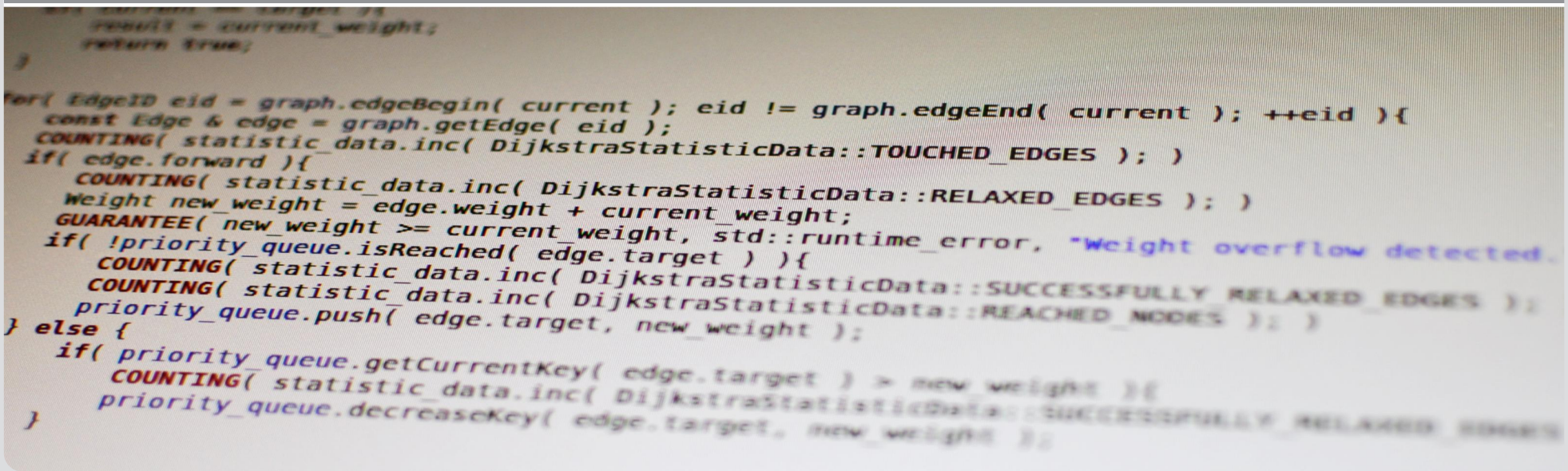
Karlsruher Institut für Technologie

Übung 1 – Algorithmen II

Hans-Peter Lehmann, Daniel Seemaier – *{hans-peter.lehmann, daniel.seemaier}@kit.edu*

`http://algo2.iti.kit.edu/AlgorithmenII_WS21.php`

Institut für Theoretische Informatik - Algorithmik II



Organisatorisches

Vorlesungen:

Mo 10.00–11.30 Neue Chemie – Vorlesung
Di 16.00–17.30 Neue Chemie – Vorlesung + Übung

Übungsblätter:

14-tägig, jeweils Montags, Musterlösung wird zusammen mit nächsten Übungsblatt veröffentlicht
1. Blatt: 25.10.2021 (gestern)

Ilias Forum:

Fragen zum Vorlesungsinhalt

Vorlesungsaufzeichnung:

Vorlesung vom letzten Jahr auf Youtube – (Link: *Algorithmen II - Playlist*)

Organisatorisches

Sprechstunden:

Corona bedingt: Ilias Forum oder E-Mail bevorzugt

Letzte Vorlesung:

08. Februar 2022

Klausur:

15.03.2022, 11:00 Uhr

Inhalt

Themenübersicht für heute

- Fibonacci Heaps
  - Wiederholung der Operationen
  - Ammortisierte Analyse (Beispiel)
  - Ammortisierte Analyse (Formal)

Legende

Karlsruher Institut für Technologie

Markierung - Kennzeichnet Knoten, die ein Kind verloren haben

Doppelte Markierung - Temporäre Markierung. Muss sofort aufgelöst werden

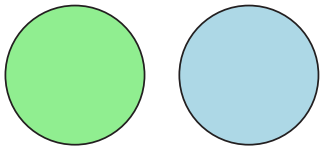
Neues, unbenutztes Token

Bezahltoken - wird zum Bezahlen genutzt

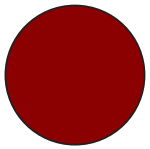
InsertTree-Token - kann genau 1 InsertTree Operation bezahlen

Union-Token - zahlt für 1 Union Operation; zusätzlich einen InsertTree-Token erstellen

