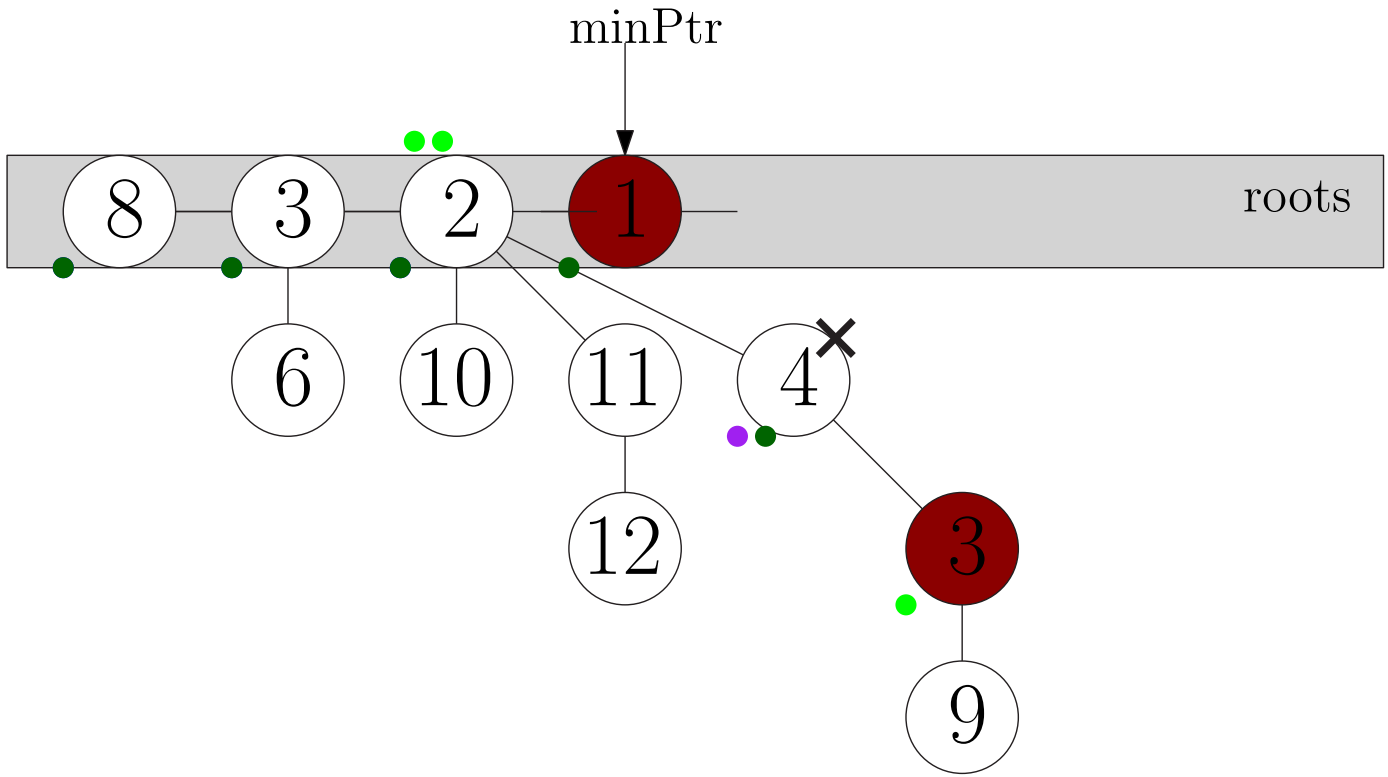
Fibonacci Heaps - Decrease Key



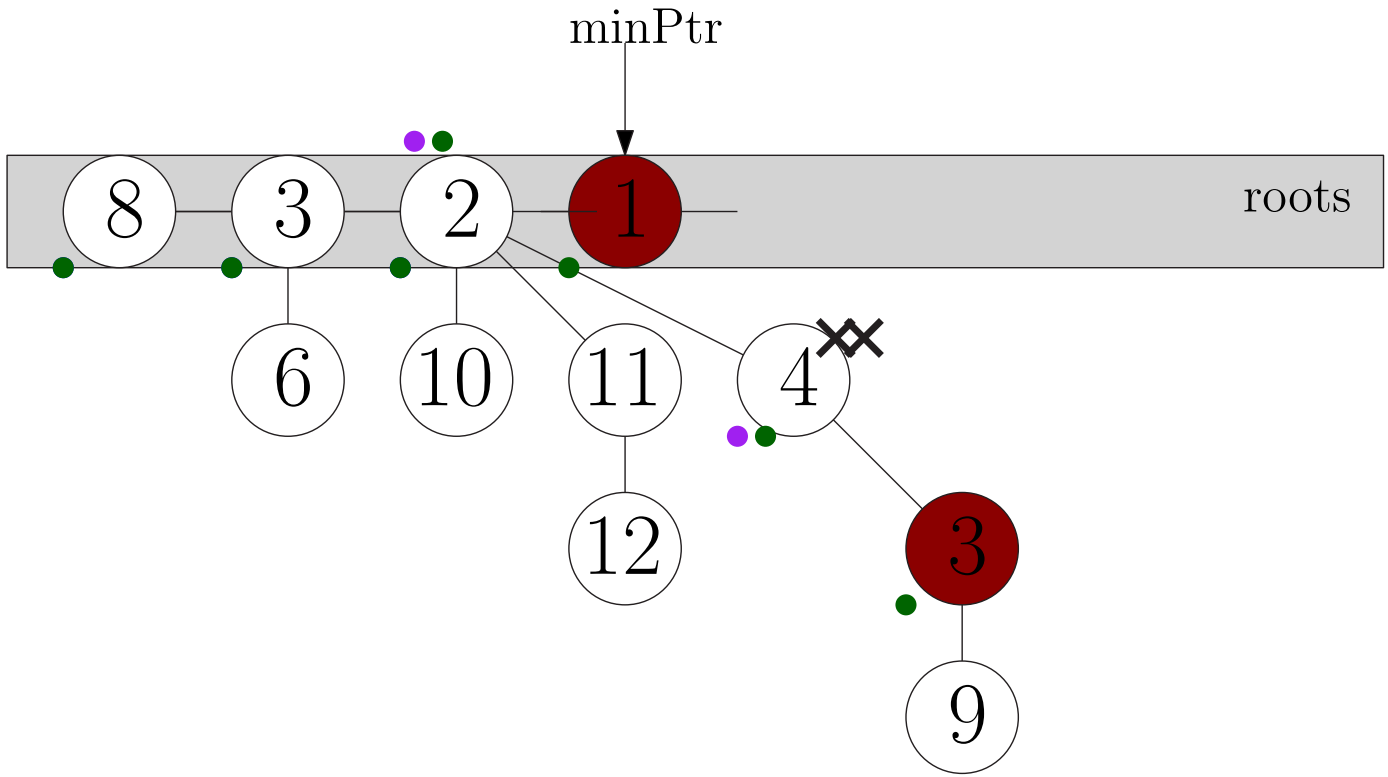
Fibonacci Heaps - Decrease Key



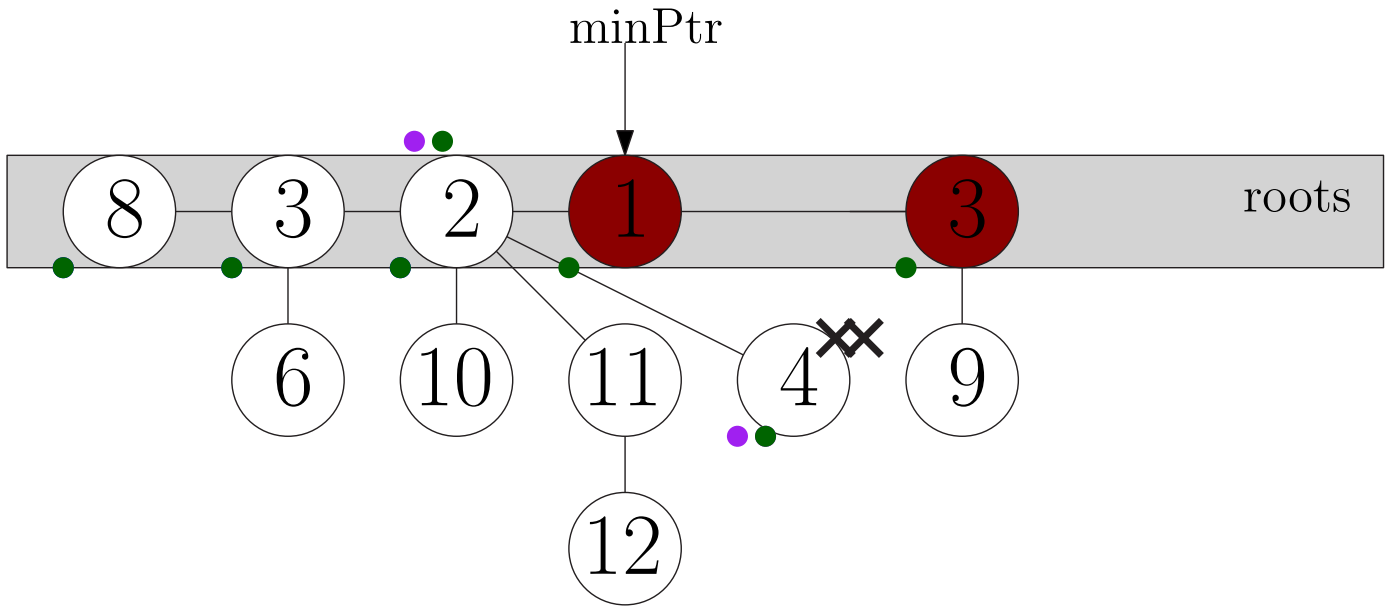
Fibonacci Heaps - Decrease Key



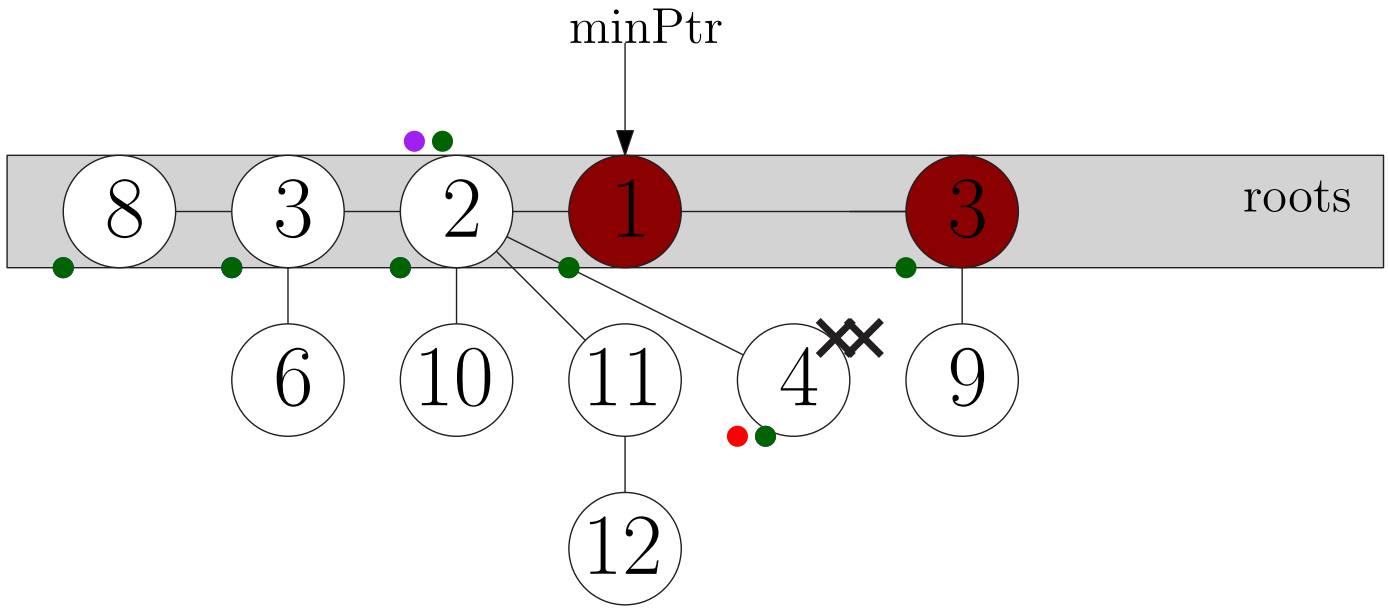
Fibonacci Heaps - Decrease Key



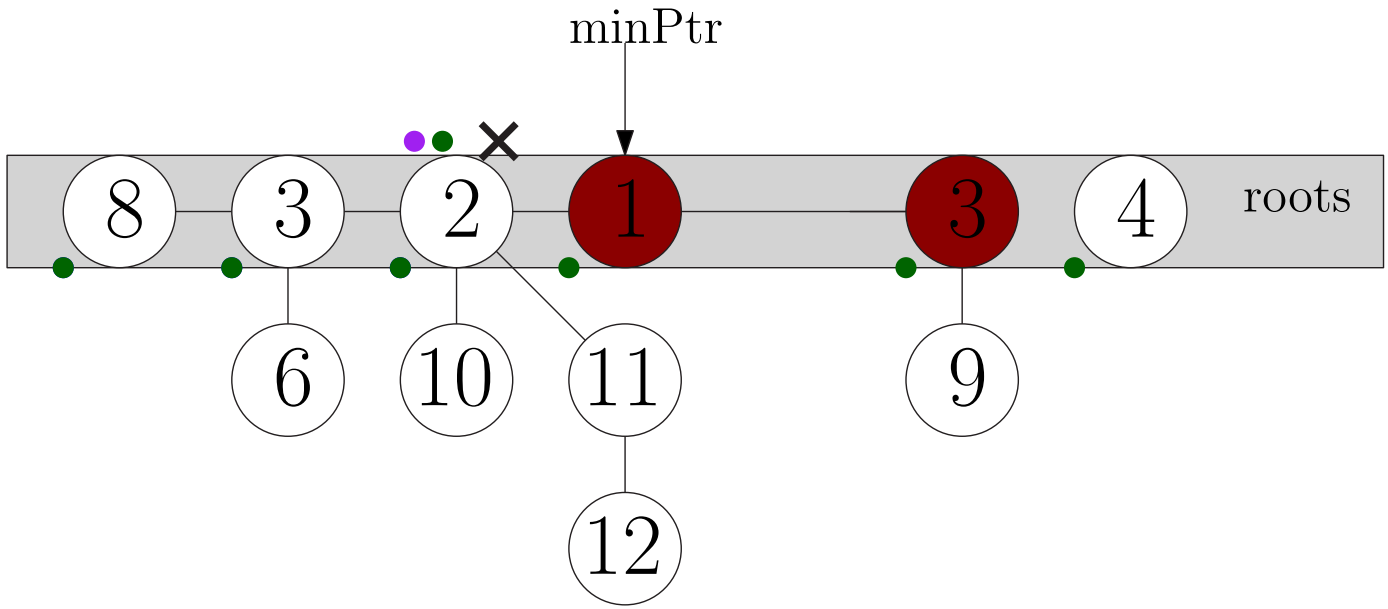