Mark Token - Zahlt Cut und Markierung des Parents

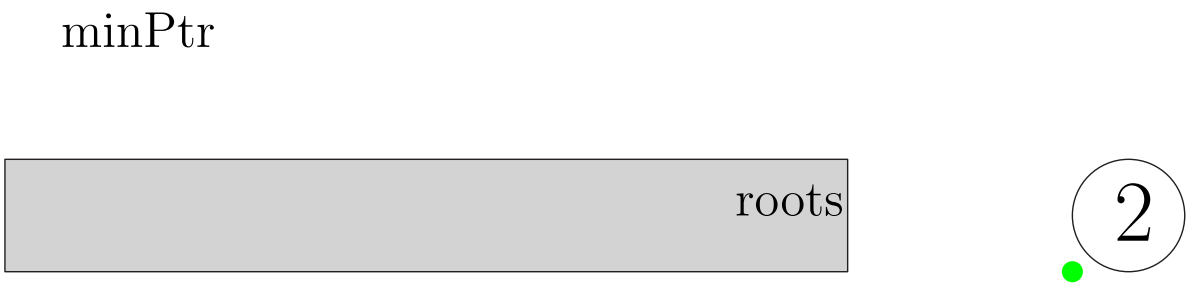


Aktive Knoten

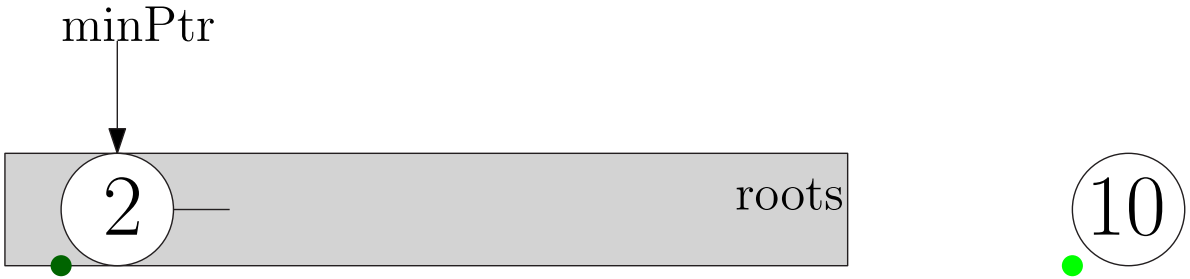


Knoten zum Löschen oder Cutten

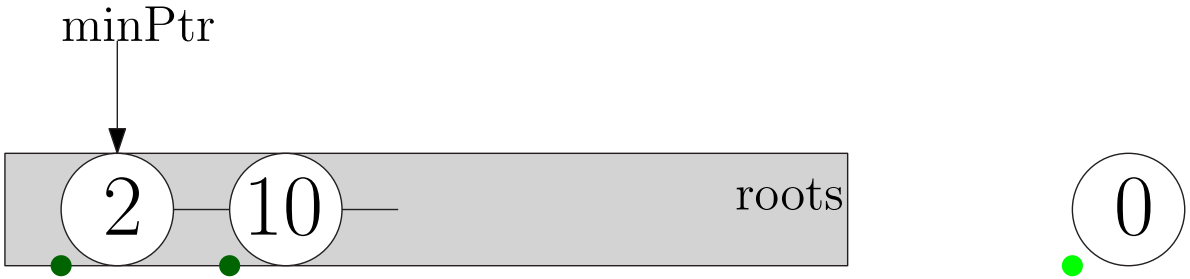
Fibonacci Heaps - Insert



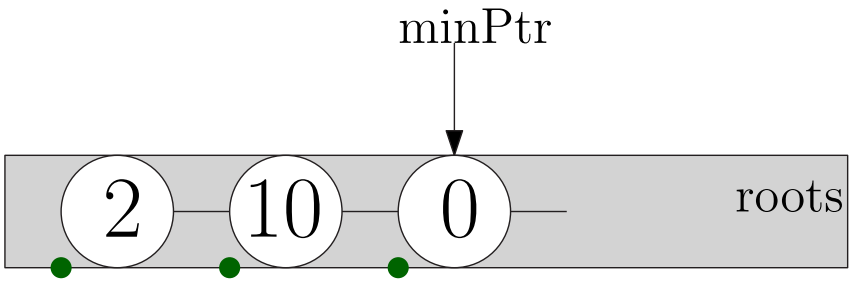
Fibonacci Heaps - Insert



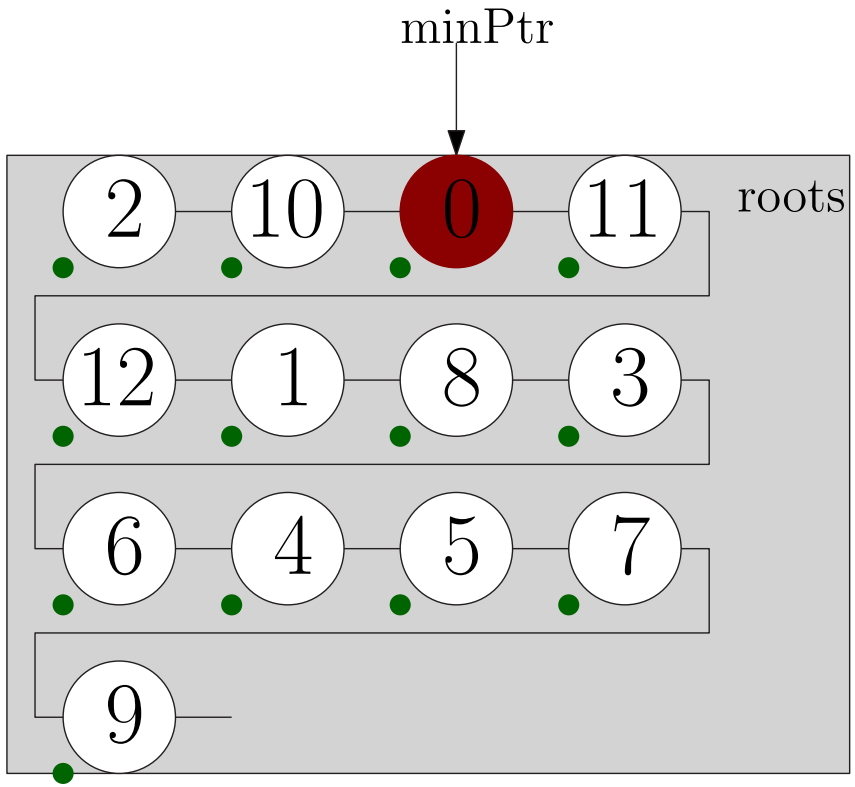
Fibonacci Heaps - Insert



Fibonacci Heaps - Insert



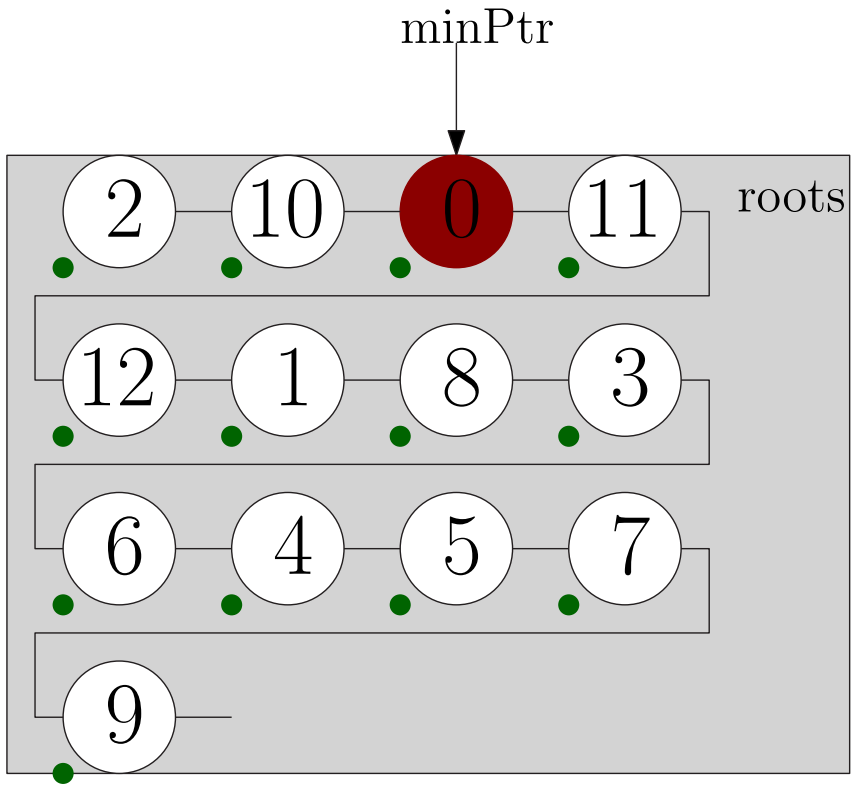
Fibonacci Heaps - Insert



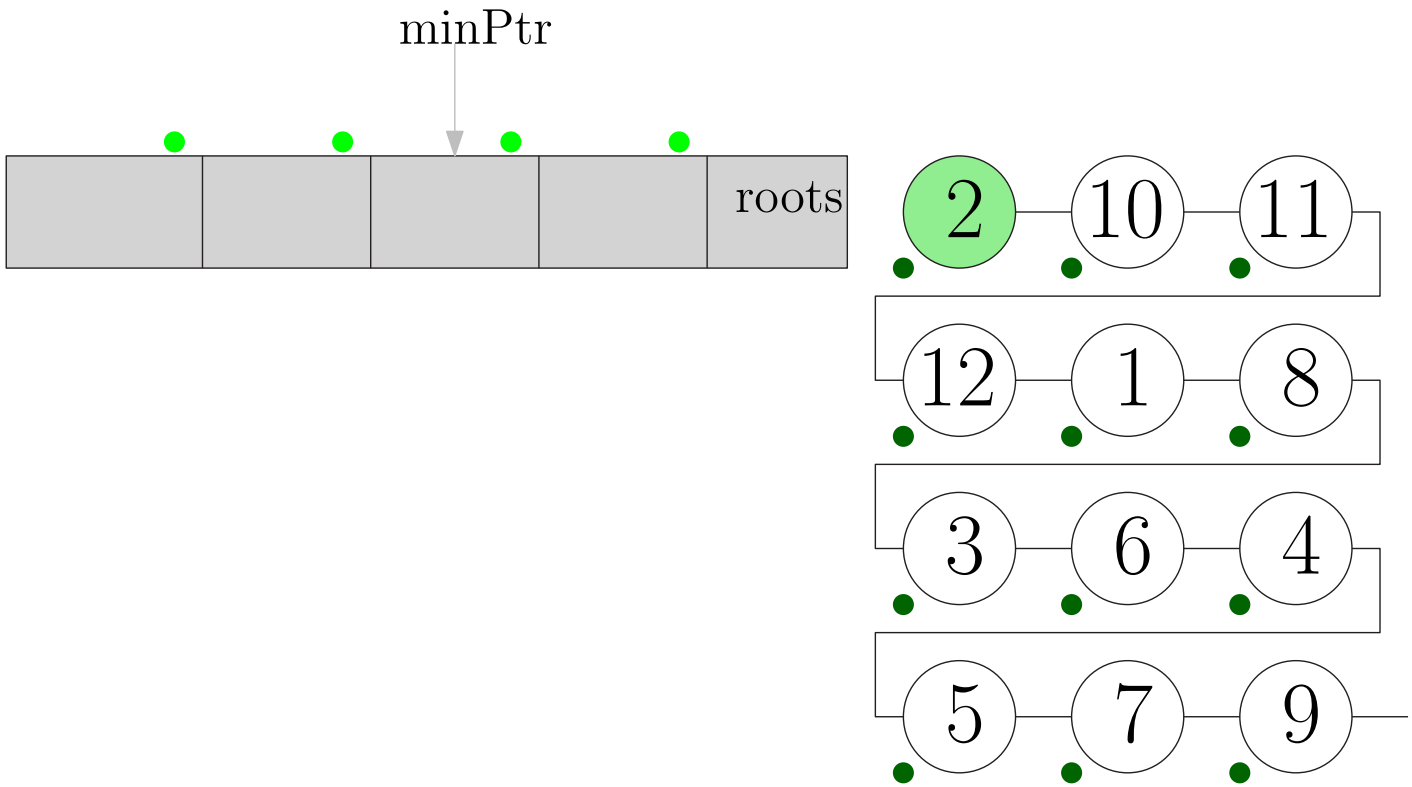
Fibonacci Heaps - Insert

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

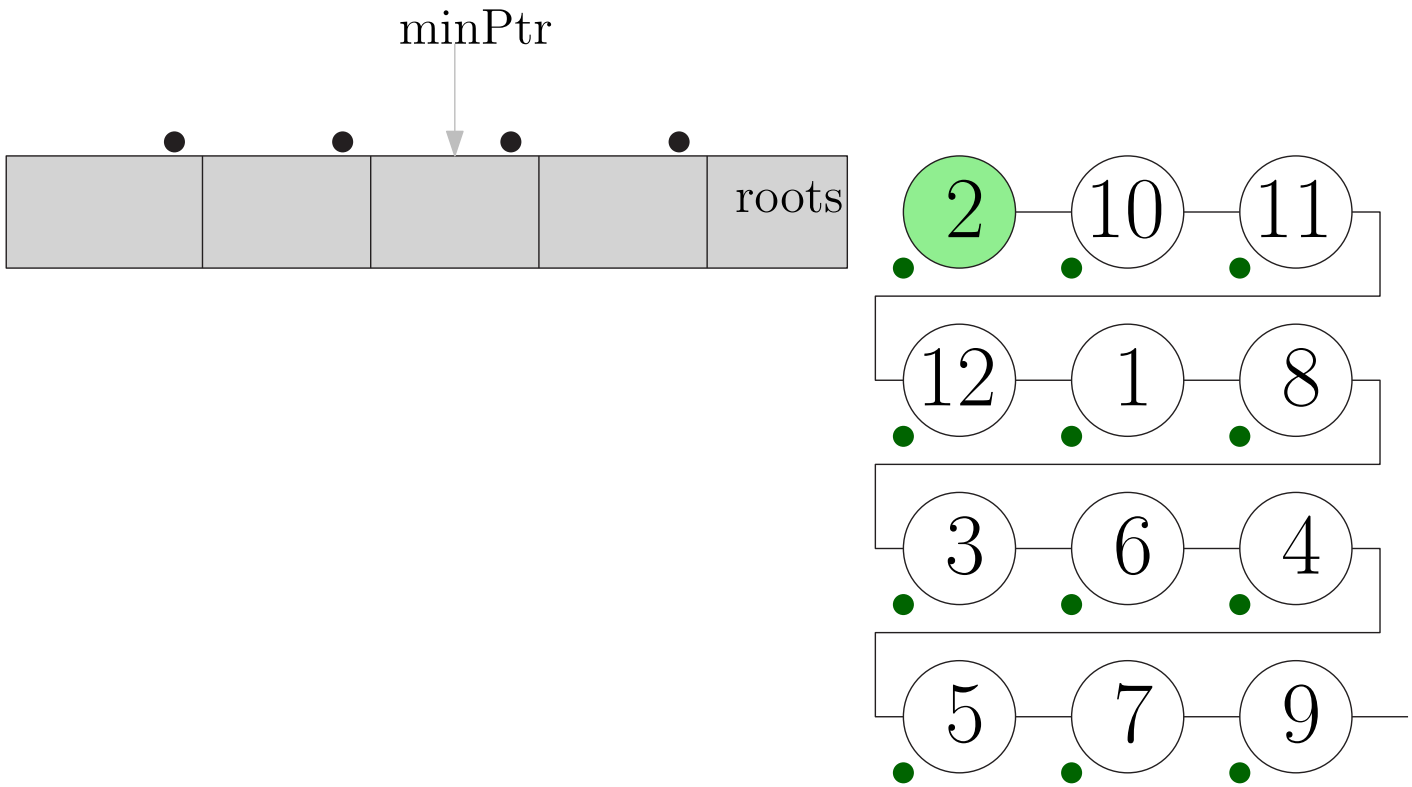
Fibonacci Heaps - Delete Min



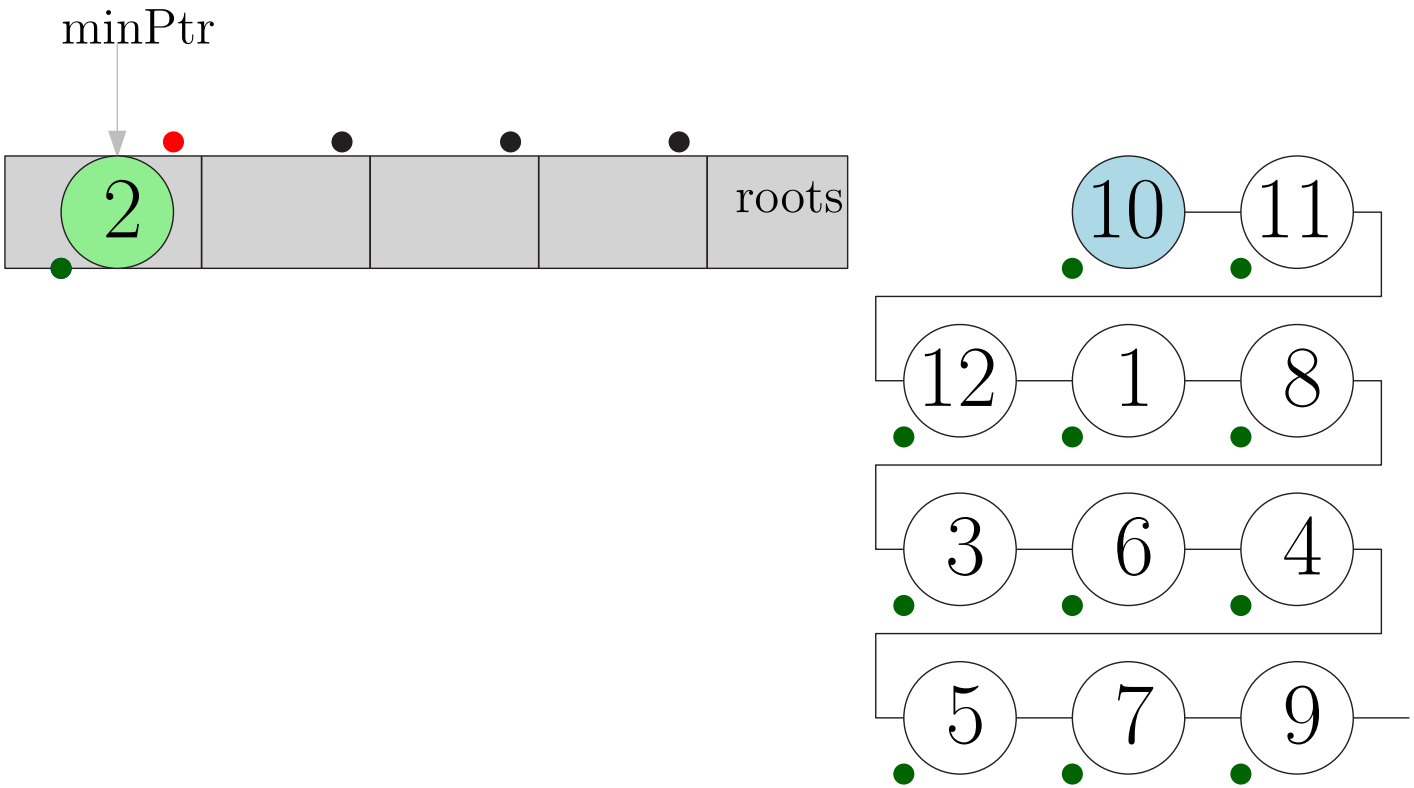
Fibonacci Heaps - Delete Min



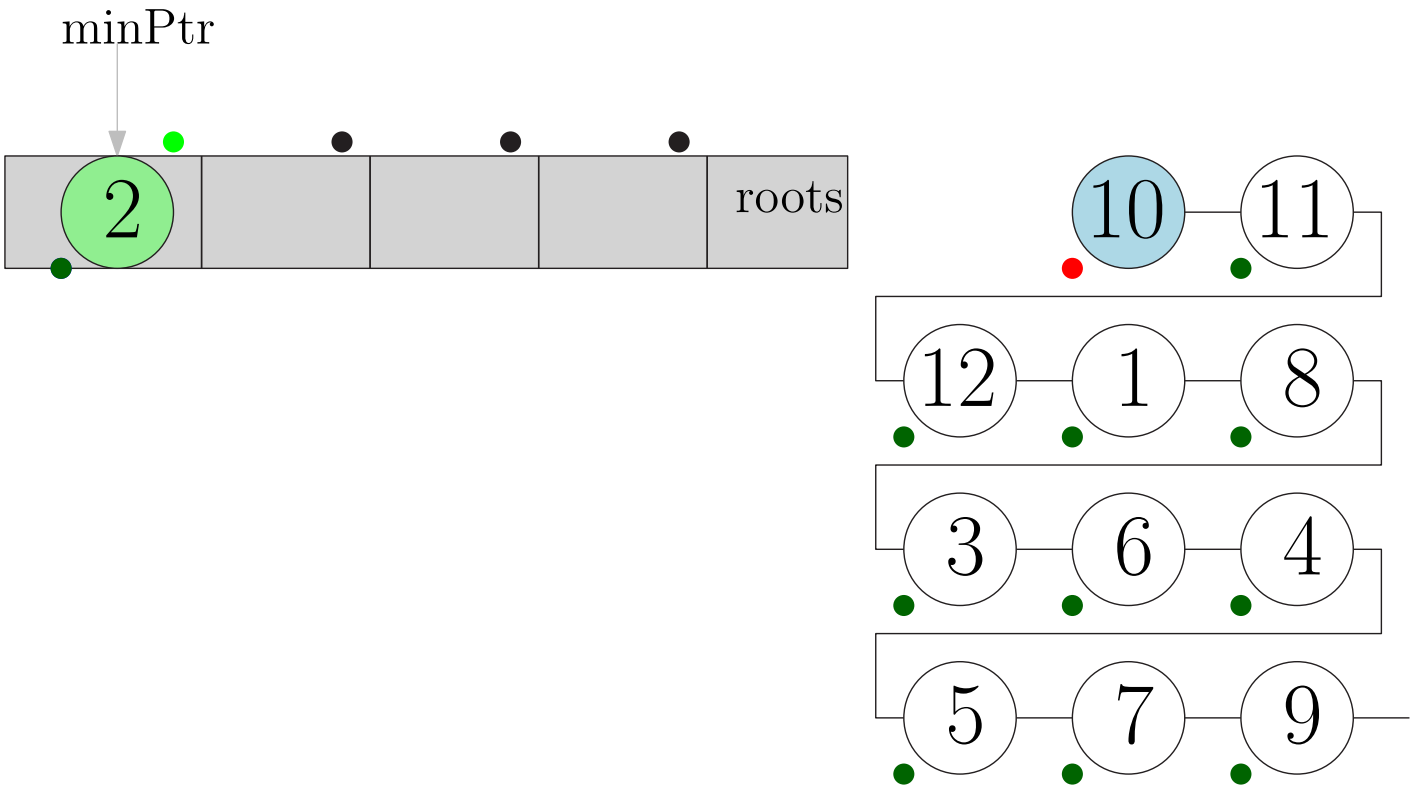
Fibonacci Heaps - Delete Min



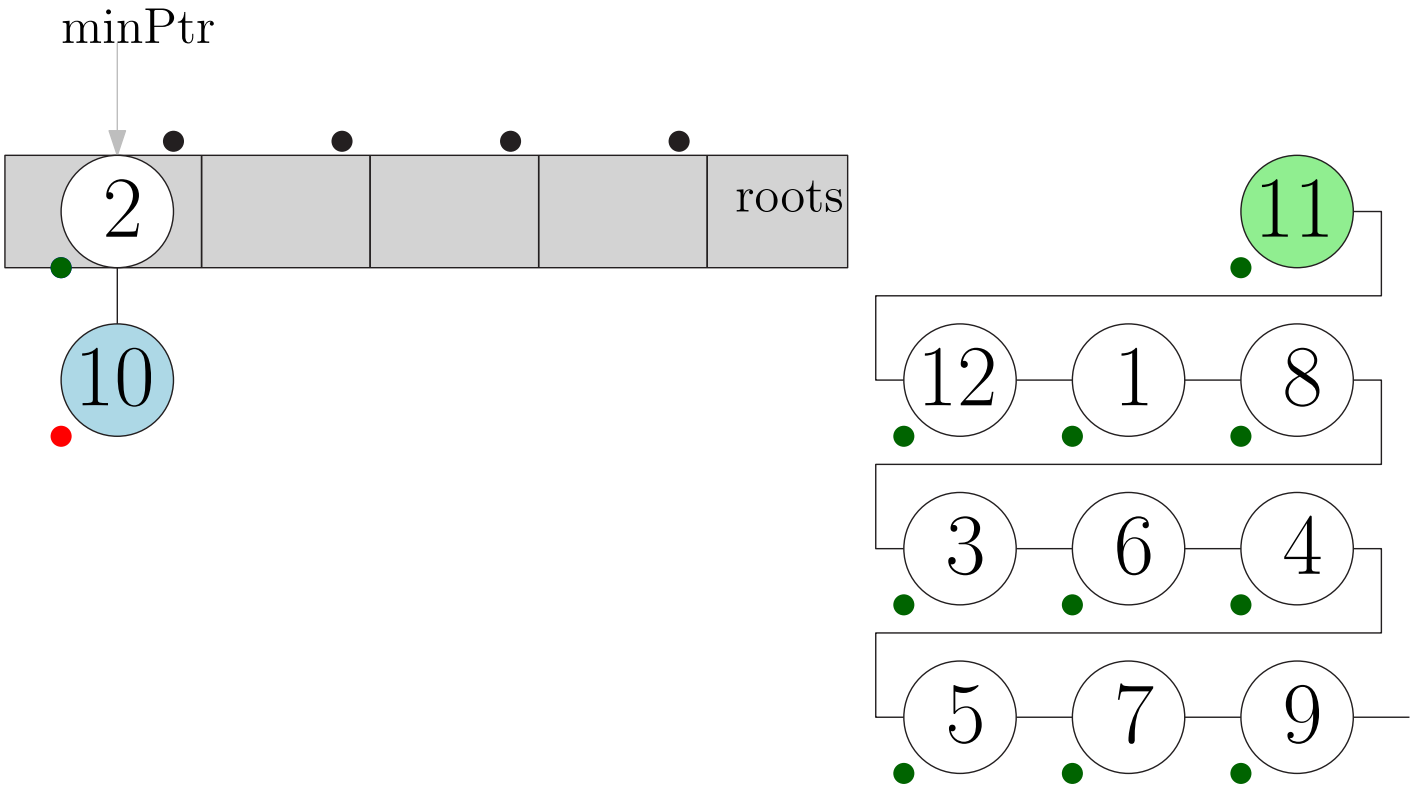
Fibonacci Heaps - Delete Min



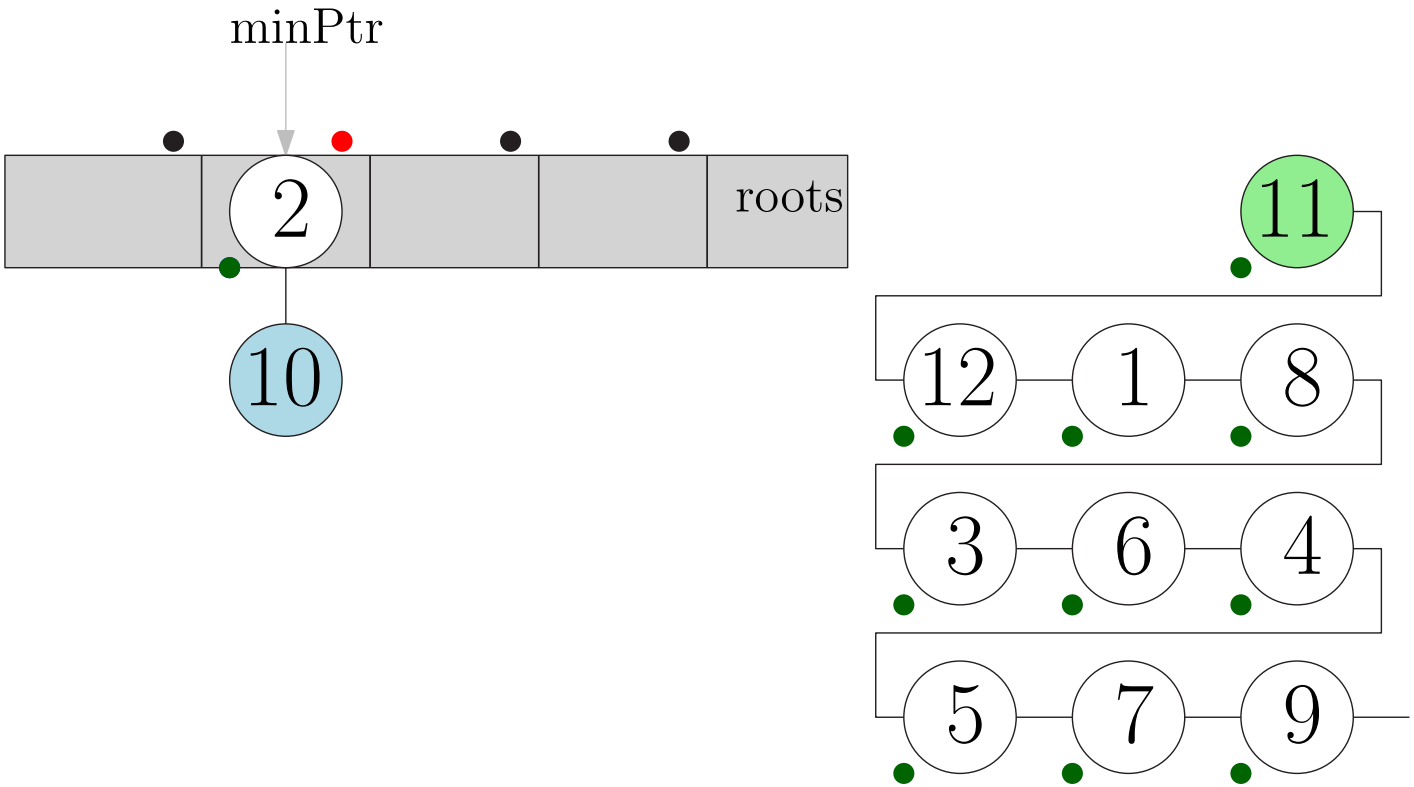
Fibonacci Heaps - Delete Min



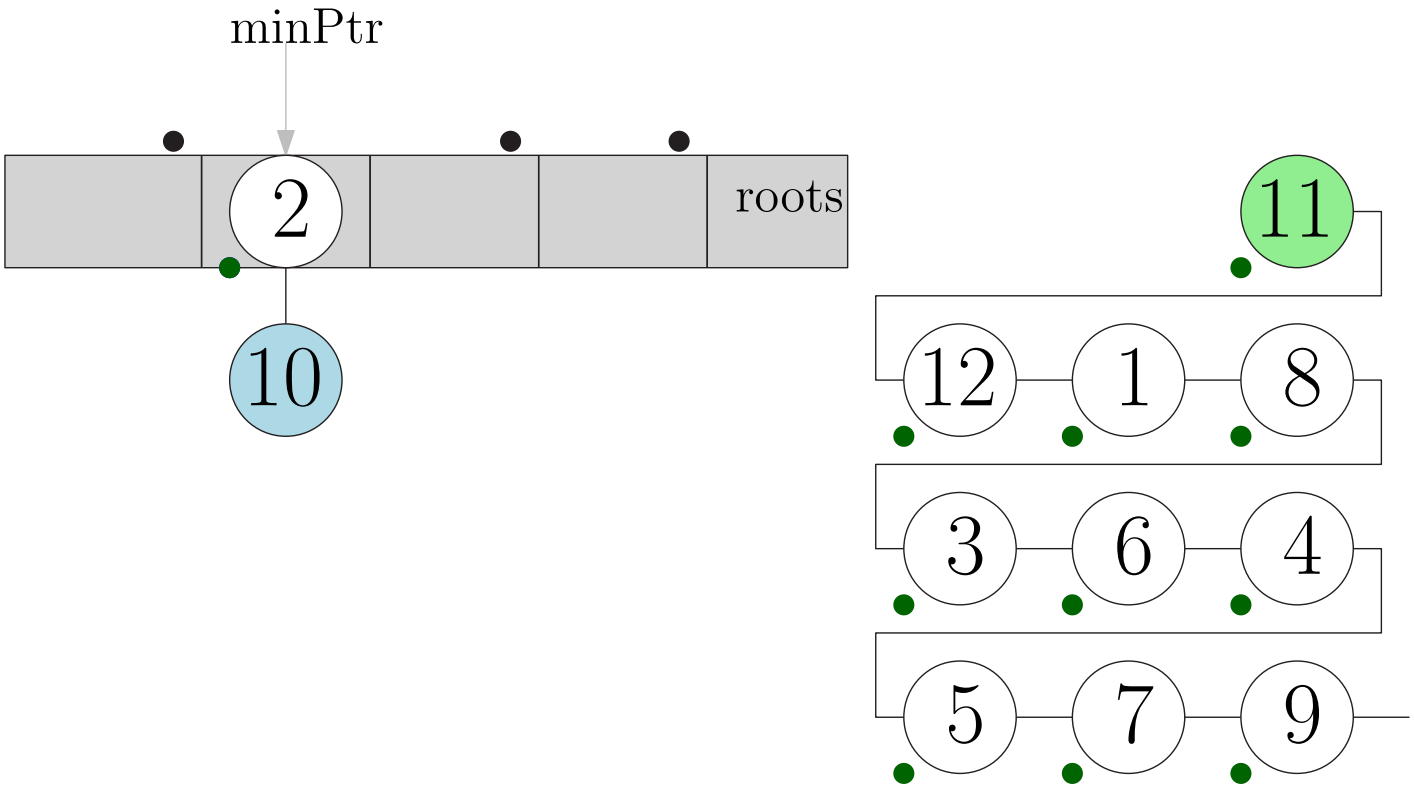
Fibonacci Heaps - Delete Min



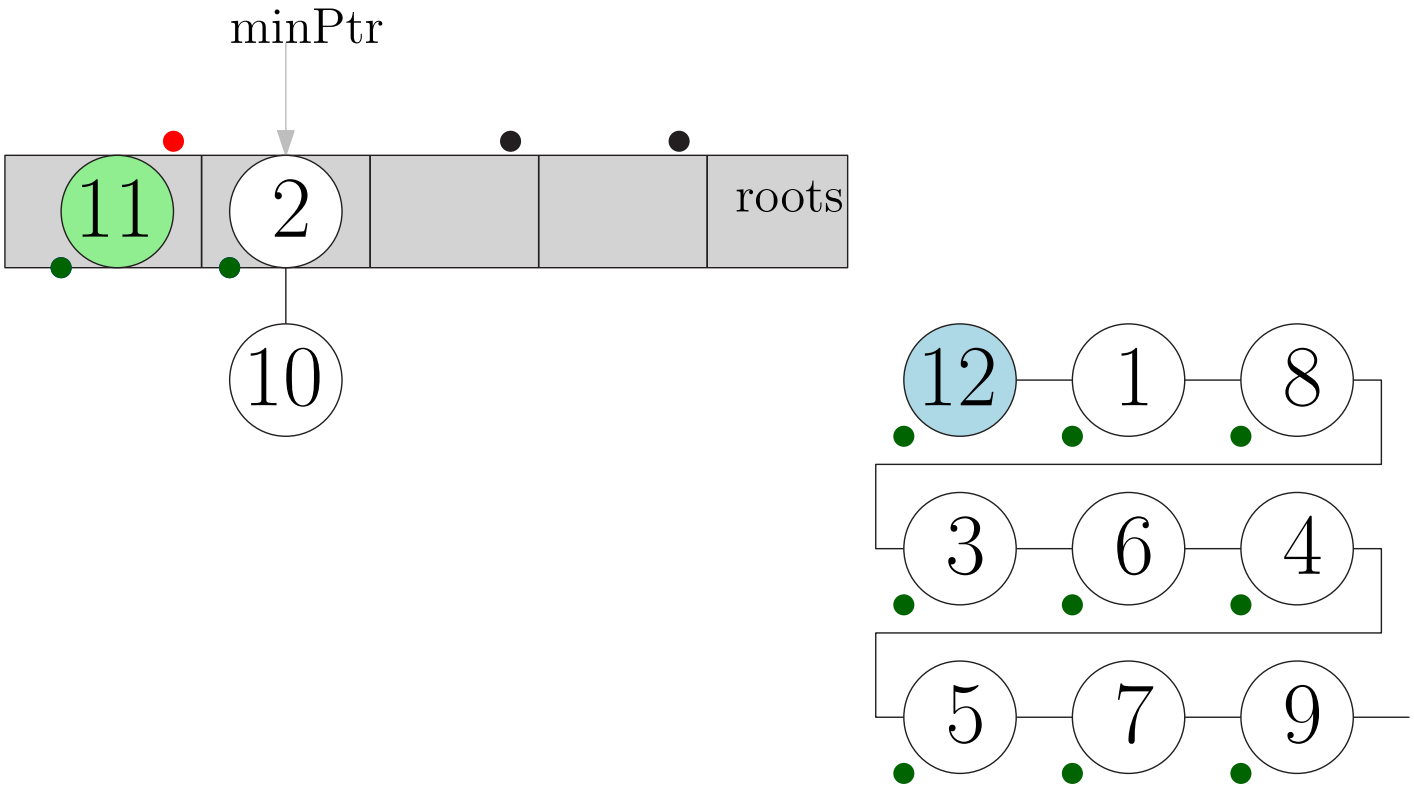
Fibonacci Heaps - Delete Min



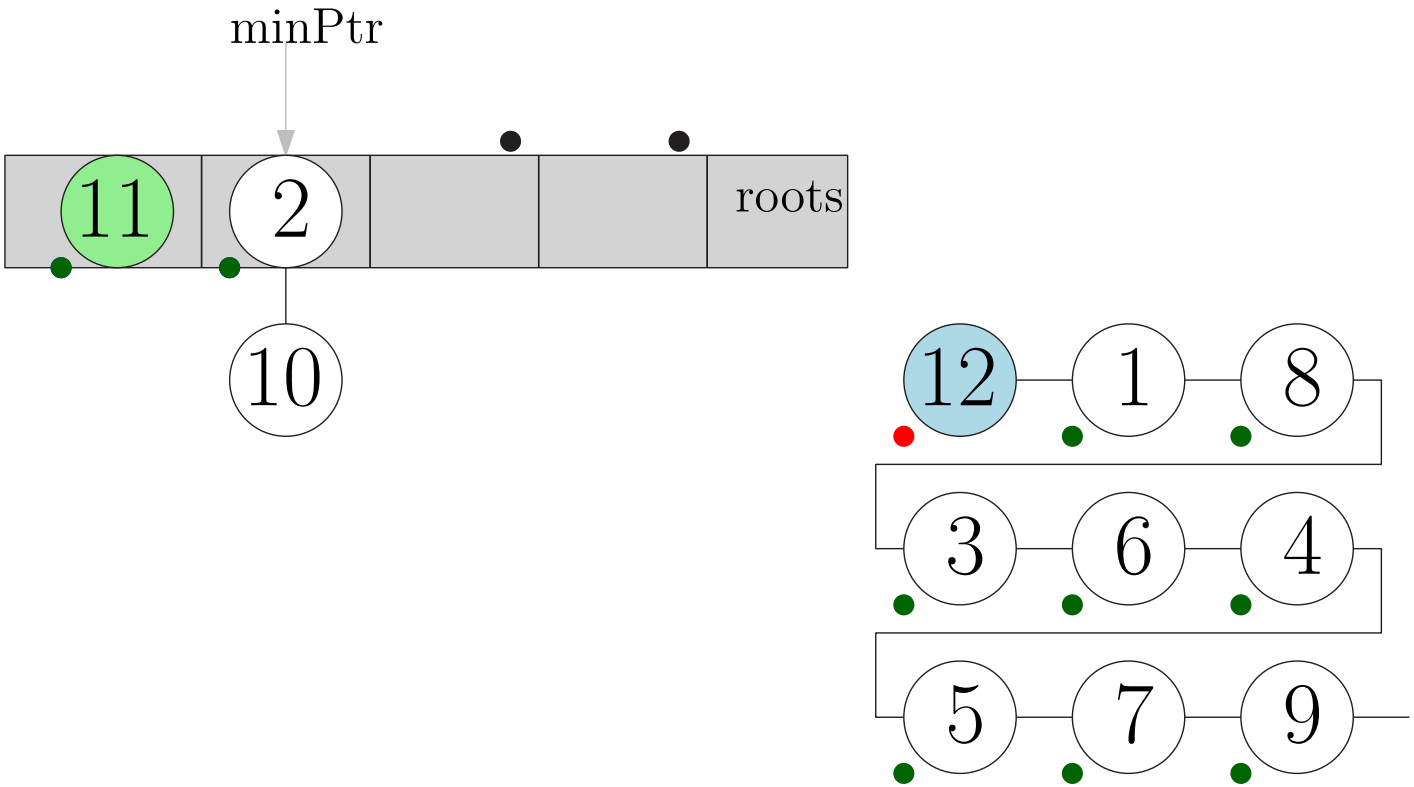
Fibonacci Heaps - Delete Min