Fibonacci Heaps - Decrease Key



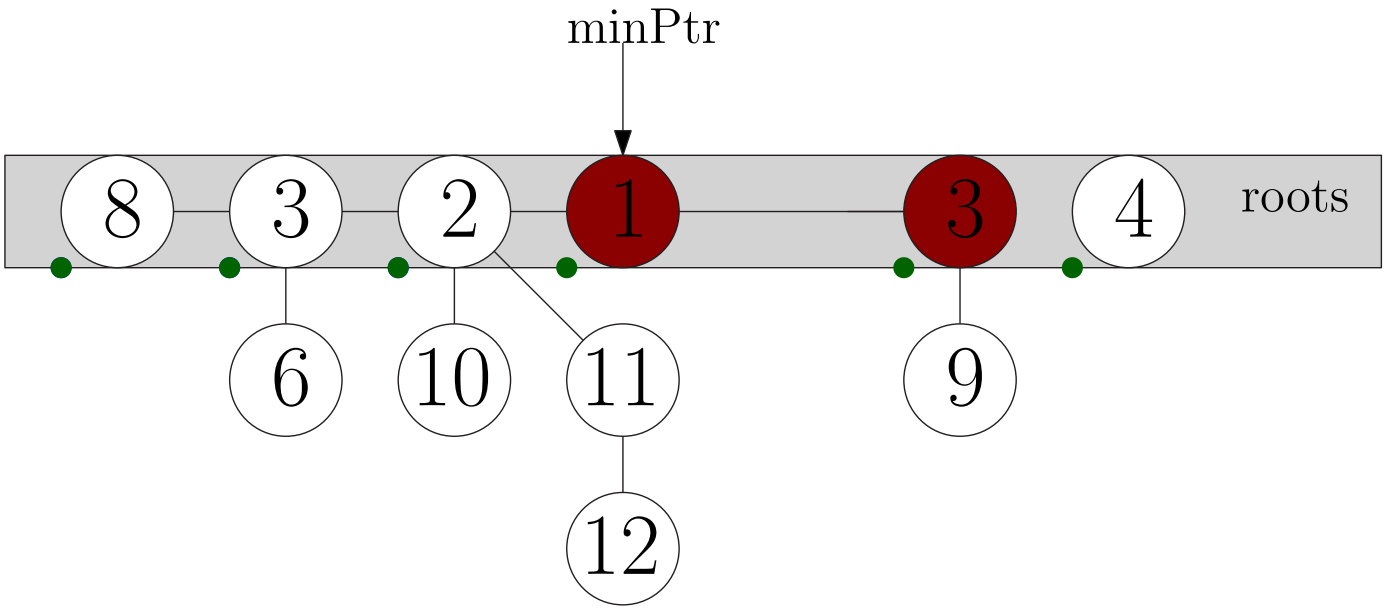
Fibonacci Heaps - Decrease Key



Fibonacci Heaps - Decrease Key



Fibonacci Heaps - Decrease Key



Amortisierte Analyse

- Operationen $Op = \langle Op_1, \ldots, Op_n \rangle$ angewandt auf Datenstruktur D
- Op_i überführt D von Zustand s_{i-1} in s_i
- Kosten/Laufzeit $T_{Op_i}(s_{i-1})$ (T_{Op_i} hängt von Zustand s_{i-1} ab)

$$T(Op) = \sum_{i=1}^{n} T_{Op_i}(s_{i-1})$$

- **Gesucht:** Amortisierte Laufzeit $A_{Op_i}(s_{i-1})$ für jede Operation, sodass

$$T(Op) \leq A(Op) := c + \sum_{i=1}^{n} A_{Op_i}(s_{i-1}) \text{ für beliebige Konstance } c$$