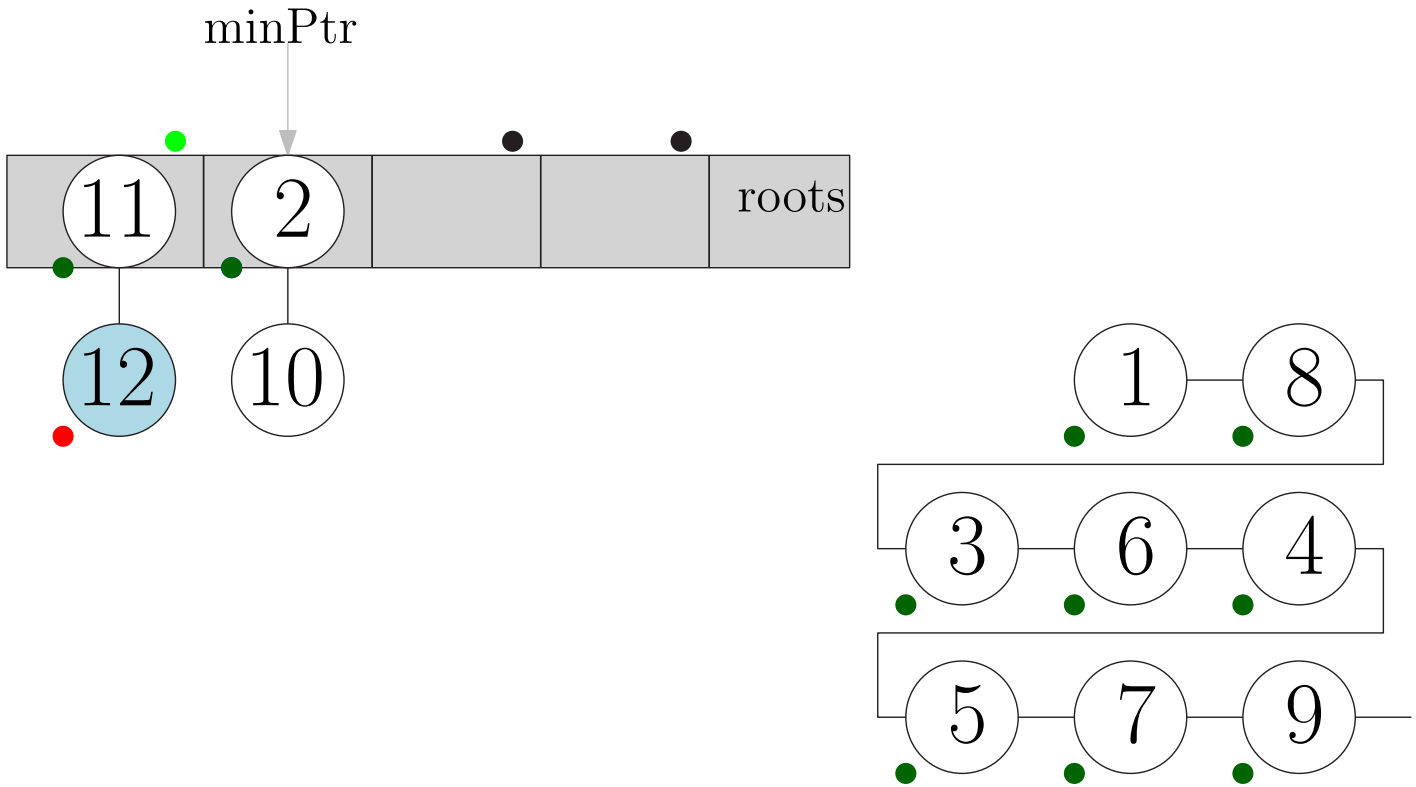
Fibonacci Heaps - Delete Min



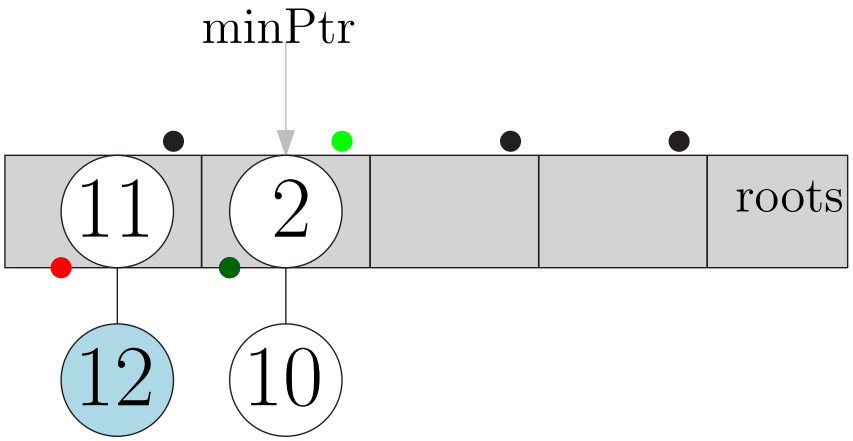
Fibonacci Heaps - Delete Min

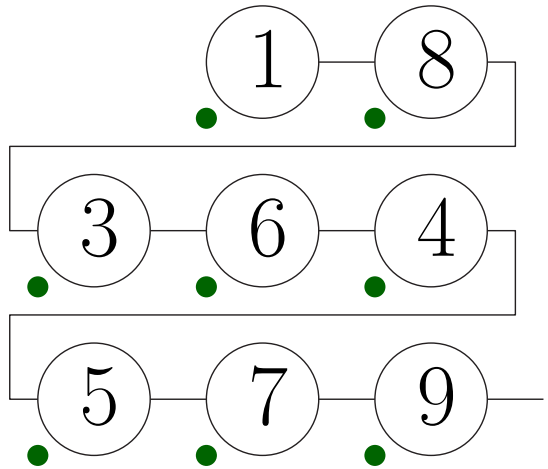


Fibonacci Heaps - Delete Min

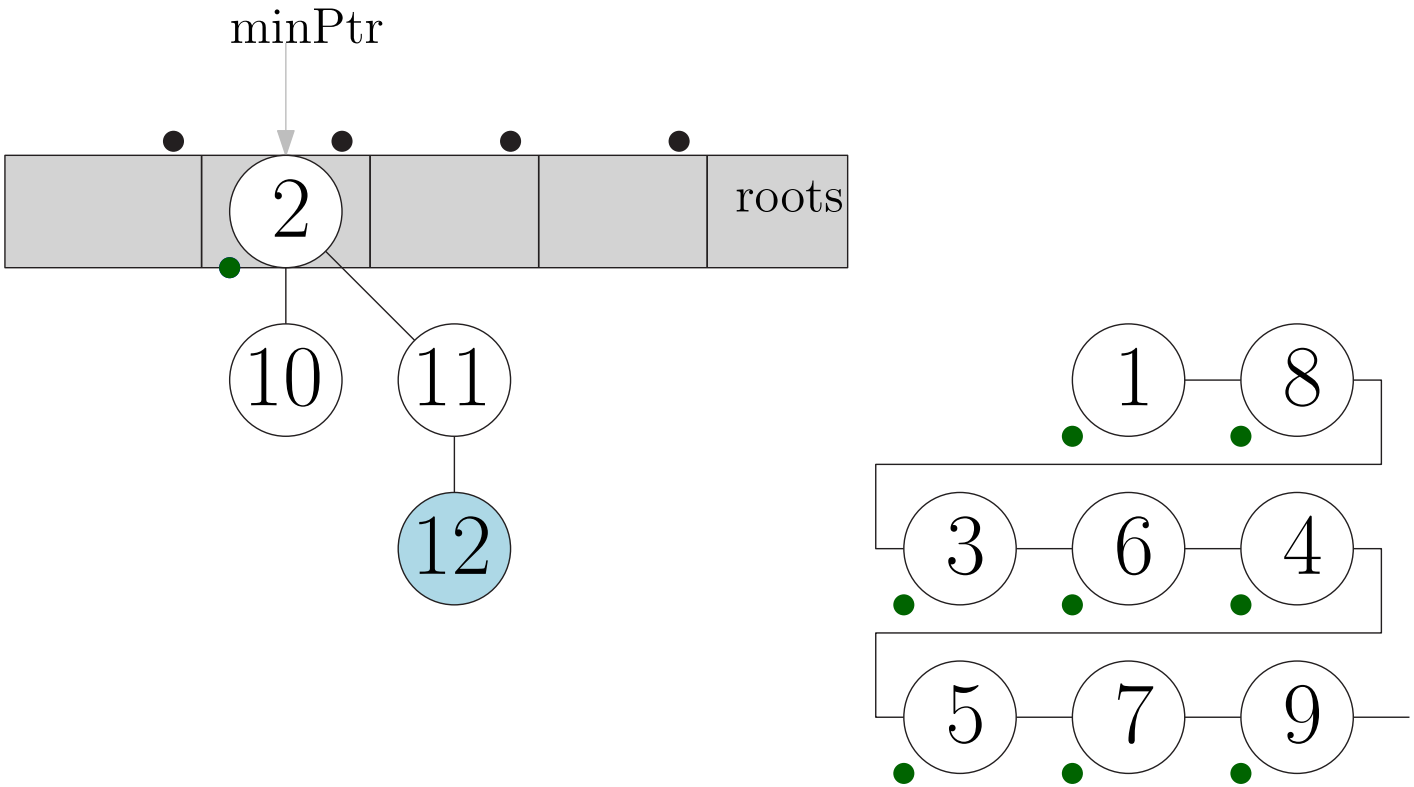


Fibonacci Heaps - Delete Min

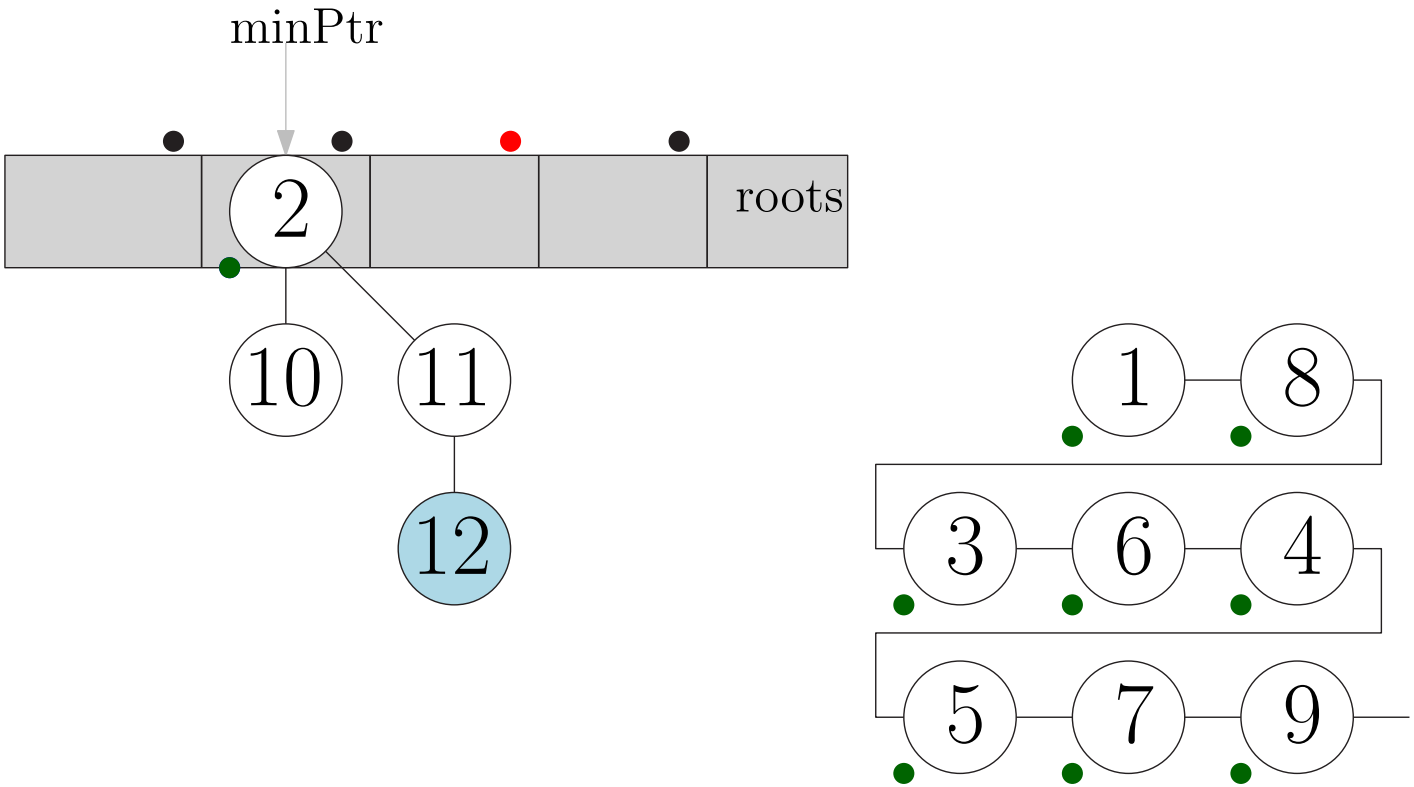




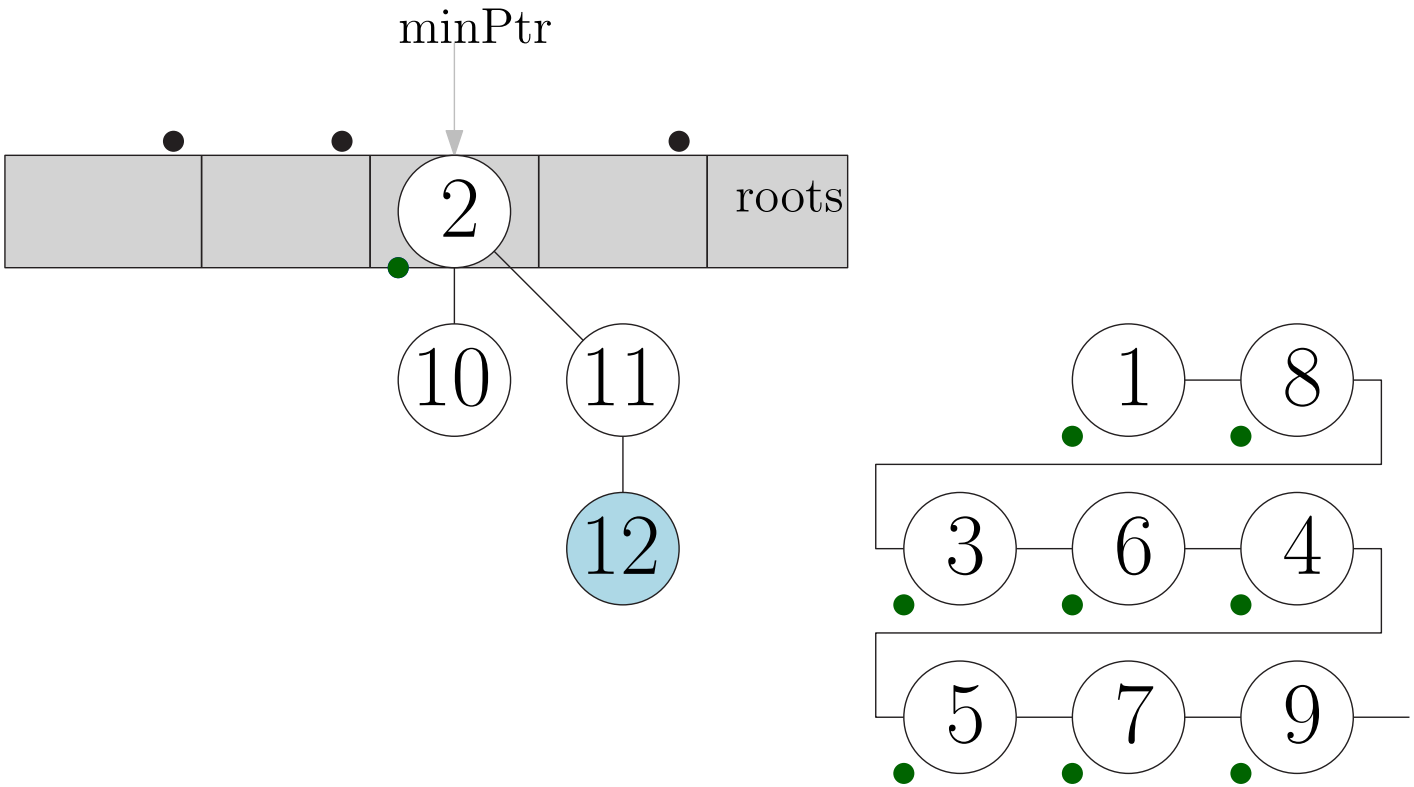
Fibonacci Heaps - Delete Min



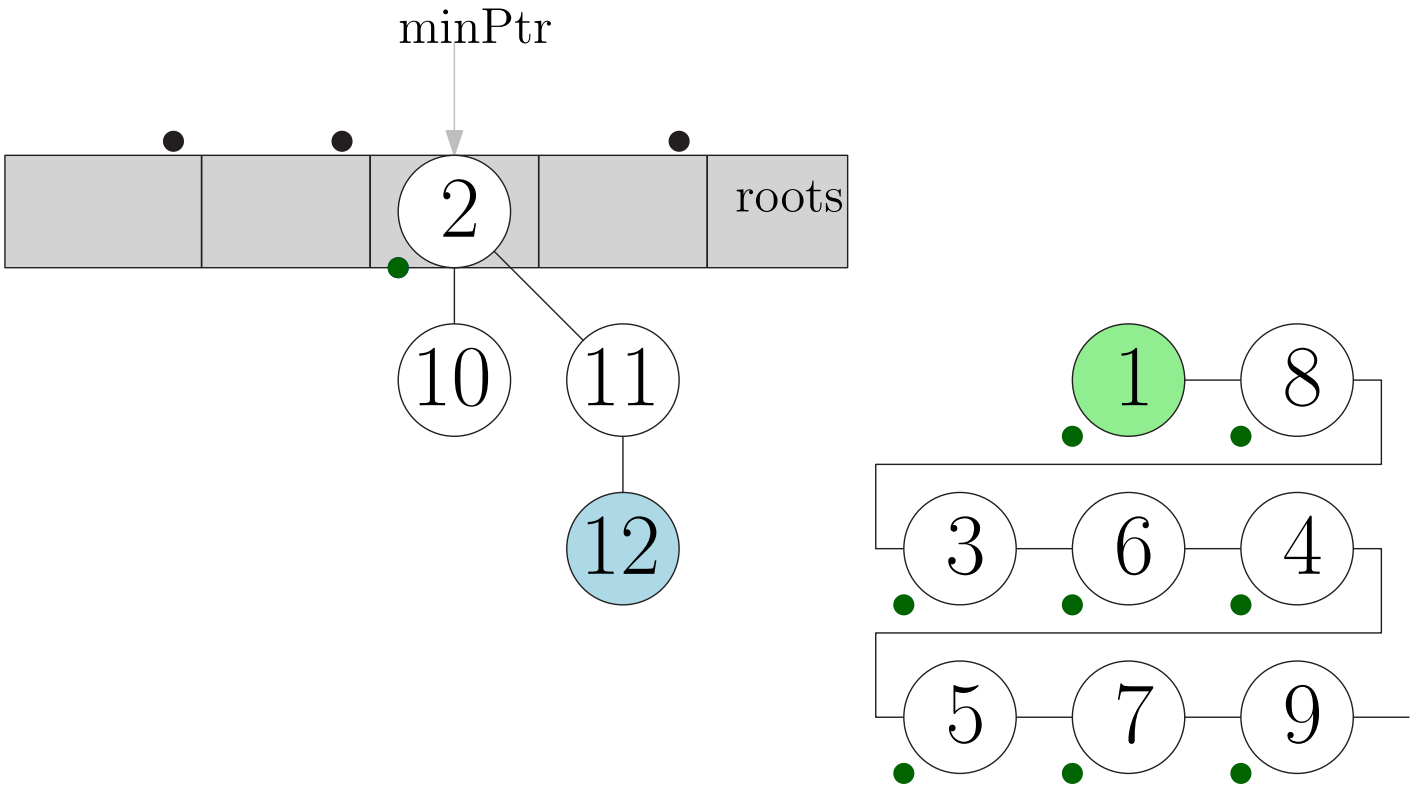
Fibonacci Heaps - Delete Min



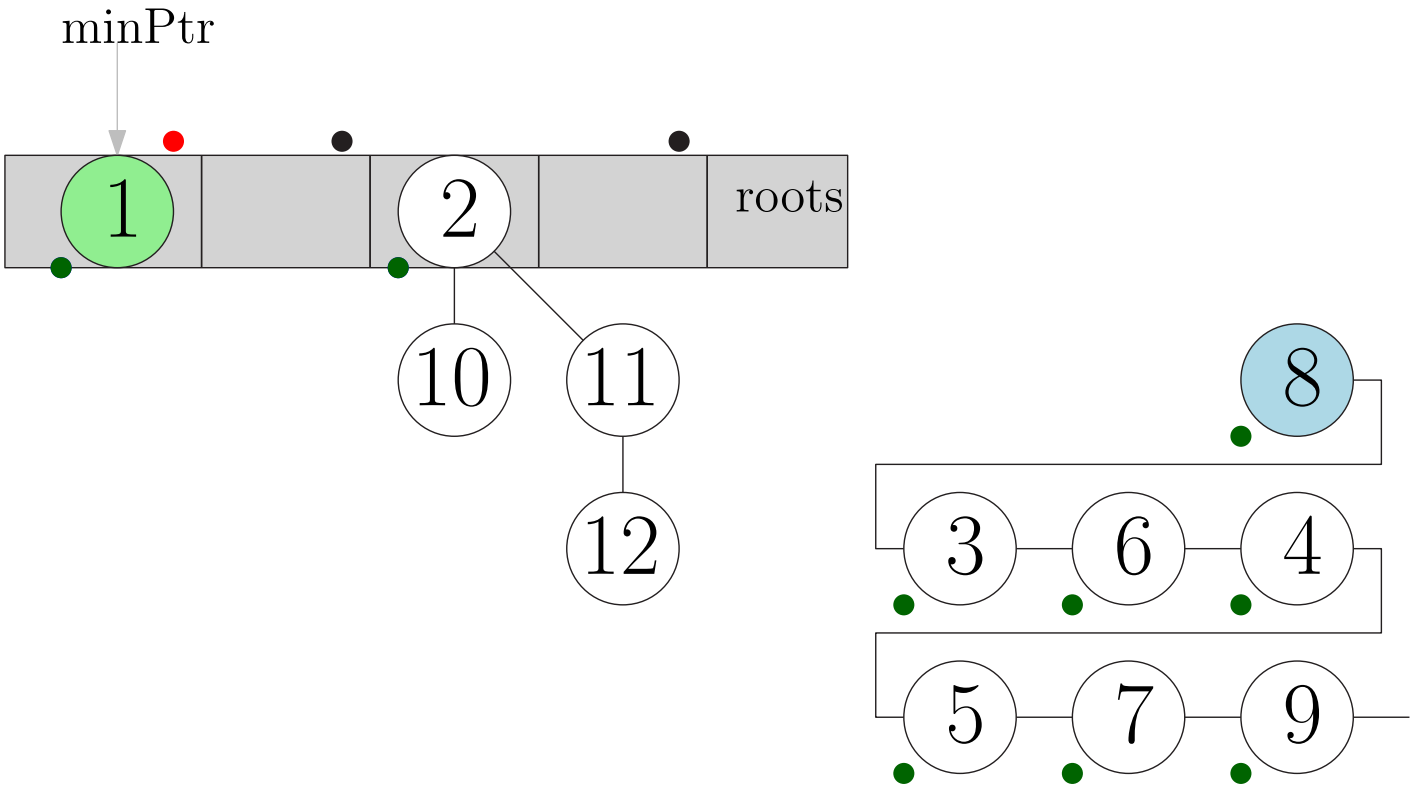
Fibonacci Heaps - Delete Min



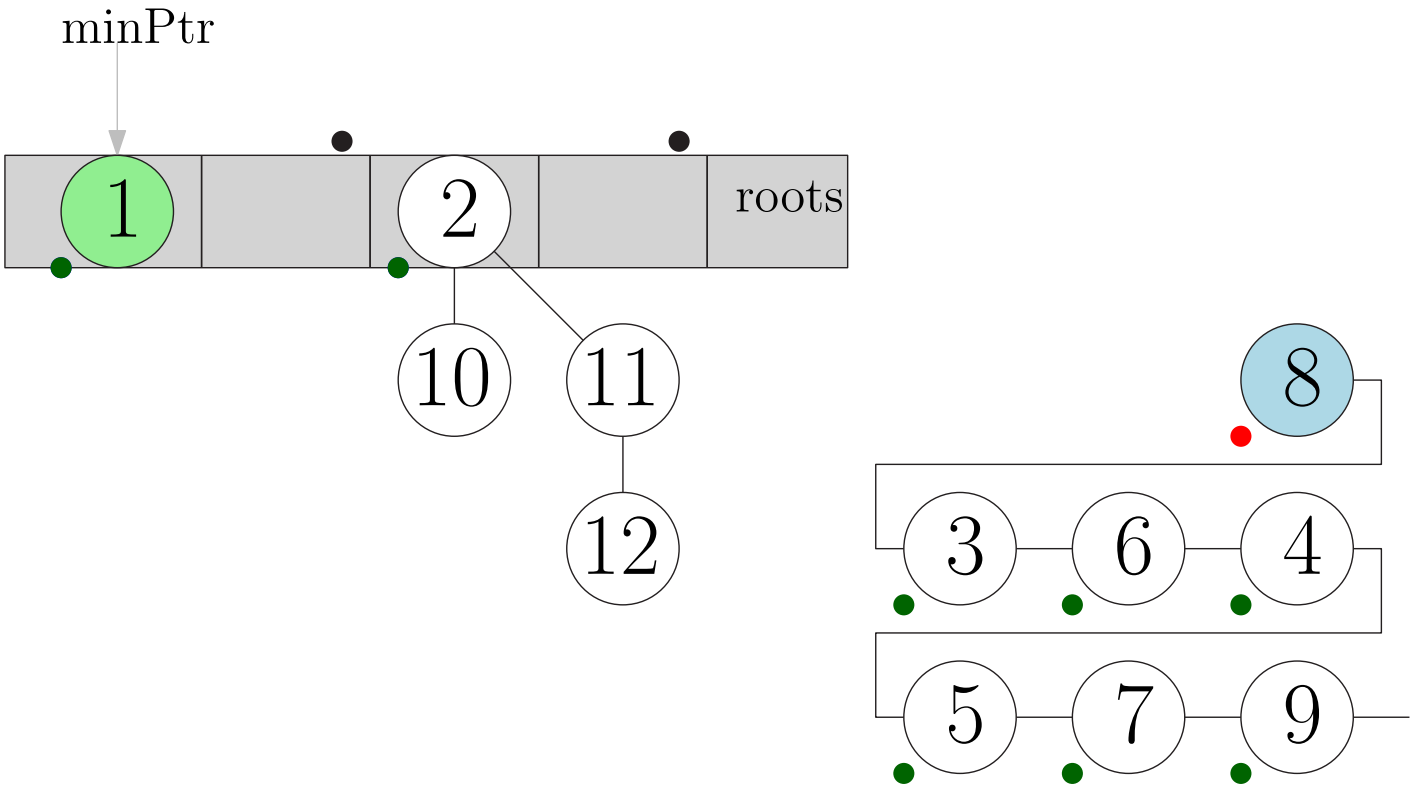