Amortisierte Analyse

- Operationen $Op = \langle Op_1, \ldots, Op_n \rangle$ angewandt auf Datenstruktur D
- Op_i überführt D von Zustand s_{i-1} in s_i
- Kosten/Laufzeit $T_{Op_i}(s_{i-1})$ (T_{Op_i} hängt von Zustand s_{i-1} ab)

$$T(Op) = \sum_{i=1}^{n} T_{Op_i}(s_{i-1})$$

- **Gesucht:** Amortisierte Laufzeit $A_{Op_i}(s_{i-1})$ für jede Operation, sodass

$$T(Op) \leq A(Op) := c + \sum_{i=1}^{n} A_{Op_i}(s_{i-1}) \text{ für beliebige Konstance } c$$

Potential Methode

- Potentialfunktion $\Phi : S \to \mathbb{R}_{>}$ (S = Zustandsraum)
- $\Delta\Phi(s_{i-1}) := \Phi(s_i) - \Phi(s_{i-1})$

$$A_{Op_i}(s_{i-1}) := T_{Op_i}(s_{i-1}) + \Delta\Phi(s_{i-1})$$
$$\Rightarrow T(OP) \leq A(OP) + \Phi(s_0) \text{ (ohne Beweis)}$$

- **Anmerkung**: Φ kann beliebig gewählt werden (z.B. $\Phi(s) = 42$ für alle $s \in S$), jedoch führen nicht alle Φ zu sinnvollen amortisierten Schranken.

Potential Methode

- Potentialfunktion $\Phi : S \to \mathbb{R}_{>}$ (S = Zustandsraum)
- $\Delta\Phi(s_{i-1}) := \Phi(s_i) - \Phi(s_{i-1})$

$$A_{Op_i}(s_{i-1}) := T_{Op_i}(s_{i-1}) + \Delta\Phi(s_{i-1})$$

$$\Rightarrow T(OP) \leq A(OP) + \Phi(s_0) \text{ (ohne Beweis)}$$

- **Anmerkung**: Φ kann beliebig gewählt werden (z.B. $\Phi(s) = 42$ für alle $s \in S$), jedoch führen nicht alle Φ zu sinnvollen amortisierten Schranken.

Potential Methode

- Potentialfunktion $\Phi : S \to \mathbb{R}_>$ (S = Zustandsraum)
- $\Delta\Phi(s_{i-1}) := \Phi(s_i) - \Phi(s_{i-1})$

$$A_{Op_i}(s_{i-1}) := T_{Op_i}(s_{i-1}) + \Delta\Phi(s_{i-1})$$
$$\Rightarrow T(OP) \leq A(OP) + \Phi(s_0) \text{ (ohne Beweis)}$$

- **Anmerkung**: Φ kann beliebig gewählt werden (z.B. $\Phi(s) = 42$ für alle $s \in S$), jedoch führen nicht alle Φ zu sinnvollen amortisierten Schranken.

Potential Methode

- Potentialfunktion $\Phi : S \to \mathbb{R}_>$ (S = Zustandsraum)
- $\Delta\Phi(s_{i-1}) := \Phi(s_i) - \Phi(s_{i-1})$

$$A_{Op_i}(s_{i-1}) := T_{Op_i}(s_{i-1}) + \Delta\Phi(s_{i-1})$$
$$\Rightarrow T(OP) \leq A(OP) + \Phi(s_0) \text{ (ohne Beweis)}$$

- **Anmerkung**: Φ kann beliebig gewählt werden (z.B. $\Phi(s) = 42$ für alle $s \in S$), jedoch führen nicht alle Φ zu sinnvollen amortisierten Schranken.

Potentialfunktion für Fibonacci-Heaps

$$\Phi(s) := \mathit{roots}(s) + 2 \cdot \mathit{marks}(s)$$

- $\mathit{roots}(s) =$ Anzahl an Wurzeln im Zustand s
- $\mathit{marks}(s) =$ Anzahl markierter Knoten im Zustand s

Fibonacci Heaps - Delete Min

Amortisierte Analyse

Wiederholung: $\Phi(s) := \mathit{roots}(s) + 2 \cdot \mathit{marks}(s)$

- `DeleteMin` überführt Heap von Zustand s nach s'
- Laufzeit `DeleteMin`: $T_{\texttt{DeleteMin}}(s) = \mathcal{O}(\mathit{roots}(s) + \mathit{maxRank}(s))$
  - $\mathit{roots}(s) =$ Anzahl an **merge** Operationen
  - $\mathit{maxRank}(s) =$ Maximale Anzahl an Kindern eines Knoten im Zustand s
- Amortisierte Kosten: $A_{\texttt{DeleteMin}}(s) = T_{\texttt{DeleteMin}}(s) + \Delta\Phi(s)$
- $\Delta\Phi(s) = \Delta\mathit{roots}(s) + 2 \cdot \Delta\mathit{marks}(s) \leq \mathit{maxRank}(s) + 1 - \mathit{roots}(s)$
  - $\mathit{marks}(s') \leq \mathit{marks}(s) \Rightarrow \Delta\mathit{marks}(s) \leq 0$
  - $\mathit{roots}(s') \leq \mathit{maxRank}(s) + 1 \Rightarrow \Delta\mathit{roots}(s) \leq \mathit{maxRank}(s) + 1 - \mathit{roots}(s)$

$$\Rightarrow A_{\texttt{DeleteMin}}(s) = \mathcal{O}(\mathit{roots}(s) + \mathit{maxRank}(s)) + \mathit{maxRank}(s) + 1 - \mathit{roots}(s) = \mathcal{O}(\mathit{maxRank}(s))$$