Fibonacci Heaps - Delete Min



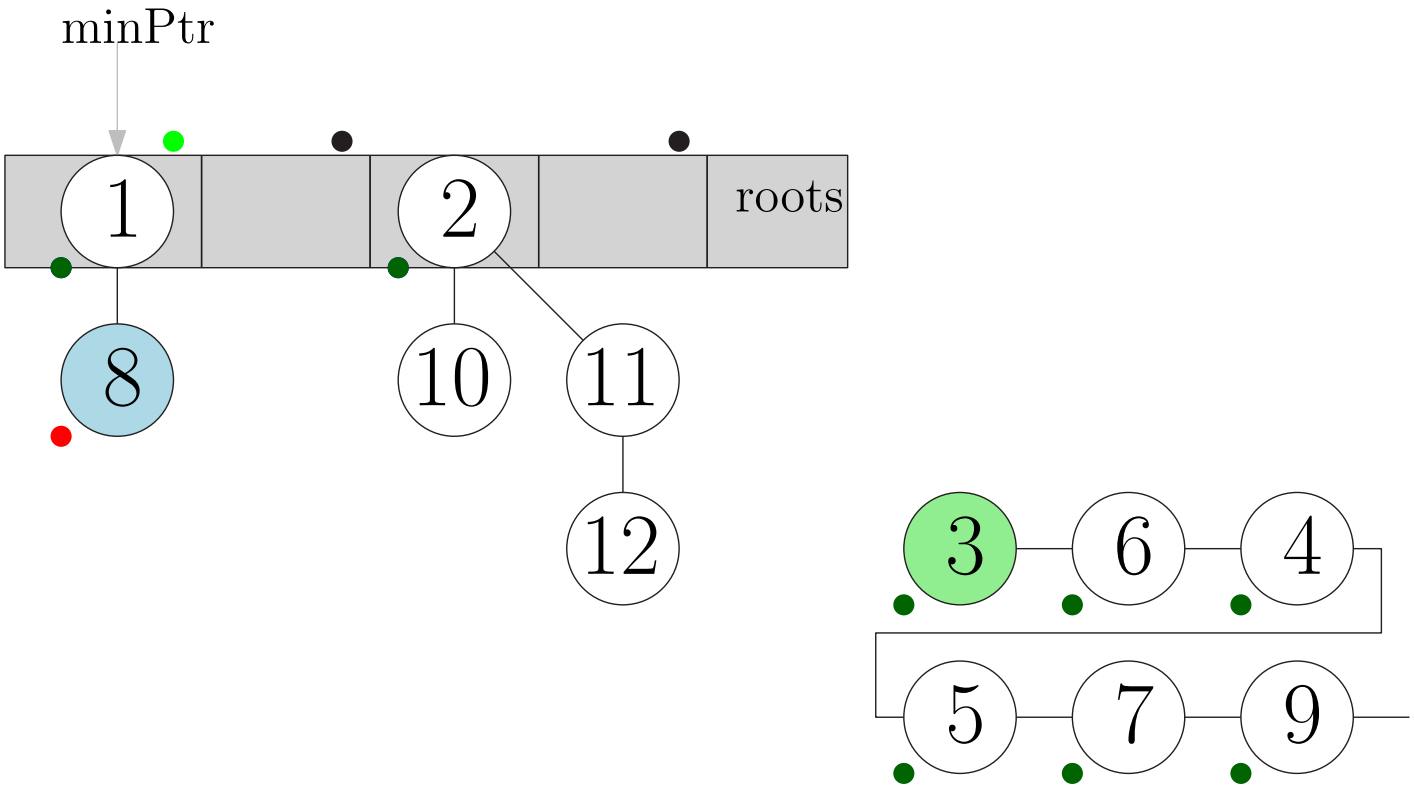
Fibonacci Heaps - Delete Min



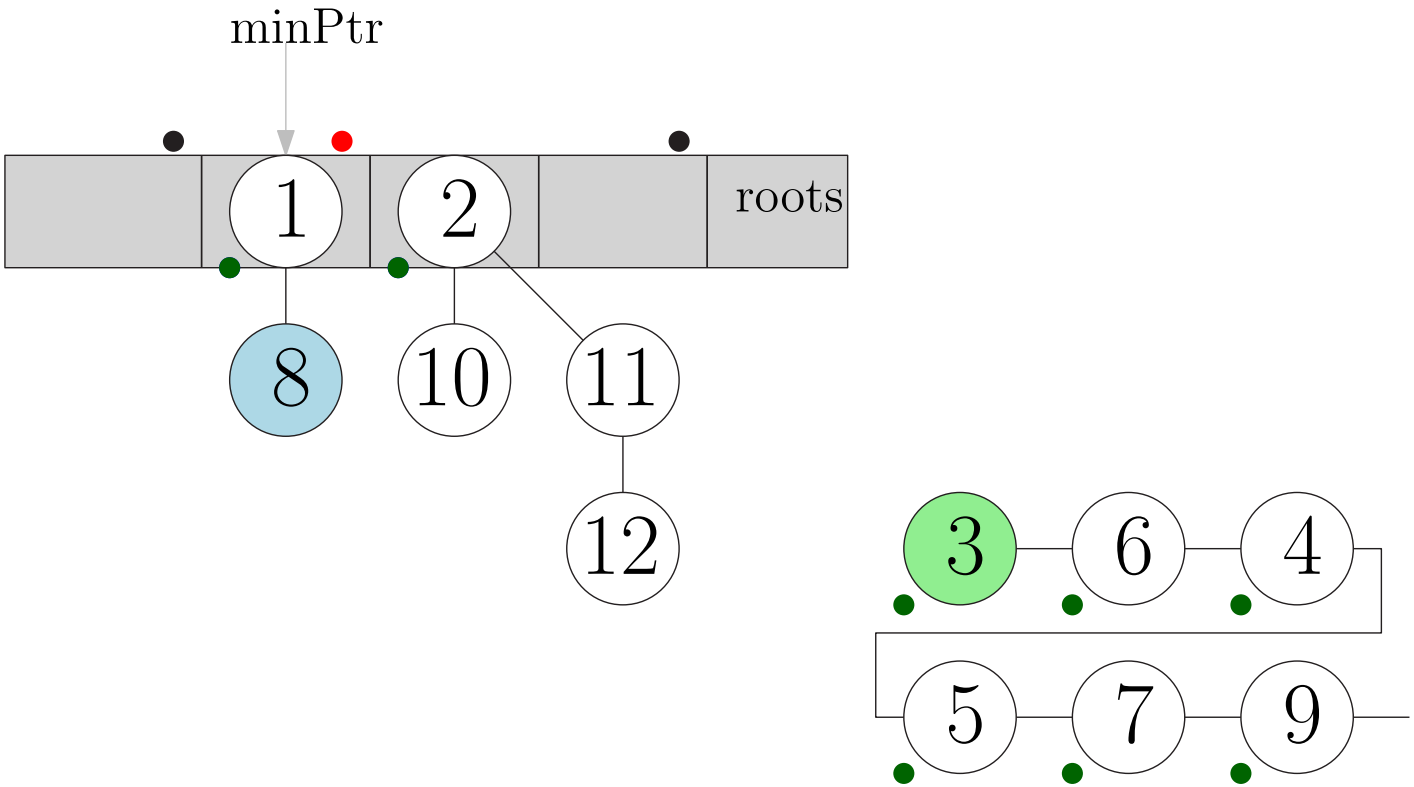
Fibonacci Heaps - Delete Min



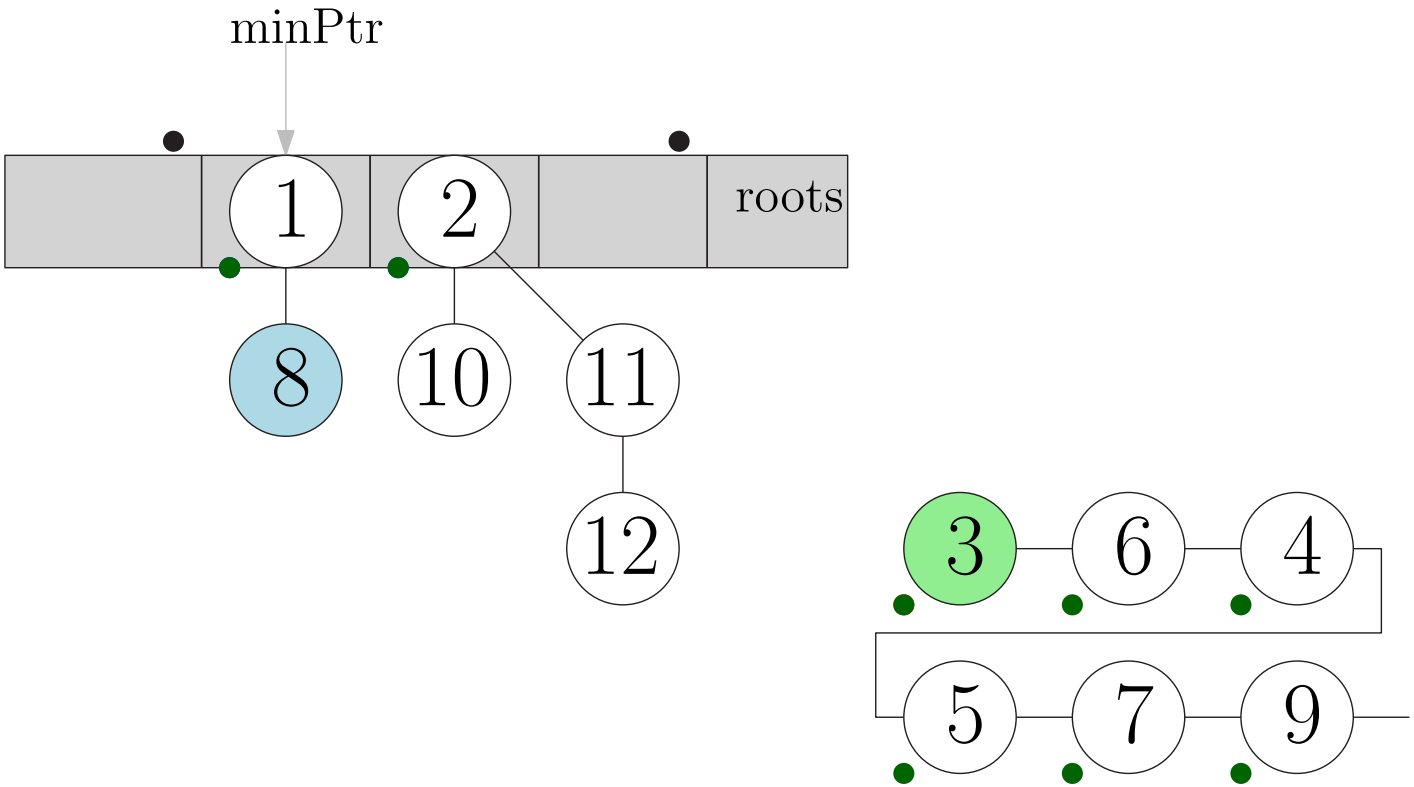
Fibonacci Heaps - Delete Min



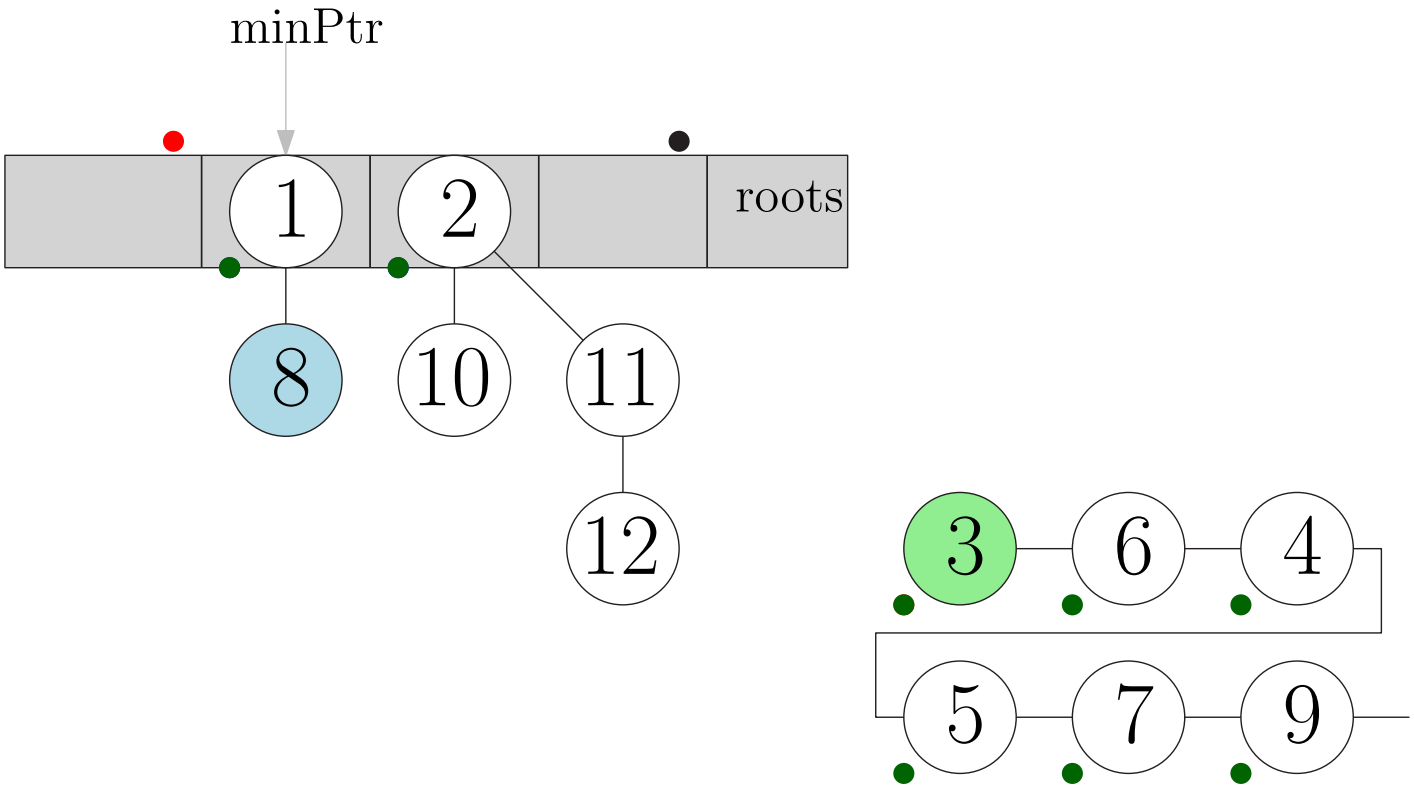
Fibonacci Heaps - Delete Min



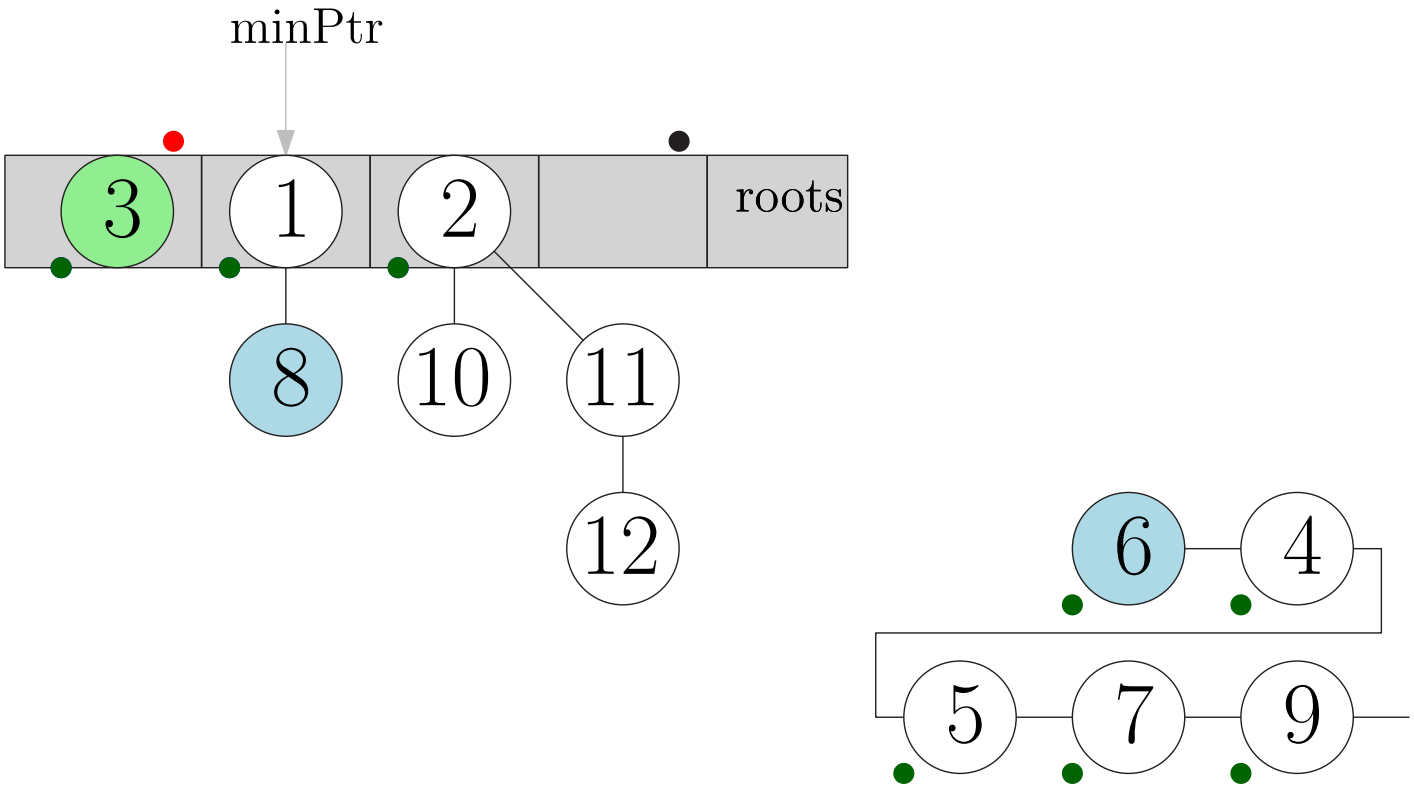
Fibonacci Heaps - Delete Min



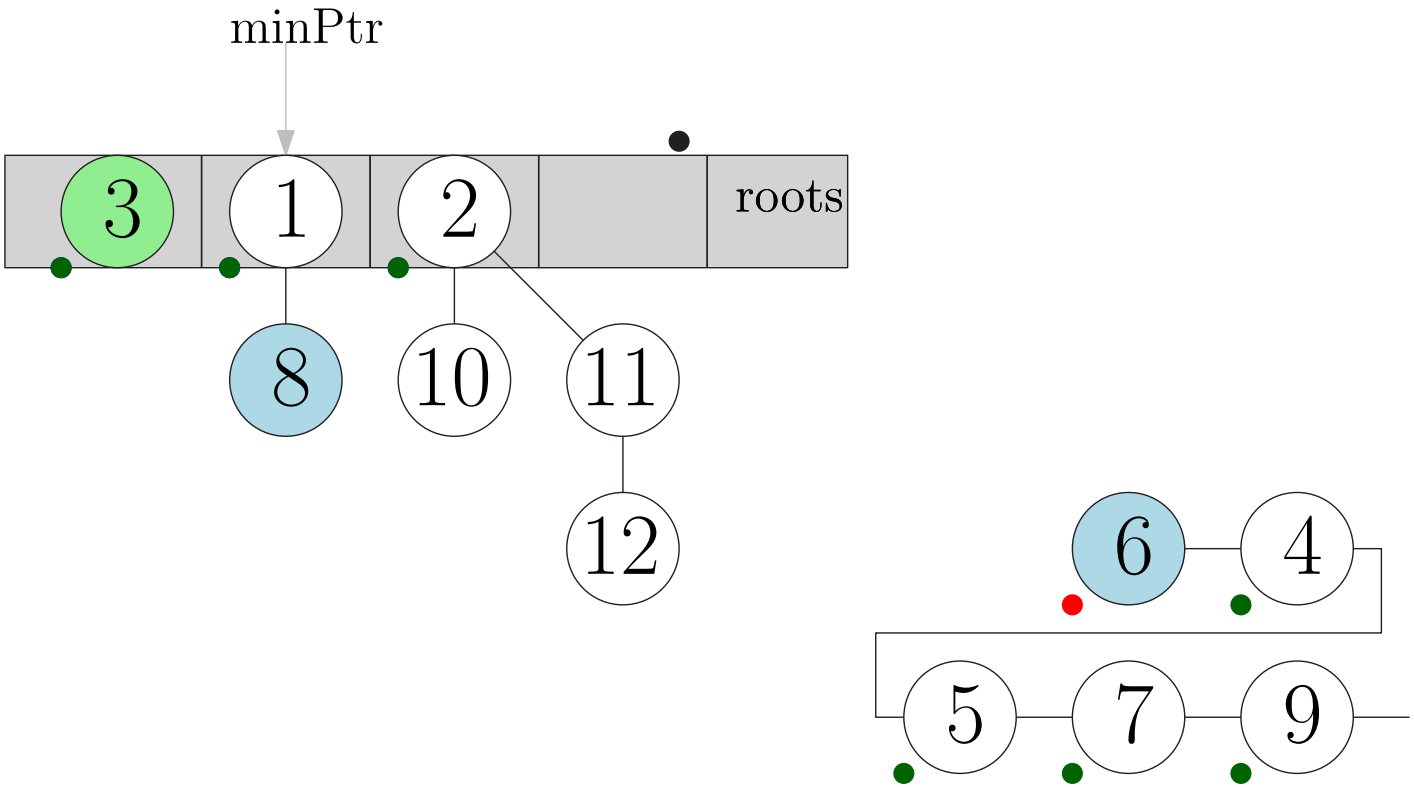
Fibonacci Heaps - Delete Min



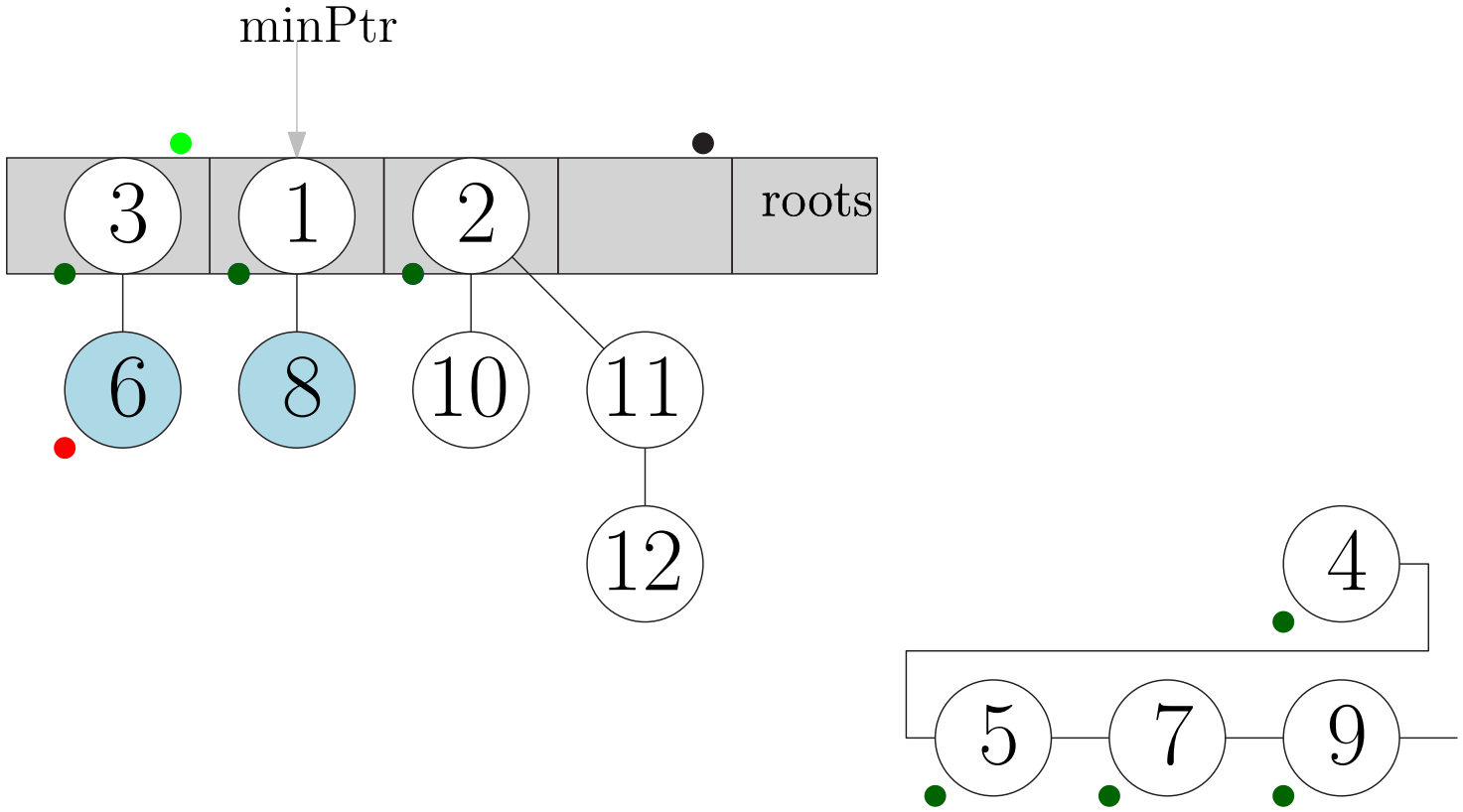
Fibonacci Heaps - Delete Min



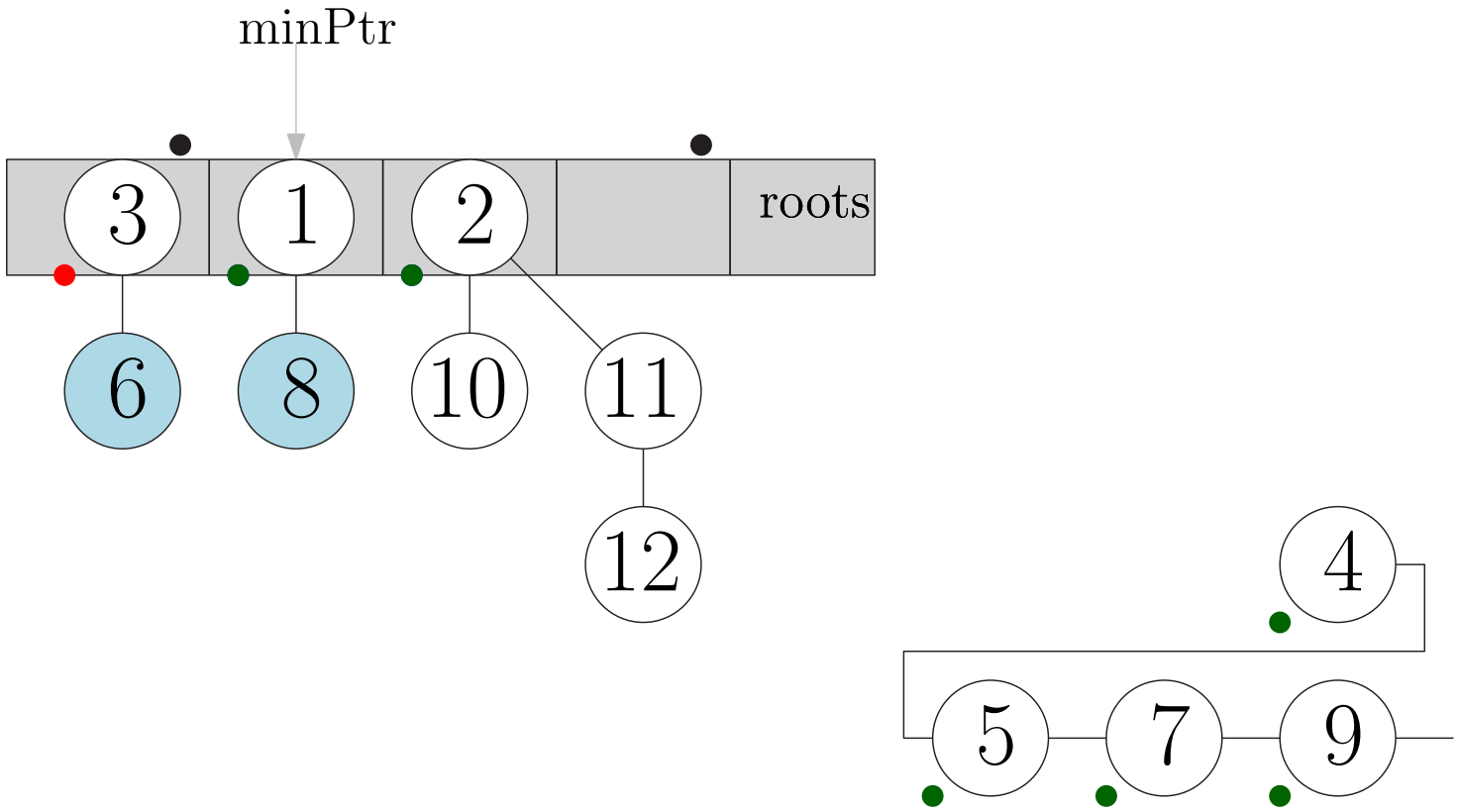