Fibonacci Heaps - Delete Min

Amortisierte Analyse

Wiederholung: $\Phi(s) := \mathit{roots}(s) + 2 \cdot \mathit{marks}(s)$

- `DeleteMin` überführt Heap von Zustand s nach s'
- Laufzeit `DeleteMin`: $T_{\texttt{DeleteMin}}(s) = \mathcal{O}(\mathit{roots}(s) + \mathit{maxRank}(s))$
  - $\mathit{roots}(s) =$ Anzahl an **merge** Operationen
  - $\mathit{maxRank}(s) =$ Maximale Anzahl an Kindern eines Knoten im Zustand s
- Amortisierte Kosten: $A_{\texttt{DeleteMin}}(s) = T_{\texttt{DeleteMin}}(s) + \Delta\Phi(s)$
- $\Delta\Phi(s) = \Delta\mathit{roots}(s) + 2 \cdot \Delta\mathit{marks}(s) \leq \mathit{maxRank}(s) + 1 - \mathit{roots}(s)$
  - $\mathit{marks}(s') \leq \mathit{marks}(s) \Rightarrow \Delta\mathit{marks}(s) \leq 0$
  - $\mathit{roots}(s') \leq \mathit{maxRank}(s) + 1 \Rightarrow \Delta\mathit{roots}(s) \leq \mathit{maxRank}(s) + 1 - \mathit{roots}(s)$

$$\Rightarrow A_{\texttt{DeleteMin}}(s) = \mathcal{O}(\mathit{roots}(s) + \mathit{maxRank}(s)) + \mathit{maxRank}(s) + 1 - \mathit{roots}(s) = \mathcal{O}(\mathit{maxRank}(s))$$

Fibonacci Heaps - Delete Min

Amortisierte Analyse

Wiederholung: $\Phi(s) := \mathit{roots}(s) + 2 \cdot \mathit{marks}(s)$

- `DeleteMin` überführt Heap von Zustand s nach s'
- Laufzeit `DeleteMin`: $T_{\texttt{DeleteMin}}(s) = \mathcal{O}(\mathit{roots}(s) + \mathit{maxRank}(s))$
  - $\mathit{roots}(s) =$ Anzahl an **merge** Operationen
  - $\mathit{maxRank}(s) =$ Maximale Anzahl an Kindern eines Knoten im Zustand s
- Amortisierte Kosten: $A_{\texttt{DeleteMin}}(s) = T_{\texttt{DeleteMin}}(s) + \Delta\Phi(s)$
- $\Delta\Phi(s) = \Delta\mathit{roots}(s) + 2 \cdot \Delta\mathit{marks}(s) \leq \mathit{maxRank}(s) + 1 - \mathit{roots}(s)$
  - $\mathit{marks}(s') \leq \mathit{marks}(s) \Rightarrow \Delta\mathit{marks}(s) \leq 0$
  - $\mathit{roots}(s') \leq \mathit{maxRank}(s) + 1 \Rightarrow \Delta\mathit{roots}(s) \leq \mathit{maxRank}(s) + 1 - \mathit{roots}(s)$

$$\Rightarrow A_{\texttt{DeleteMin}}(s) = \mathcal{O}(\mathit{roots}(s) + \mathit{maxRank}(s)) + \mathit{maxRank}(s) + 1 - \mathit{roots}(s) = \mathcal{O}(\mathit{maxRank}(s))$$

Fibonacci Heaps - Delete Min

Amortisierte Analyse

Wiederholung: $\Phi(s) := \mathit{roots}(s) + 2 \cdot \mathit{marks}(s)$

- `DeleteMin` überführt Heap von Zustand s nach s'
- Laufzeit `DeleteMin`: $T_{\texttt{DeleteMin}}(s) = \mathcal{O}(\mathit{roots}(s) + \mathit{maxRank}(s))$
  - $\mathit{roots}(s) =$ Anzahl an **merge** Operationen
  - $\mathit{maxRank}(s) =$ Maximale Anzahl an Kindern eines Knoten im Zustand s
- Amortisierte Kosten: $A_{\texttt{DeleteMin}}(s) = T_{\texttt{DeleteMin}}(s) + \Delta\Phi(s)$
- $\Delta\Phi(s) = \Delta\mathit{roots}(s) + 2 \cdot \Delta\mathit{marks}(s) \leq \mathit{maxRank}(s) + 1 - \mathit{roots}(s)$
  - $\mathit{marks}(s') \leq \mathit{marks}(s) \Rightarrow \Delta\mathit{marks}(s) \leq 0$
  - $\mathit{roots}(s') \leq \mathit{maxRank}(s) + 1 \Rightarrow \Delta\mathit{roots}(s) \leq \mathit{maxRank}(s) + 1 - \mathit{roots}(s)$

$$\Rightarrow A_{\texttt{DeleteMin}}(s) = \mathcal{O}(\mathit{roots}(s) + \mathit{maxRank}(s)) + \mathit{maxRank}(s) + 1 - \mathit{roots}(s) = \mathcal{O}(\mathit{maxRank}(s))$$

Fibonacci Heaps - Delete Min

Amortisierte Analyse

Wiederholung: $\Phi(s) := \mathit{roots}(s) + 2 \cdot \mathit{marks}(s)$

- `DeleteMin` überführt Heap von Zustand s nach s'
- Laufzeit `DeleteMin`: $T_{\texttt{DeleteMin}}(s) = \mathcal{O}(\mathit{roots}(s) + \mathit{maxRank}(s))$
  - $\mathit{roots}(s) =$ Anzahl an **merge** Operationen
  - $\mathit{maxRank}(s) =$ Maximale Anzahl an Kindern eines Knoten im Zustand s
- Amortisierte Kosten: $A_{\texttt{DeleteMin}}(s) = T_{\texttt{DeleteMin}}(s) + \Delta\Phi(s)$
- $\Delta\Phi(s) = \Delta\mathit{roots}(s) + 2 \cdot \Delta\mathit{marks}(s) \leq \mathit{maxRank}(s) + 1 - \mathit{roots}(s)$
  - $\mathit{marks}(s') \leq \mathit{marks}(s) \Rightarrow \Delta\mathit{marks}(s) \leq 0$
  - $\mathit{roots}(s') \leq \mathit{maxRank}(s) + 1 \Rightarrow \Delta\mathit{roots}(s) \leq \mathit{maxRank}(s) + 1 - \mathit{roots}(s)$

$$\Rightarrow A_{\texttt{DeleteMin}}(s) = \mathcal{O}(\mathit{roots}(s) + \mathit{maxRank}(s)) + \mathit{maxRank}(s) + 1 - \mathit{roots}(s) = \mathcal{O}(\mathit{maxRank}(s))$$