Fibonacci Heaps - Delete Min



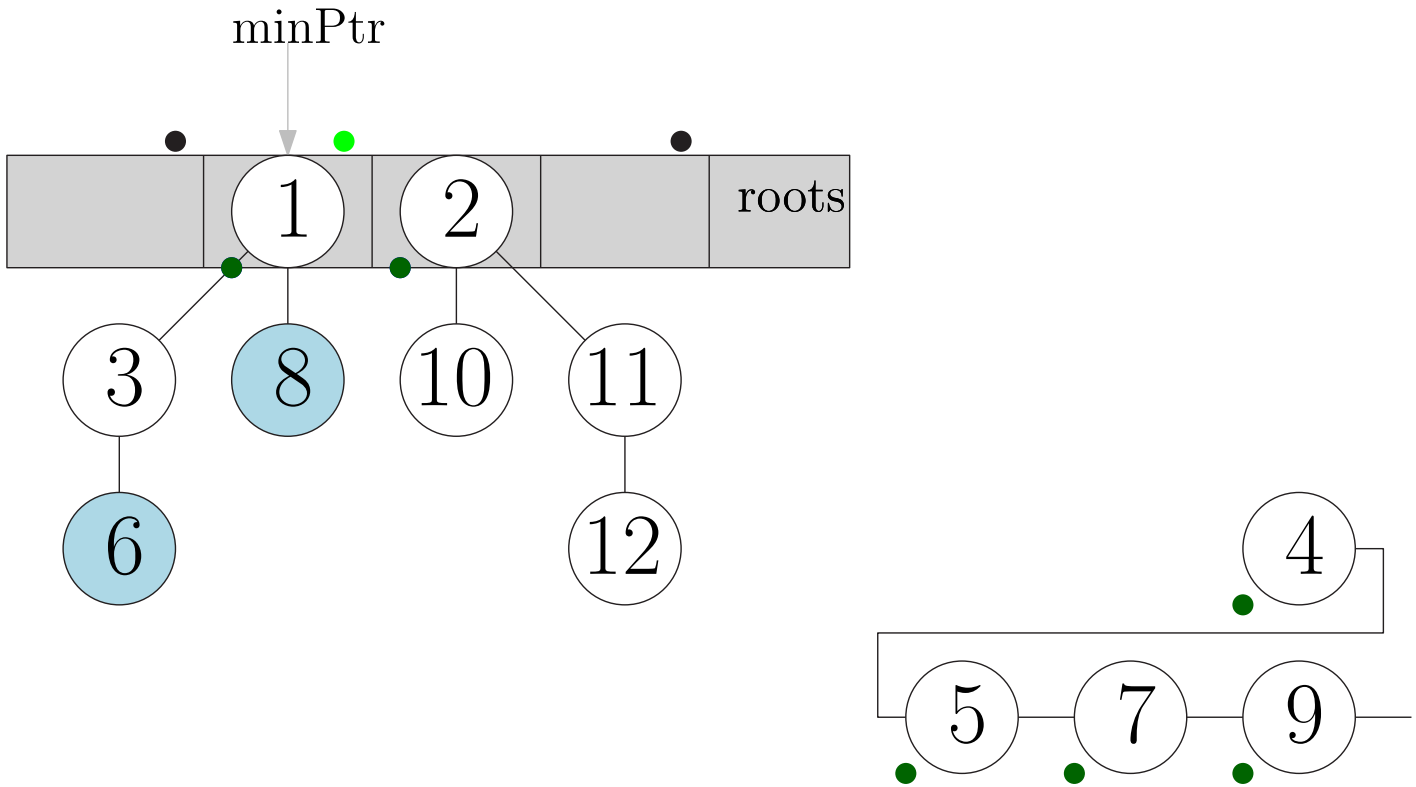
Fibonacci Heaps - Delete Min



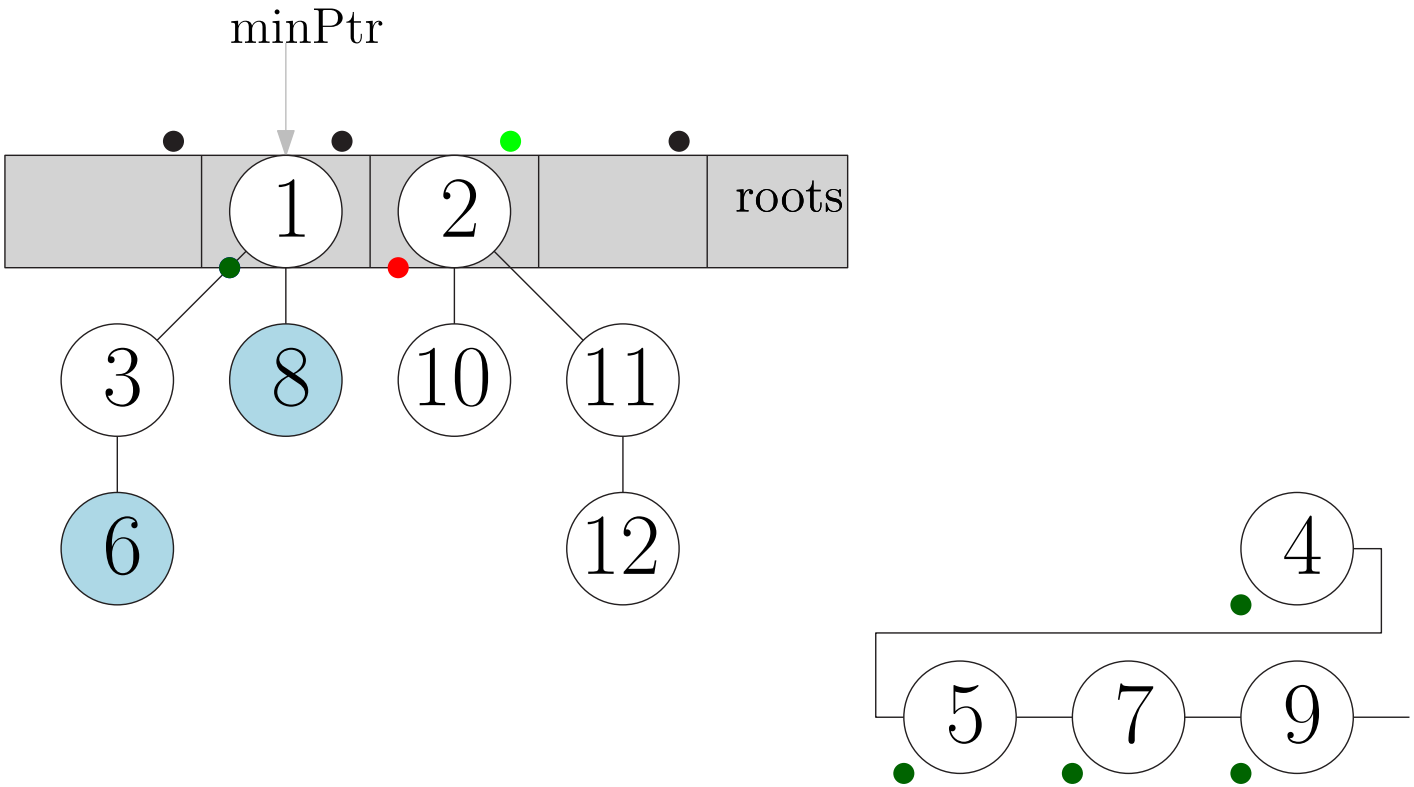
Fibonacci Heaps - Delete Min



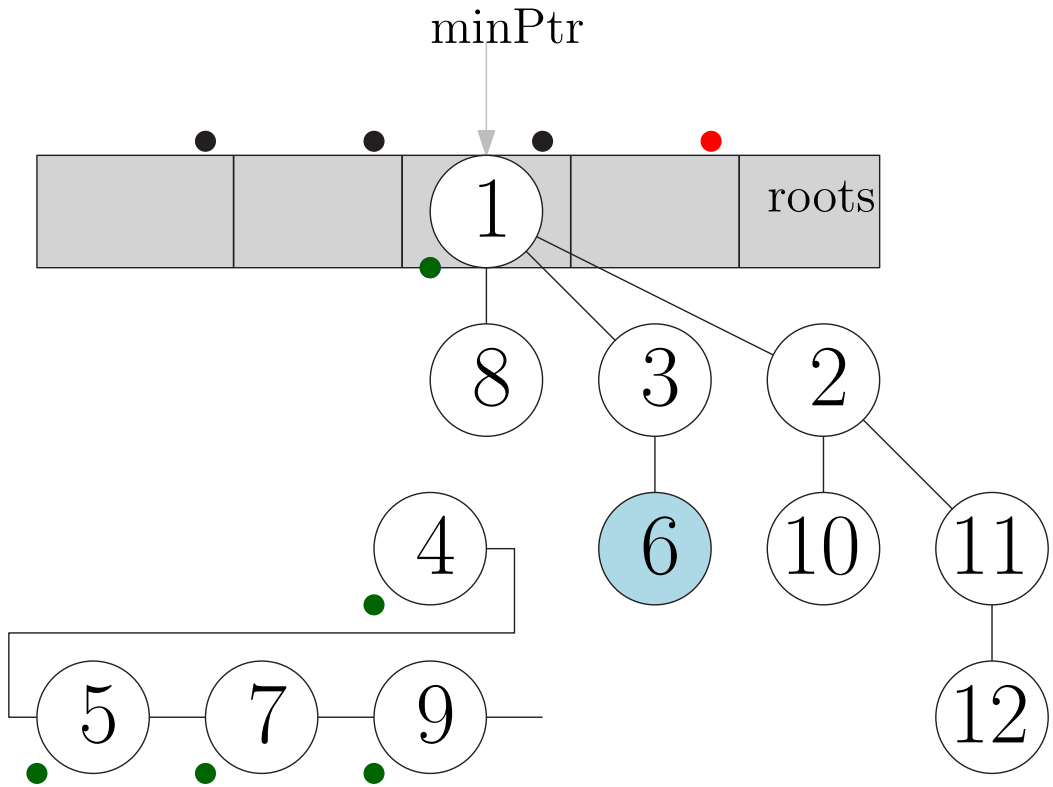
Fibonacci Heaps - Delete Min



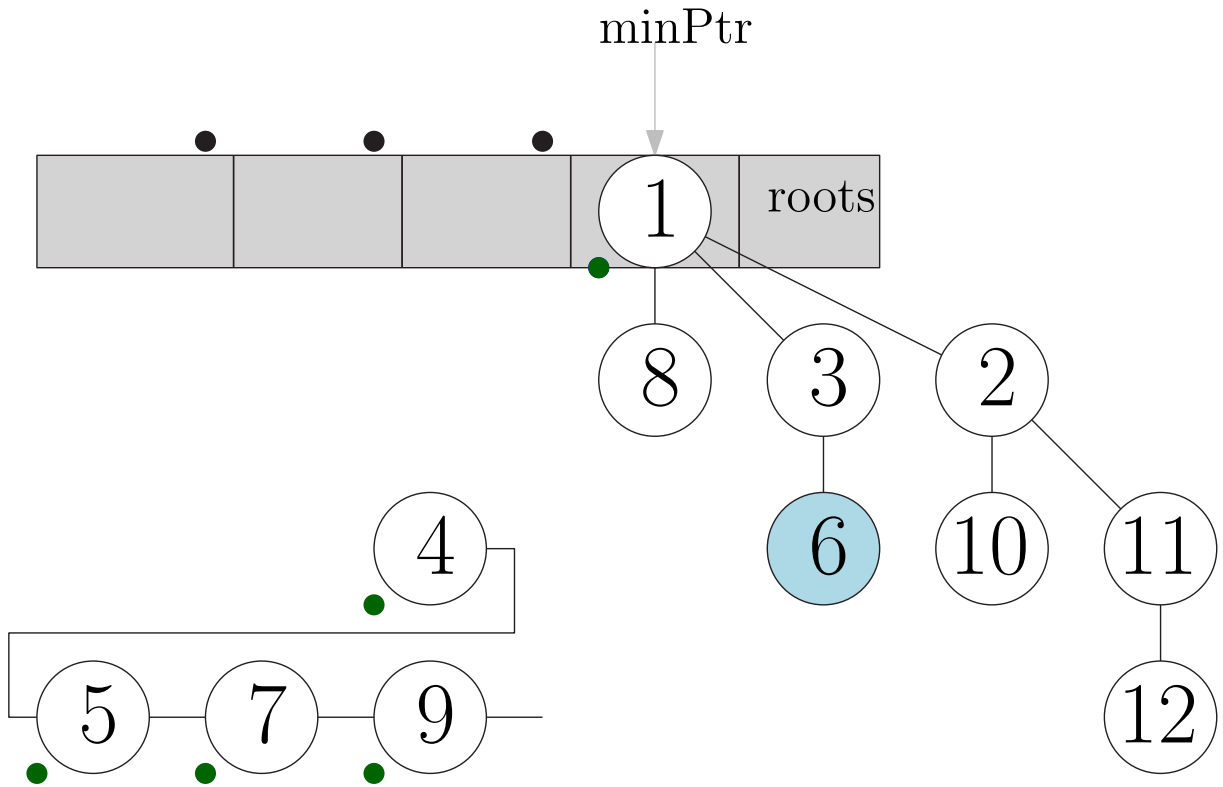
Fibonacci Heaps - Delete Min



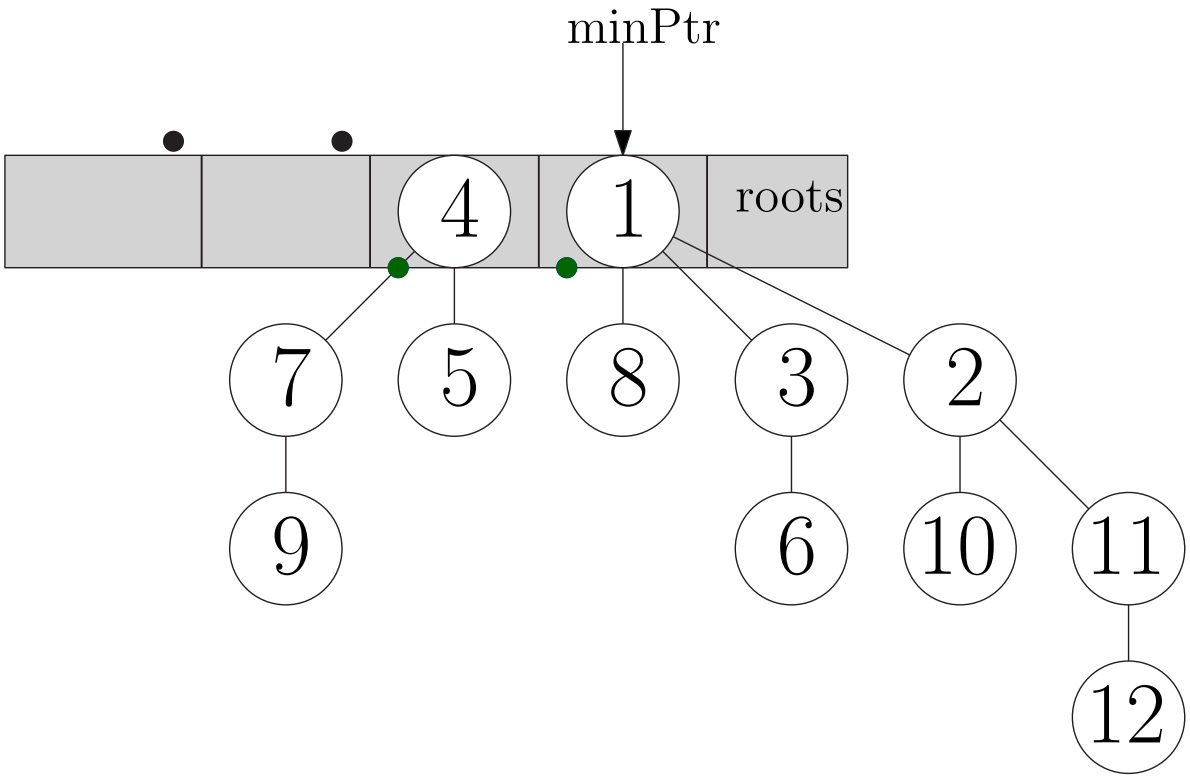
Fibonacci Heaps - Delete Min



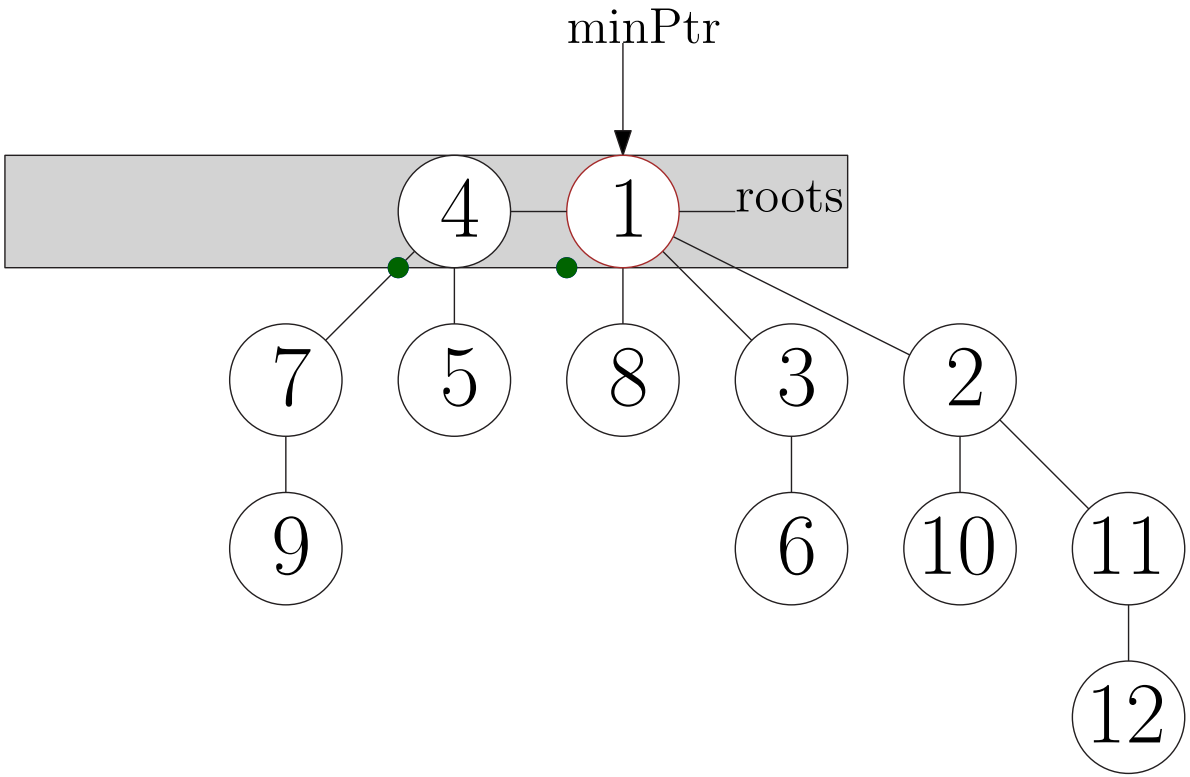
Fibonacci Heaps - Delete Min



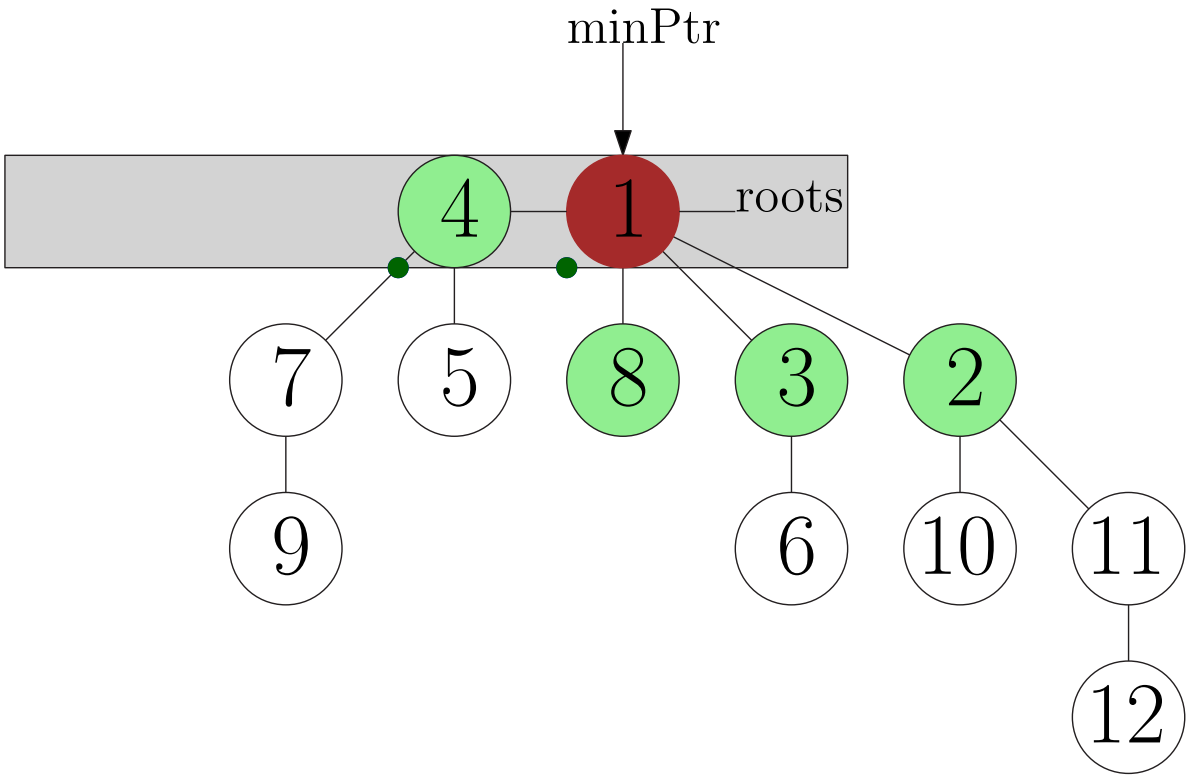
Fibonacci Heaps - Delete Min



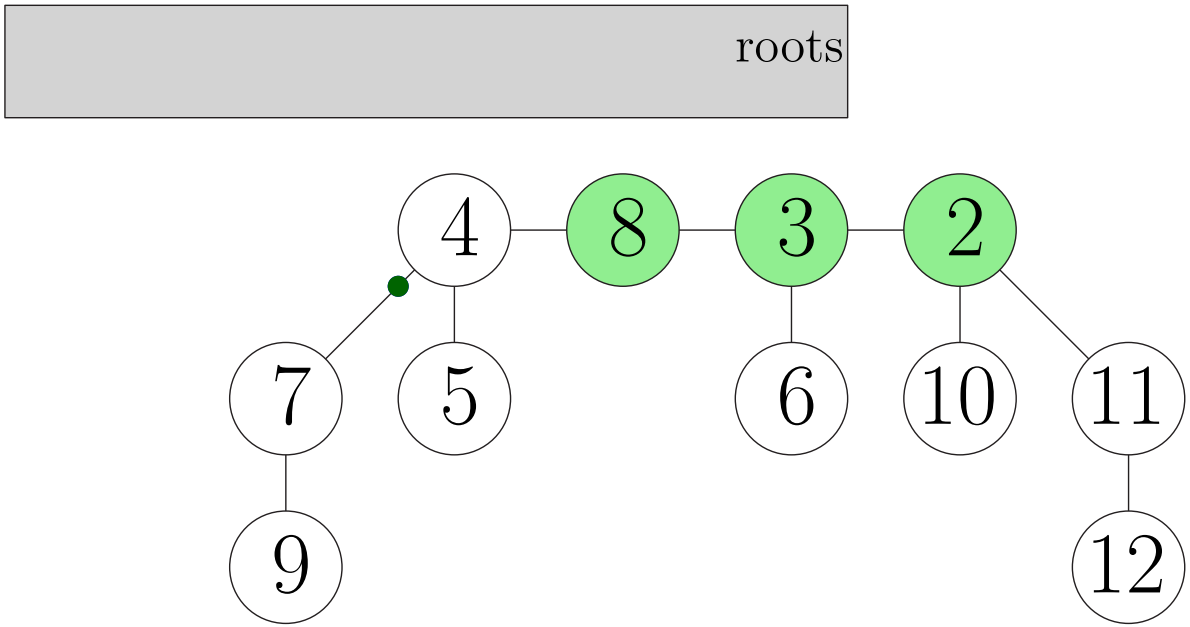
Fibonacci Heaps - Delete Min