Fibonacci Heaps - Decrease Key

Amortisierte Analyse

Wiederholung: $\Phi(s) := \mathit{roots}(s) + 2 \cdot \mathit{marks}(s)$

- $\texttt{DecreaseKey}(h)$ überführt Heap von Zustand s nach s'
- Laufzeit $\texttt{DecreaseKey}$: $T_{\texttt{DecreaseKey}}(s) = \mathit{cuts}(h, s) + 1$
  - $\mathit{cuts}(h, s)$ = Anzahl an markierten *direkten* Parents von Handle h im Zustand s (Cascading Cuts)
- Amortisierte Kosten: $A_{\texttt{DeleteMin}}(s) = T_{\texttt{DeleteMin}}(s) + \Delta\Phi(s)$
- $\Delta\Phi(s) = \Delta\mathit{roots}(s) + 2 \cdot \Delta\mathit{marks}(s) = 4 - \mathit{cuts}(h, s)$
  - $\mathit{marks}(s') = \mathit{marks}(s) - \mathit{cuts}(h, s) + 2 \Rightarrow \Delta\mathit{marks}(s') = 2 - \mathit{cuts}(h, s)$
  - $\mathit{roots}(s') = \mathit{roots}(s) + \mathit{cuts}(h, s) \Rightarrow \Delta\mathit{roots}(s) = \mathit{cuts}(h, s)$

$$\Rightarrow A_{\texttt{DeleteMin}}(s) = \mathit{cuts}(h,s) + 1 + 4 - \mathit{cuts}(h,s) = \mathcal{O}(1)$$

Fibonacci Heaps - Decrease Key

Amortisierte Analyse

Wiederholung: $\Phi(s) := \mathit{roots}(s) + 2 \cdot \mathit{marks}(s)$

- $\texttt{DecreaseKey}(h)$ überführt Heap von Zustand s nach s'
- Laufzeit $\texttt{DecreaseKey}$: $T_{\texttt{DecreaseKey}}(s) = \mathit{cuts}(h, s) + 1$
  - $\mathit{cuts}(h, s) =$ Anzahl an markierten *direkten* Parents von Handle h im Zustand s (Cascading Cuts)
- Amortisierte Kosten: $A_{\texttt{DeleteMin}}(s) = T_{\texttt{DeleteMin}}(s) + \Delta\Phi(s)$
- $\Delta\Phi(s) = \Delta\mathit{roots}(s) + 2 \cdot \Delta\mathit{marks}(s) = 4 - \mathit{cuts}(h, s)$
  - $\mathit{marks}(s') = \mathit{marks}(s) - \mathit{cuts}(h, s) + 2 \Rightarrow \Delta\mathit{marks}(s') = 2 - \mathit{cuts}(h, s)$
  - $\mathit{roots}(s') = \mathit{roots}(s) + \mathit{cuts}(h, s) \Rightarrow \Delta\mathit{roots}(s) = \mathit{cuts}(h, s)$

$$\Rightarrow A_{\texttt{DeleteMin}}(s) = \mathit{cuts}(h,s) + 1 + 4 - \mathit{cuts}(h,s) = \mathcal{O}(1)$$

Fibonacci Heaps - Decrease Key

Amortisierte Analyse

Wiederholung: $\Phi(s) := \mathit{roots}(s) + 2 \cdot \mathit{marks}(s)$

- $\texttt{DecreaseKey}(h)$ überführt Heap von Zustand s nach s'
- Laufzeit $\texttt{DecreaseKey}$: $T_{\texttt{DecreaseKey}}(s) = \mathit{cuts}(h, s) + 1$
  - $\mathit{cuts}(h, s) =$ Anzahl an markierten *direkten* Parents von Handle h im Zustand s (Cascading Cuts)
- Amortisierte Kosten: $A_{\texttt{DeleteMin}}(s) = T_{\texttt{DeleteMin}}(s) + \Delta\Phi(s)$
- $\Delta\Phi(s) = \Delta\mathit{roots}(s) + 2 \cdot \Delta\mathit{marks}(s) = 4 - \mathit{cuts}(h, s)$
  - $\mathit{marks}(s') = \mathit{marks}(s) - \mathit{cuts}(h, s) + 2 \Rightarrow \Delta\mathit{marks}(s') = 2 - \mathit{cuts}(h, s)$
  - $\mathit{roots}(s') = \mathit{roots}(s) + \mathit{cuts}(h, s) \Rightarrow \Delta\mathit{roots}(s) = \mathit{cuts}(h, s)$

$$\Rightarrow A_{\texttt{DeleteMin}}(s) = \mathit{cuts}(h,s) + 1 + 4 - \mathit{cuts}(h,s) = \mathcal{O}(1)$$

Fibonacci Heaps - Decrease Key

Amortisierte Analyse

Wiederholung: $\Phi(s) := \mathit{roots}(s) + 2 \cdot \mathit{marks}(s)$

- $\texttt{DecreaseKey}(h)$ überführt Heap von Zustand s nach s'
- Laufzeit $\texttt{DecreaseKey}$: $T_{\texttt{DecreaseKey}}(s) = \mathit{cuts}(h, s) + 1$
  - $\mathit{cuts}(h, s) =$ Anzahl an markierten *direkten* Parents von Handle h im Zustand s (Cascading Cuts)
- Amortisierte Kosten: $A_{\texttt{DeleteMin}}(s) = T_{\texttt{DeleteMin}}(s) + \Delta\Phi(s)$
- $\Delta\Phi(s) = \Delta\mathit{roots}(s) + 2 \cdot \Delta\mathit{marks}(s) = 4 - \mathit{cuts}(h, s)$
  - $\mathit{marks}(s') = \mathit{marks}(s) - \mathit{cuts}(h, s) + 2 \Rightarrow \Delta\mathit{marks}(s') = 2 - \mathit{cuts}(h, s)$
  - $\mathit{roots}(s') = \mathit{roots}(s) + \mathit{cuts}(h, s) \Rightarrow \Delta\mathit{roots}(s) = \mathit{cuts}(h, s)$

$$\Rightarrow A_{\texttt{DeleteMin}}(s) = \mathit{cuts}(h,s) + 1 + 4 - \mathit{cuts}(h,s) = \mathcal{O}(1)$$