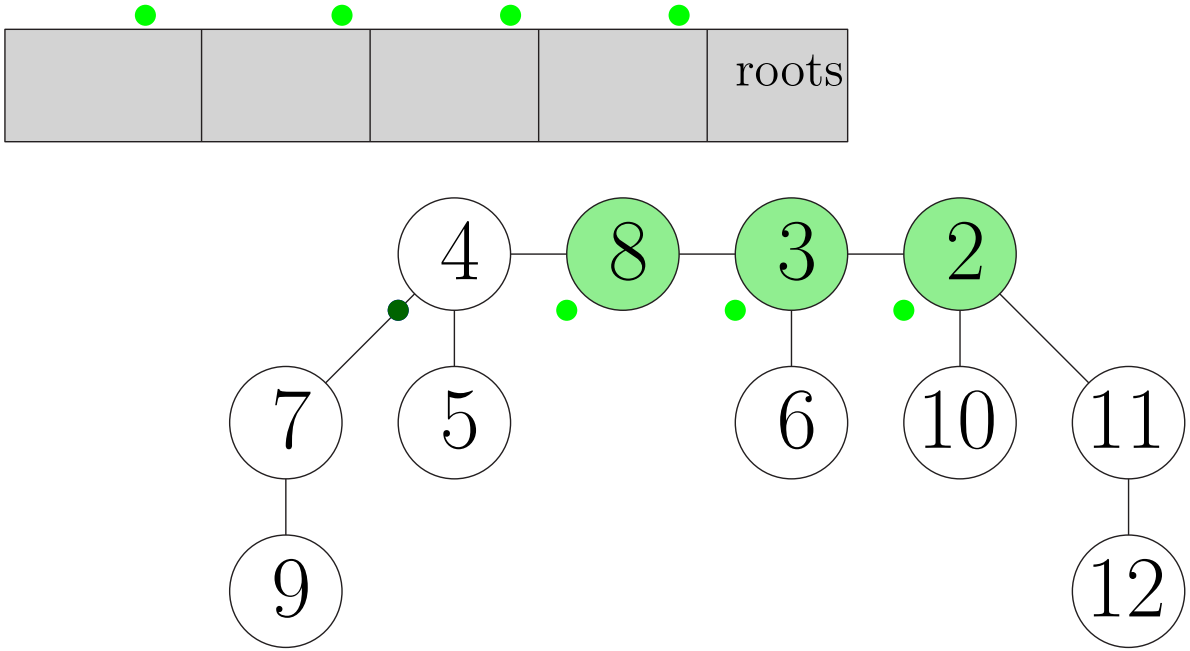
Fibonacci Heaps - Delete Min



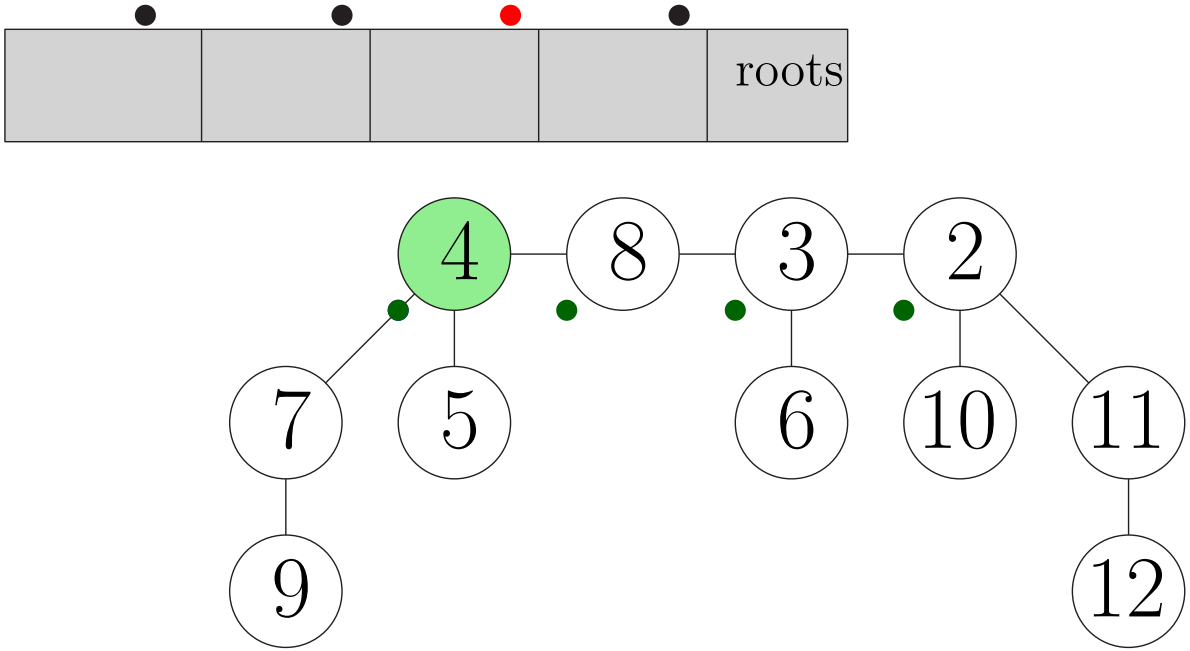
Fibonacci Heaps - Delete Min



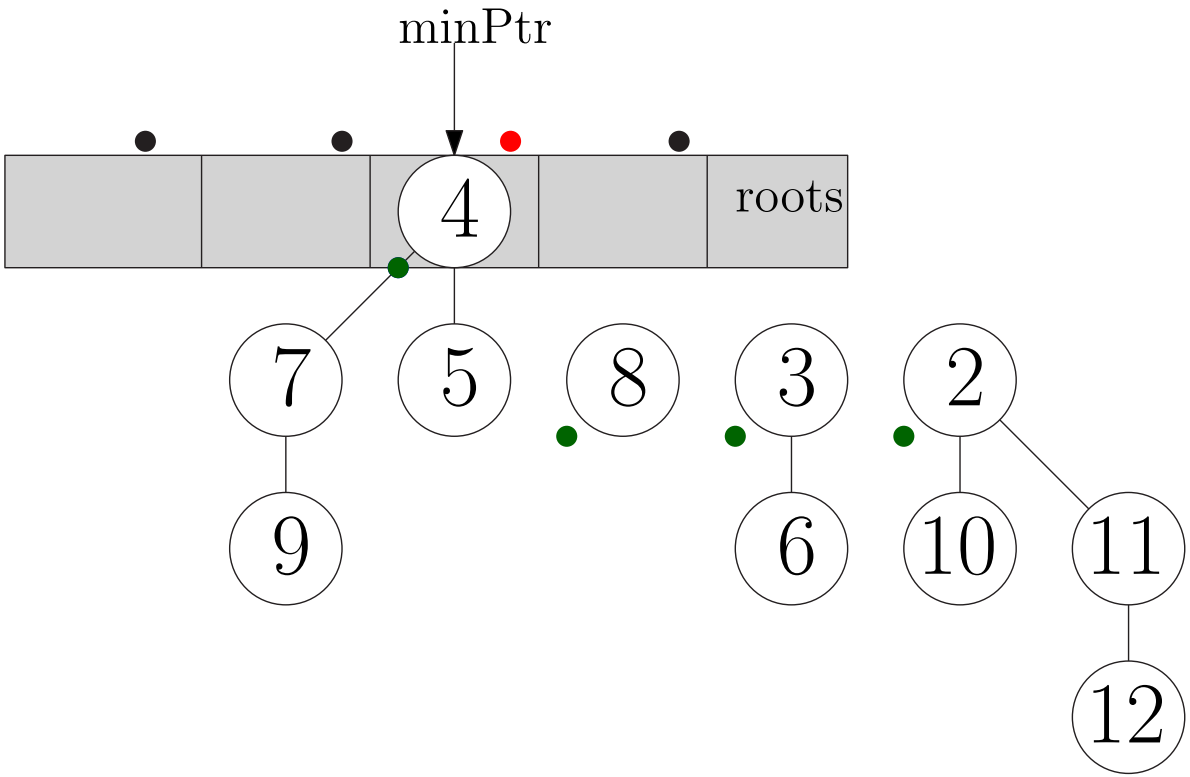
Fibonacci Heaps - Delete Min



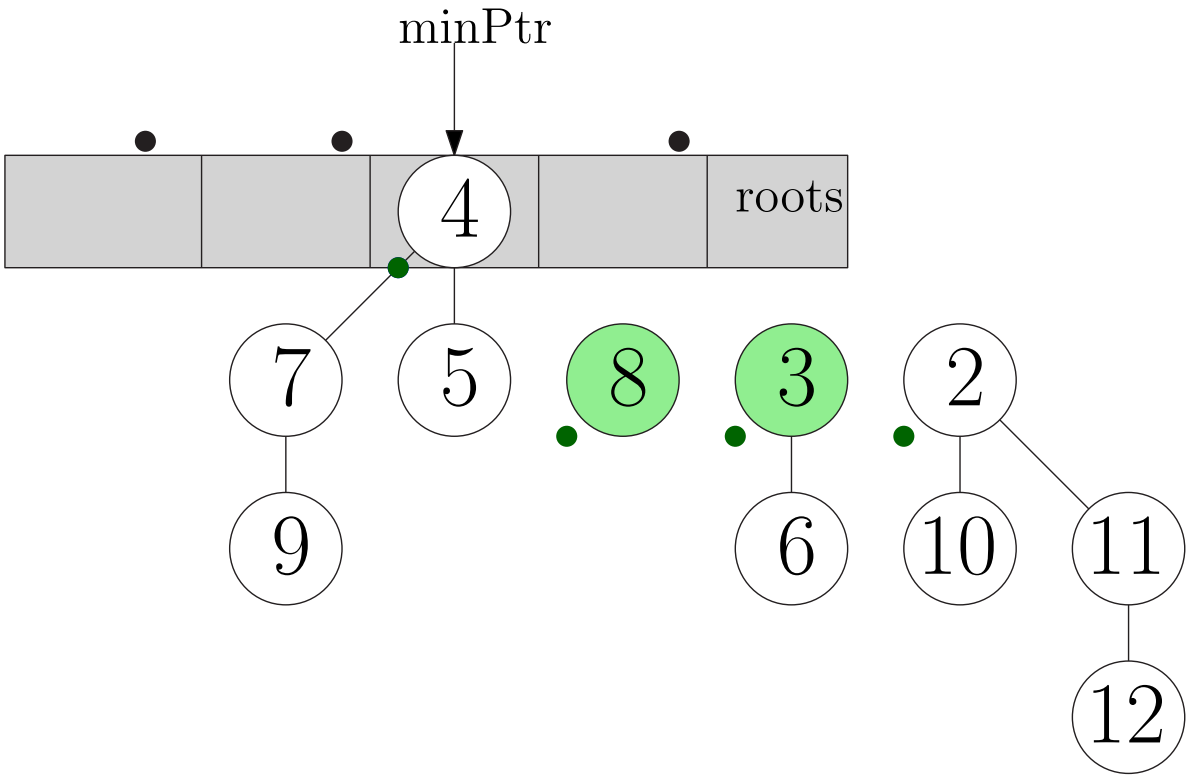
Fibonacci Heaps - Delete Min



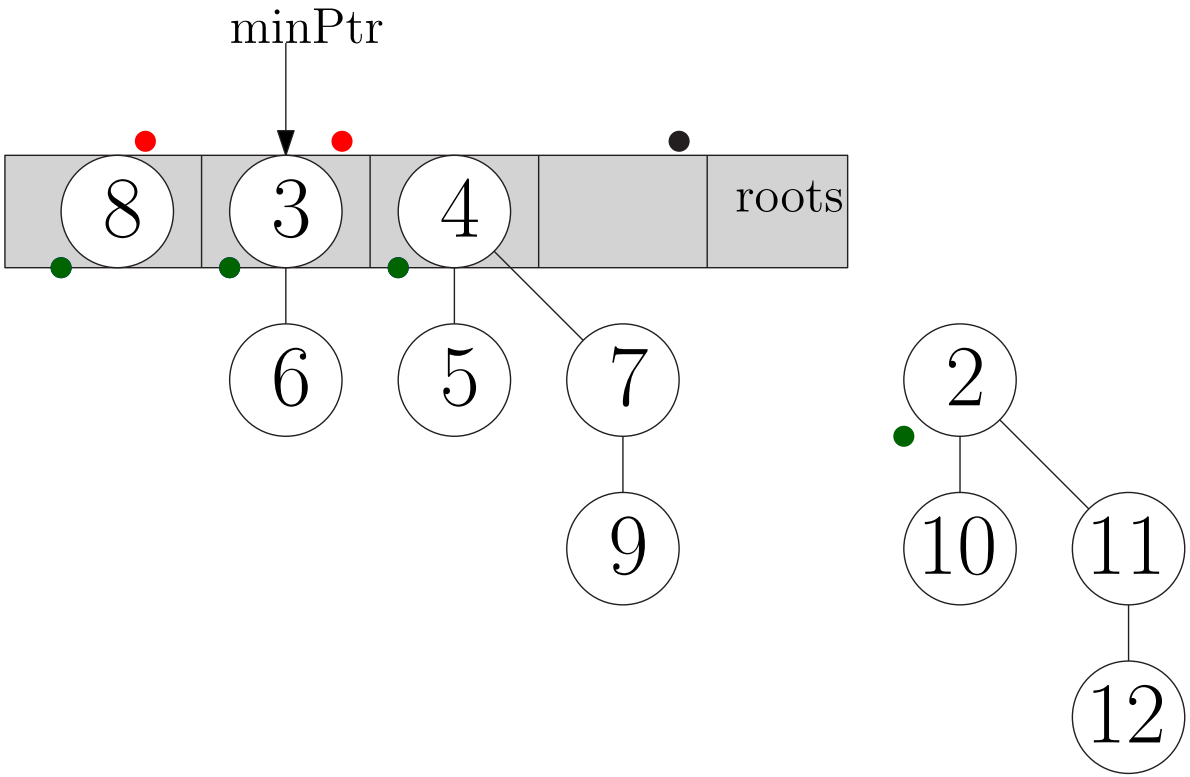
Fibonacci Heaps - Delete Min



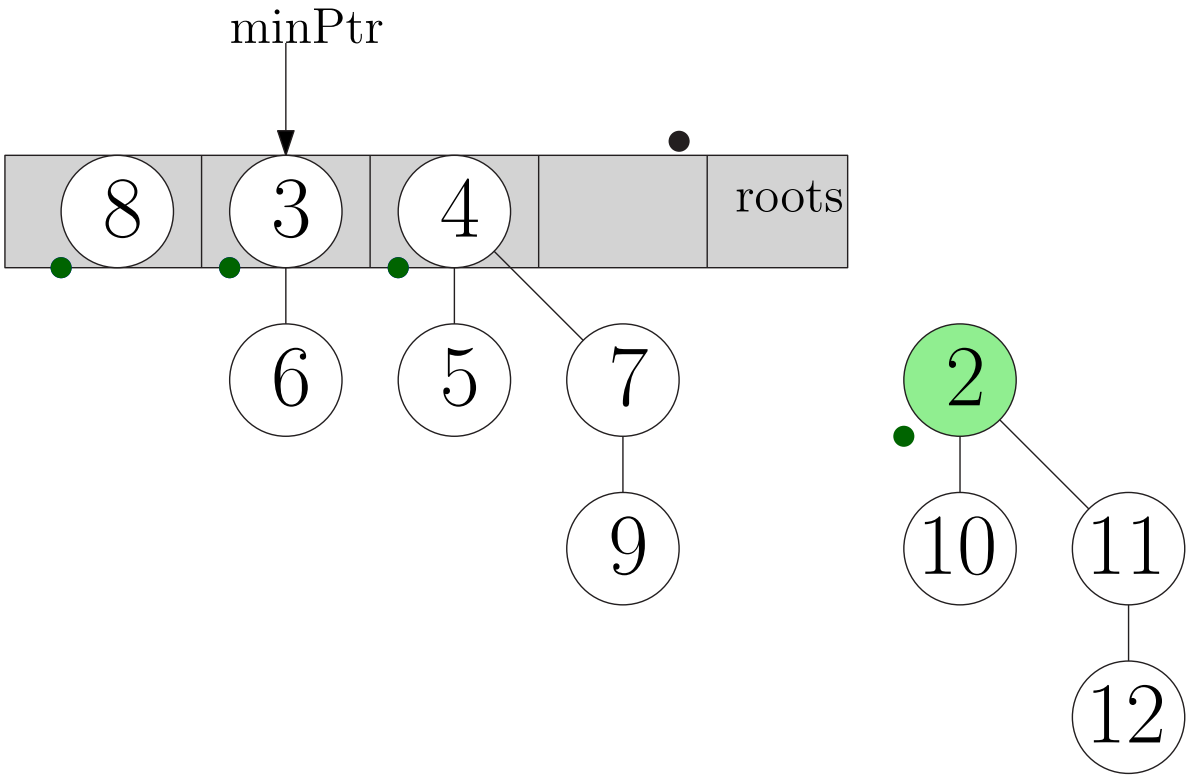
Fibonacci Heaps - Delete Min



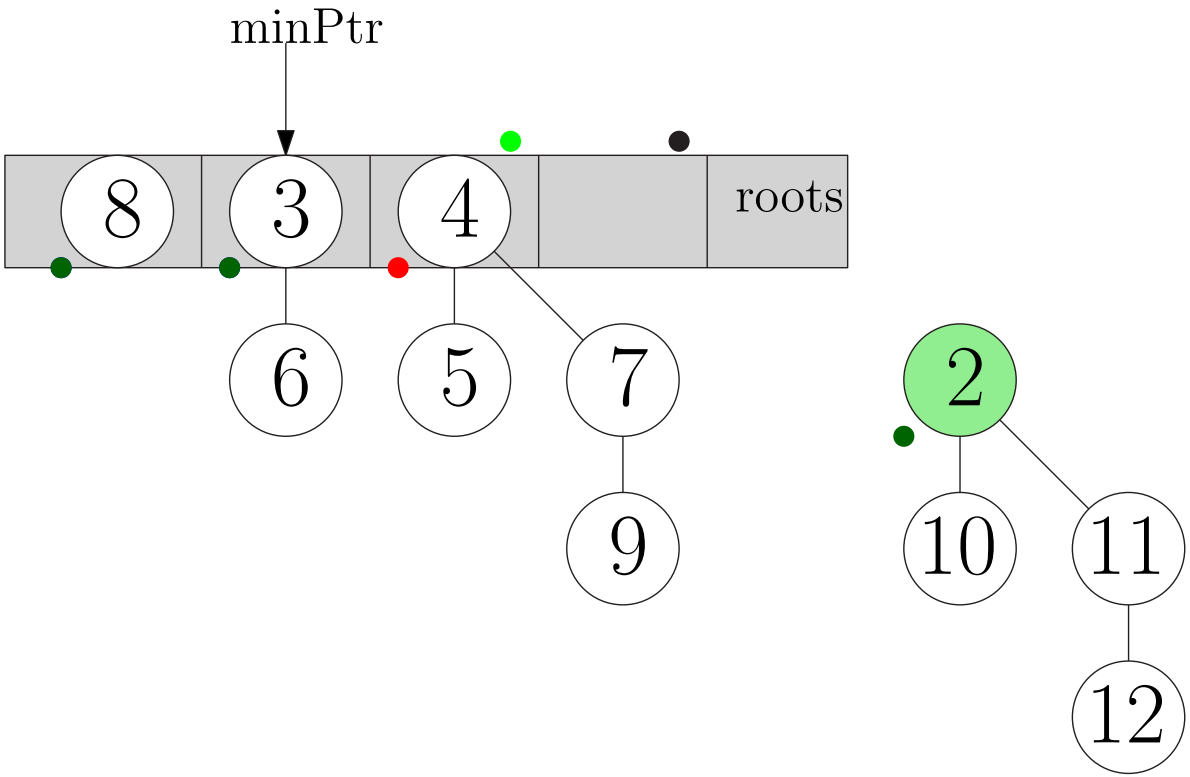
Fibonacci Heaps - Delete Min



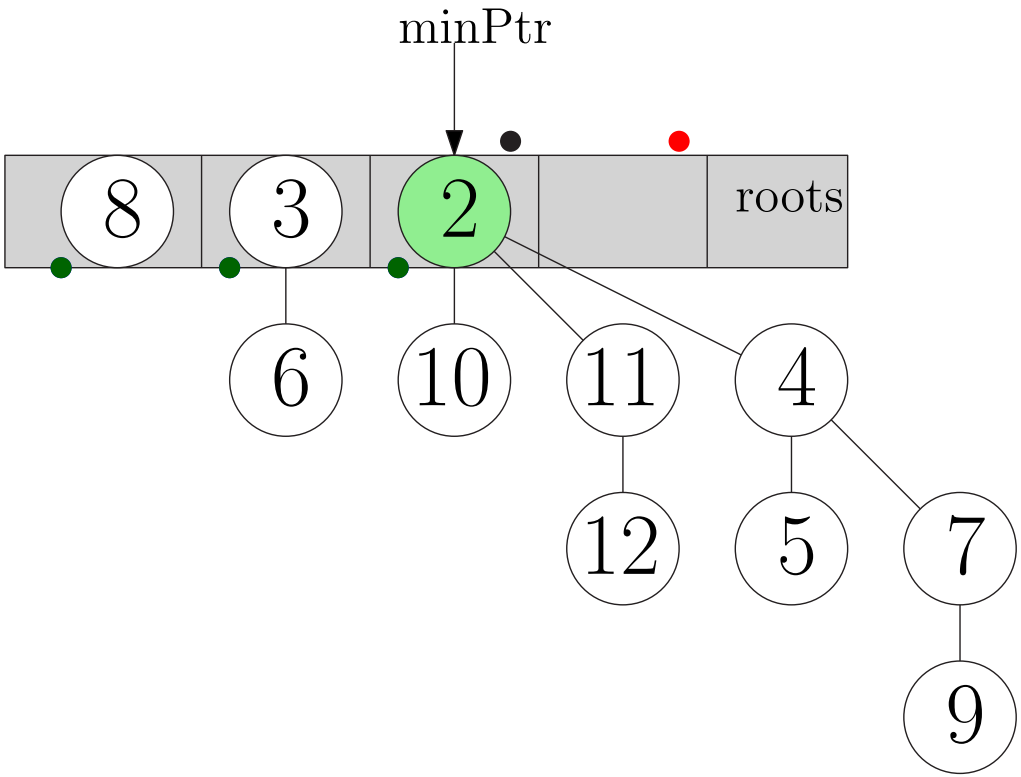
Fibonacci Heaps - Delete Min