Fibonacci Heaps - Decrease Key

Amortisierte Analyse

Wiederholung: $\Phi(s) := \mathit{roots}(s) + 2 \cdot \mathit{marks}(s)$

- $\texttt{DecreaseKey}(h)$ überführt Heap von Zustand s nach s'
- Laufzeit $\texttt{DecreaseKey}$: $T_{\texttt{DecreaseKey}}(s) = \mathit{cuts}(h, s) + 1$
  - $\mathit{cuts}(h, s) =$ Anzahl an markierten *direkten* Parents von Handle h im Zustand s (Cascading Cuts)
- Amortisierte Kosten: $A_{\texttt{DeleteMin}}(s) = T_{\texttt{DeleteMin}}(s) + \Delta\Phi(s)$
- $\Delta\Phi(s) = \Delta\mathit{roots}(s) + 2 \cdot \Delta\mathit{marks}(s) = 4 - \mathit{cuts}(h, s)$
  - $\mathit{marks}(s') = \mathit{marks}(s) - \mathit{cuts}(h, s) + 2 \Rightarrow \Delta\mathit{marks}(s') = 2 - \mathit{cuts}(h, s)$
  - $\mathit{roots}(s') = \mathit{roots}(s) + \mathit{cuts}(h, s) \Rightarrow \Delta\mathit{roots}(s) = \mathit{cuts}(h, s)$

$$\Rightarrow A_{\texttt{DeleteMin}}(s) = \mathit{cuts}(h,s) + 1 + 4 - \mathit{cuts}(h,s) = \mathcal{O}(1)$$

Fibonacci Heaps

Schranke für $maxRank(s)$

Wiederholung: $A_{\texttt{DeleteMin}}(s) \in \mathcal{O}(maxRank(s))$

- Ohne `DecreaseKey` $\Rightarrow$ Trees sind Binomialbäume
  - Binomialbaum B_i mit Rank i enthält $|B_i| = 2^i \leq n$ Knoten
    $\Rightarrow maxRank(s) \leq \log_2(n)$
- Mit `DecreaseKey`
  - Kann Knoten aus einem Binomialbaum entfernen
  - $\Rightarrow |B_i| \neq 2^i$
  - aber $maxRank(s) \leq \log_\varphi(n)$ mit $\varphi \approx 1.619$

Fibonacci Heaps

Schranke für $maxRank(s)$

Wiederholung: $A_{\texttt{DeleteMin}}(s) \in \mathcal{O}(maxRank(s))$

- Ohne `DecreaseKey` $\Rightarrow$ Trees sind Binomialbäume
  - Binomialbaum B_i mit Rank i enthält $|B_i| = 2^i \leq n$ Knoten
    $\Rightarrow maxRank(s) \leq \log_2(n)$
- Mit `DecreaseKey`
  - Kann Knoten aus einem Binomialbaum entfernen
  - $\Rightarrow |B_i| \neq 2^i$
  - aber $maxRank(s) \leq \log_\varphi(n)$ mit $\varphi \approx 1.619$

Fibonacci Heaps

Schranke für $maxRank(s)$

- $N_i =$ die Anzahl an Knoten in Baum mit Wurzel r und $rank(r) = i$
- **Annhame:** $N_i \geq F_{i+2}$ wobei $F_i =$ Fibonacci Reihe

  $F_{i+2} = F_{i+1} + F_i$ mit $F_0 = 0$ und $F_1 = 1$

  - $w_1, \ldots w_i$ sind Kinder von r sortiert nach *merge*-Zeitpunkt
  - $rank(w_j) \geq j - 2$ $(1 \leq j \leq i)$, da zum *merge*-Zeitpunkt $rank(w_j) = rank(r) = j - 1$ und maximal ein Kind von w_j durch `DecreaseKey` entfernt wurde (w_j danach markiert)
  - $\Rightarrow N_i \geq 2 + N_0 + N_1 + \cdots + N_{i-2}$ mit $N_0 = 1$ und $N_1 = 2$
  - **Übung:** $F_{i+2} = 2 + N_0 + N_1 + \cdots + N_{i-2}$ mit $N_0 = 1$ und $N_1 = 2$

$$\Rightarrow n \geq N_i \geq F_{i+2} \geq \varphi^i \Rightarrow maxRank(s) \leq \log_\varphi(n)$$

Fibonacci Heaps

Schranke für $maxRank(s)$

- $N_i =$ die Anzahl an Knoten in Baum mit Wurzel r und $rank(r) = i$
- **Annhame:** $N_i \geq F_{i+2}$ wobei $F_i =$ Fibonacci Reihe

  $F_{i+2} = F_{i+1} + F_i$ mit $F_0 = 0$ und $F_1 = 1$

  - $w_1, \ldots w_i$ sind Kinder von r sortiert nach *merge*-Zeitpunkt
  - $rank(w_j) \geq j - 2$ $(1 \leq j \leq i)$, da zum *merge*-Zeitpunkt $rank(w_j) = rank(r) = j - 1$ und maximal ein Kind von w_j durch `DecreaseKey` entfernt wurde (w_j danach markiert)
  - $\Rightarrow N_i \geq 2 + N_0 + N_1 + \cdots + N_{i-2}$ mit $N_0 = 1$ und $N_1 = 2$
  - **Übung:** $F_{i+2} = 2 + N_0 + N_1 + \cdots + N_{i-2}$ mit $N_0 = 1$ und $N_1 = 2$

$$\Rightarrow n \geq N_i \geq F_{i+2} \geq \varphi^i \Rightarrow maxRank(s) \leq \log_\varphi(n)$$

Fibonacci Heaps

Schranke für $maxRank(s)$

- $N_i =$ die Anzahl an Knoten in Baum mit Wurzel r und $rank(r) = i$
- **Annhame:** $N_i \geq F_{i+2}$ wobei $F_i =$ Fibonacci Reihe

  $F_{i+2} = F_{i+1} + F_i$ mit $F_0 = 0$ und $F_1 = 1$

  - $w_1, \ldots w_i$ sind Kinder von r sortiert nach *merge*-Zeitpunkt
  - $rank(w_j) \geq j - 2$ $(1 \leq j \leq i)$, da zum *merge*-Zeitpunkt $rank(w_j) = rank(r) = j - 1$ und maximal ein Kind von w_j durch `DecreaseKey` entfernt wurde (w_j danach markiert)
  - $\Rightarrow N_i \geq 2 + N_0 + N_1 + \cdots + N_{i-2}$ mit $N_0 = 1$ und $N_1 = 2$
  - **Übung:** $F_{i+2} = 2 + N_0 + N_1 + \cdots + N_{i-2}$ mit $N_0 = 1$ und $N_1 = 2$

$$\Rightarrow n \geq N_i \geq F_{i+2} \geq \varphi^i \Rightarrow maxRank(s) \leq \log_\varphi(n)$$

Fibonacci Heaps

Schranke für $maxRank(s)$

- $N_i =$ die Anzahl an Knoten in Baum mit Wurzel r und $rank(r) = i$
- **Annhame:** $N_i \geq F_{i+2}$ wobei $F_i =$ Fibonacci Reihe

  $F_{i+2} = F_{i+1} + F_i$ mit $F_0 = 0$ und $F_1 = 1$

  - $w_1, \ldots w_i$ sind Kinder von r sortiert nach *merge*-Zeitpunkt
  - $rank(w_j) \geq j - 2$ $(1 \leq j \leq i)$, da zum *merge*-Zeitpunkt $rank(w_j) = rank(r) = j - 1$ und maximal ein Kind von w_j durch `DecreaseKey` entfernt wurde (w_j danach markiert)
  - $\Rightarrow N_i \geq 2 + N_0 + N_1 + \cdots + N_{i-2}$ mit $N_0 = 1$ und $N_1 = 2$
  - **Übung:** $F_{i+2} = 2 + N_0 + N_1 + \cdots + N_{i-2}$ mit $N_0 = 1$ und $N_1 = 2$

$$\Rightarrow n \geq N_i \geq F_{i+2} \geq \varphi^i \Rightarrow maxRank(s) \leq \log_\varphi(n)$$

Fibonacci Heaps

Schranke für $maxRank(s)$

- $N_i =$ die Anzahl an Knoten in Baum mit Wurzel r und $rank(r) = i$
- **Annhame:** $N_i \geq F_{i+2}$ wobei $F_i =$ Fibonacci Reihe

  $F_{i+2} = F_{i+1} + F_i$ mit $F_0 = 0$ und $F_1 = 1$

  - $w_1, \ldots w_i$ sind Kinder von r sortiert nach *merge*-Zeitpunkt
  - $rank(w_j) \geq j - 2$ $(1 \leq j \leq i)$, da zum *merge*-Zeitpunkt $rank(w_j) = rank(r) = j - 1$ und maximal ein Kind von w_j durch `DecreaseKey` entfernt wurde (w_j danach markiert)
  - $\Rightarrow N_i \geq 2 + N_0 + N_1 + \cdots + N_{i-2}$ mit $N_0 = 1$ und $N_1 = 2$
  - **Übung:** $F_{i+2} = 2 + N_0 + N_1 + \cdots + N_{i-2}$ mit $N_0 = 1$ und $N_1 = 2$

$$\Rightarrow n \geq N_i \geq F_{i+2} \geq \varphi^i \Rightarrow maxRank(s) \leq \log_\varphi(n)$$

Fibonacci Heaps

Schranke für $maxRank(s)$

- $N_i =$ die Anzahl an Knoten in Baum mit Wurzel r und $rank(r) = i$
- **Annhame:** $N_i \geq F_{i+2}$ wobei $F_i =$ Fibonacci Reihe

  $F_{i+2} = F_{i+1} + F_i$ mit $F_0 = 0$ und $F_1 = 1$

  - $w_1, \ldots w_i$ sind Kinder von r sortiert nach *merge*-Zeitpunkt
  - $rank(w_j) \geq j - 2$ $(1 \leq j \leq i)$, da zum *merge*-Zeitpunkt $rank(w_j) = rank(r) = j - 1$ und maximal ein Kind von w_j durch `DecreaseKey` entfernt wurde (w_j danach markiert)
  - $\Rightarrow N_i \geq 2 + N_0 + N_1 + \cdots + N_{i-2}$ mit $N_0 = 1$ und $N_1 = 2$
  - **Übung:** $F_{i+2} = 2 + N_0 + N_1 + \cdots + N_{i-2}$ mit $N_0 = 1$ und $N_1 = 2$

$$\Rightarrow n \geq N_i \geq F_{i+2} \geq \varphi^i \Rightarrow maxRank(s) \leq \log_\varphi(n)$$

Fibonacci Heaps

Schranke für $maxRank(s)$

- $N_i =$ die Anzahl an Knoten in Baum mit Wurzel r und $rank(r) = i$
- **Annhame:** $N_i \geq F_{i+2}$ wobei $F_i =$ Fibonacci Reihe

  $F_{i+2} = F_{i+1} + F_i$ mit $F_0 = 0$ und $F_1 = 1$

  - $w_1, \ldots w_i$ sind Kinder von r sortiert nach *merge*-Zeitpunkt
  - $rank(w_j) \geq j - 2$ $(1 \leq j \leq i)$, da zum *merge*-Zeitpunkt $rank(w_j) = rank(r) = j - 1$ und maximal ein Kind von w_j durch `DecreaseKey` entfernt wurde (w_j danach markiert)
  - $\Rightarrow N_i \geq 2 + N_0 + N_1 + \cdots + N_{i-2}$ mit $N_0 = 1$ und $N_1 = 2$
  - **Übung:** $F_{i+2} = 2 + N_0 + N_1 + \cdots + N_{i-2}$ mit $N_0 = 1$ und $N_1 = 2$

$$\Rightarrow n \geq N_i \geq F_{i+2} \geq \varphi^i \Rightarrow maxRank(s) \leq \log_\varphi(n)$$

Fibonacci Heaps

- Amortisierte Laufzeiten sind **optimal** (vergleichsbasiert)
- **Einzelne Operationen** Kosten aber deutlich **länger**

Ende!

Feierabend!