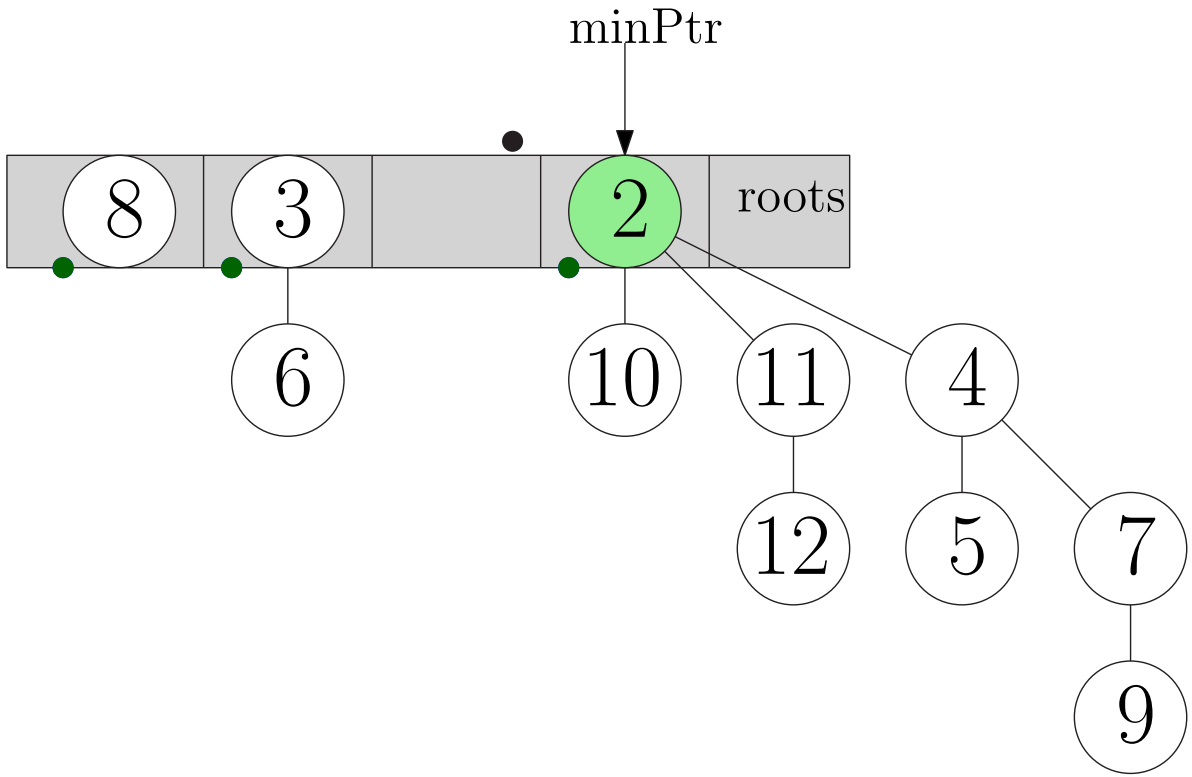
Fibonacci Heaps - Delete Min



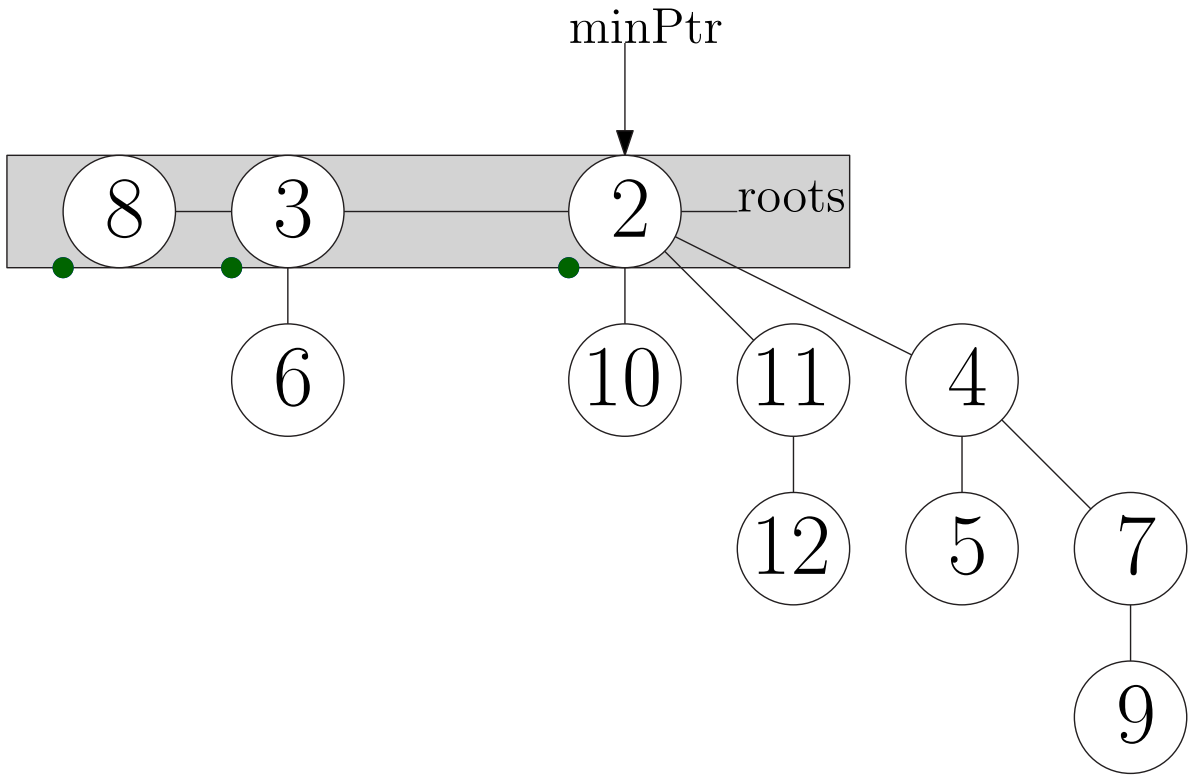
Fibonacci Heaps - Delete Min



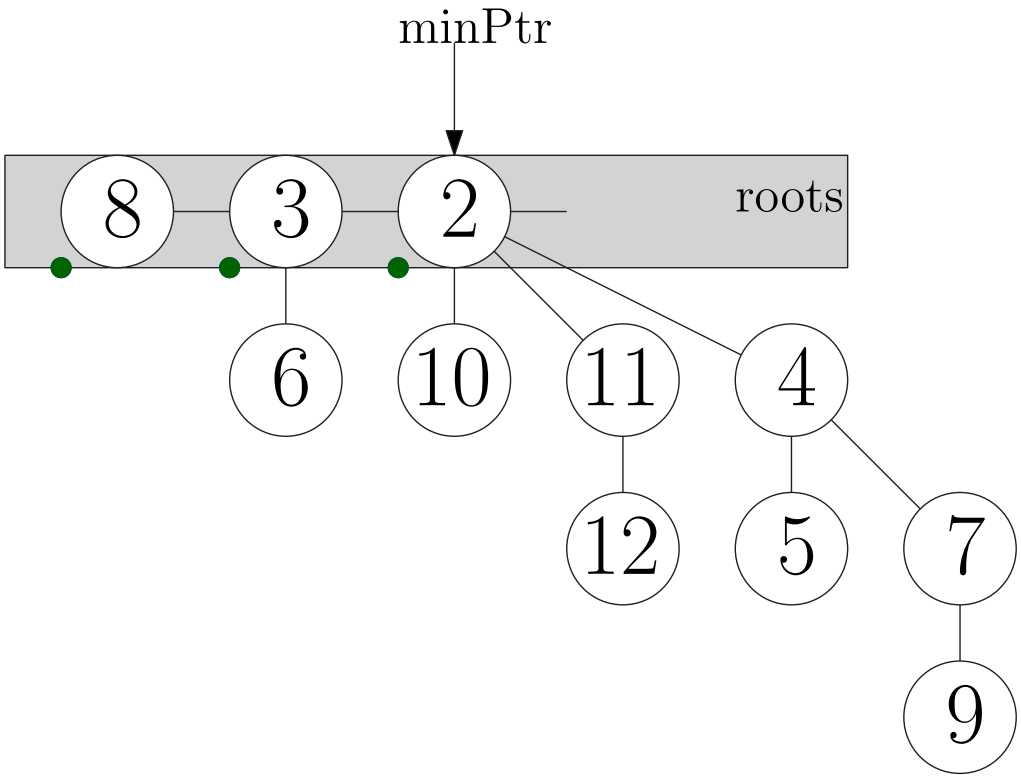
Fibonacci Heaps - Delete Min



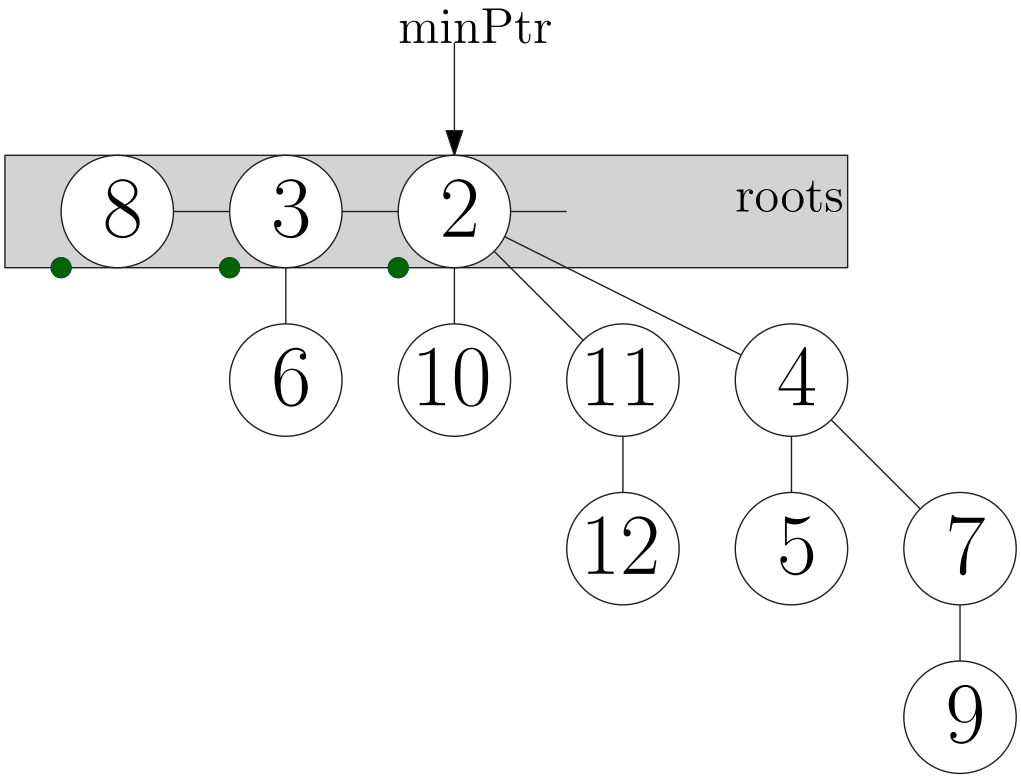
Fibonacci Heaps - Delete Min



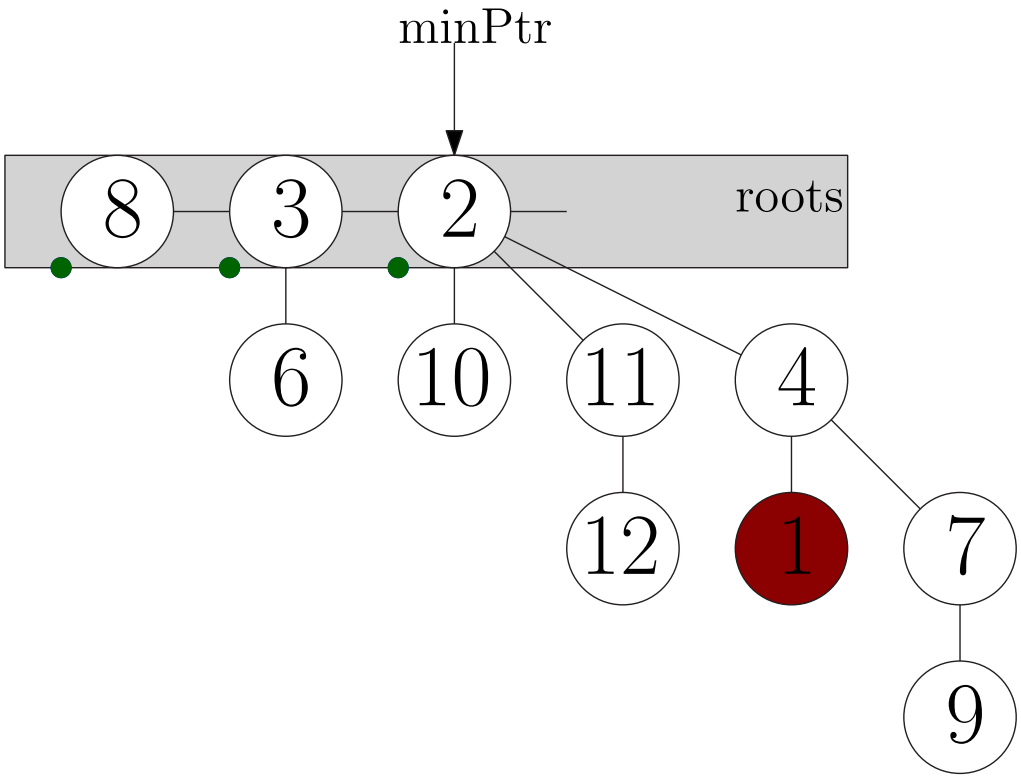
Fibonacci Heaps - Delete Min



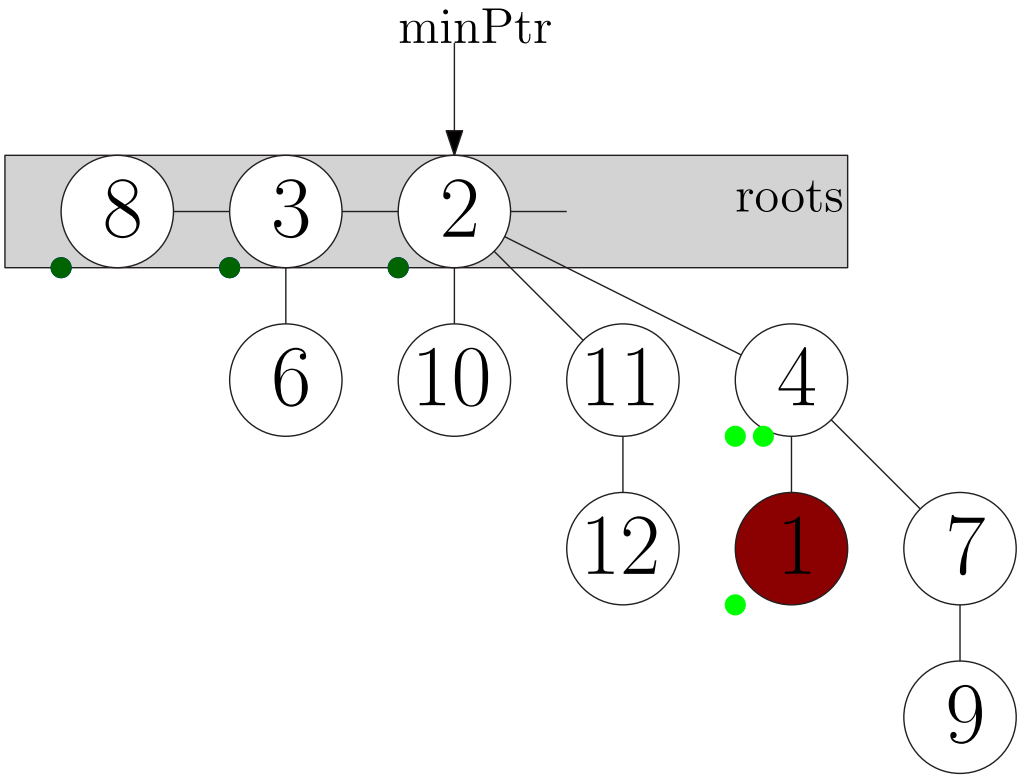
Fibonacci Heaps - Decrease Key



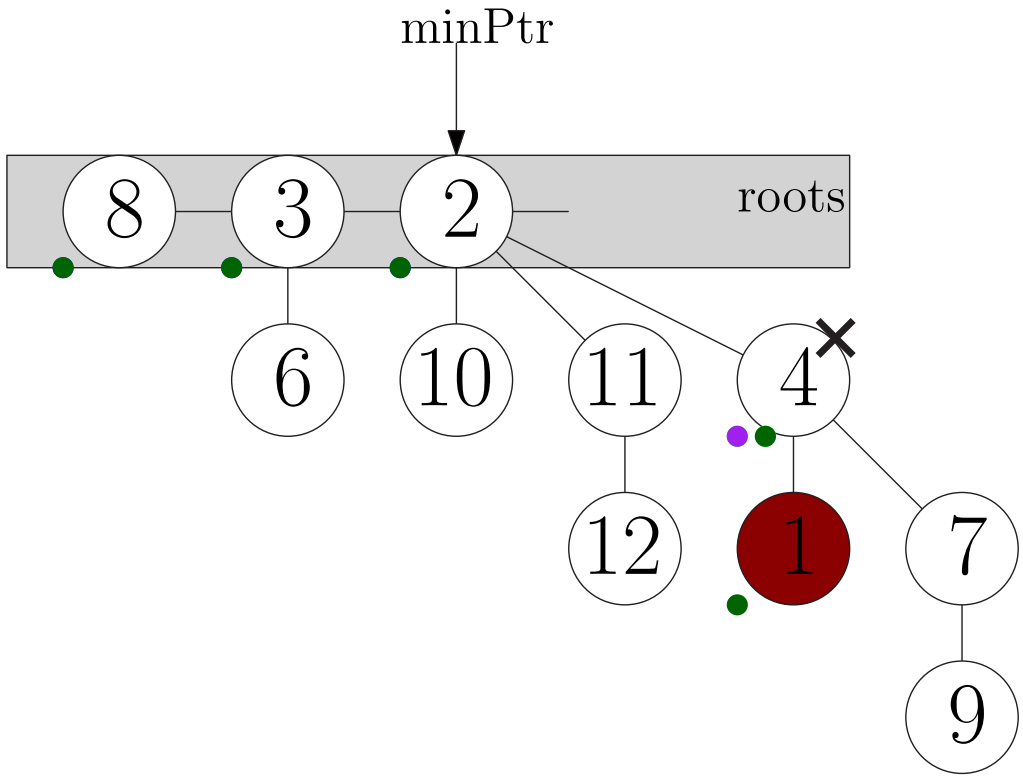
Fibonacci Heaps - Decrease Key



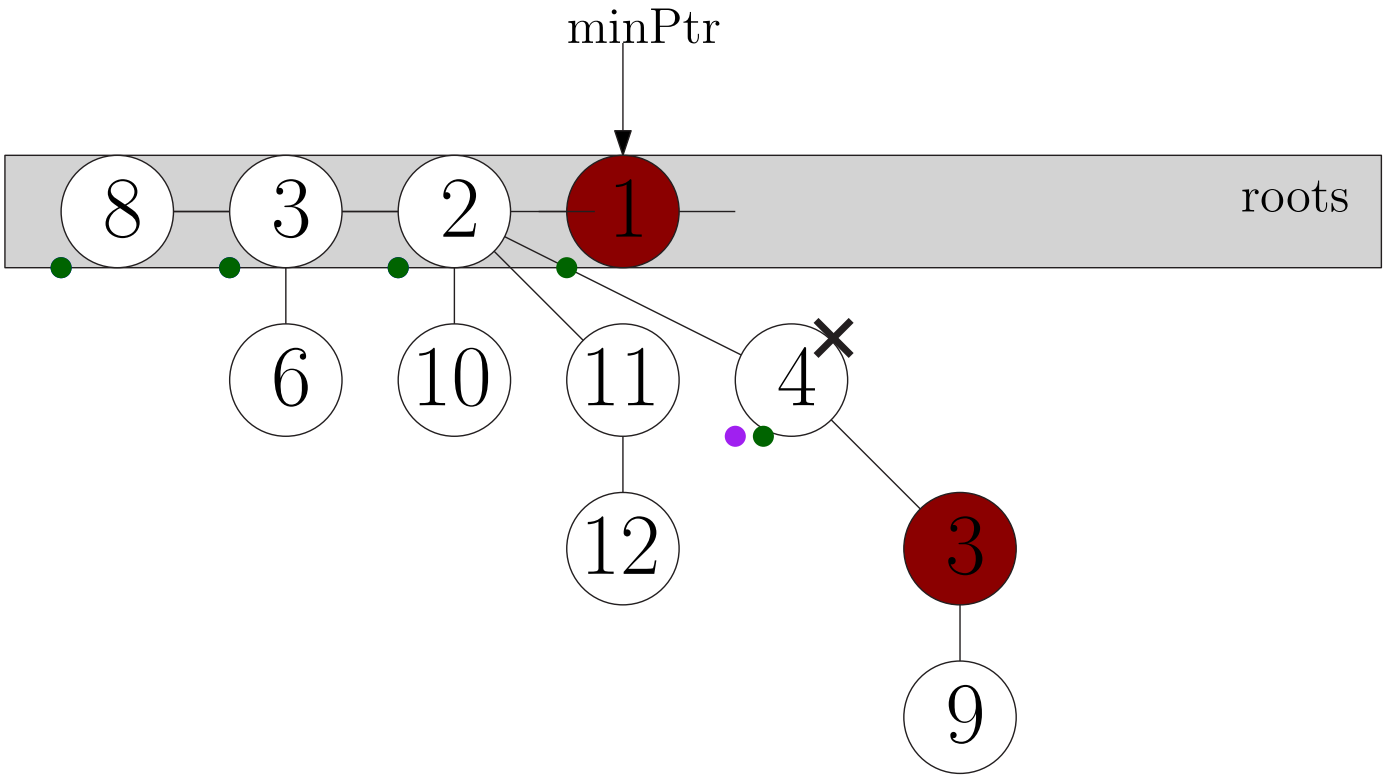
Fibonacci Heaps - Decrease Key



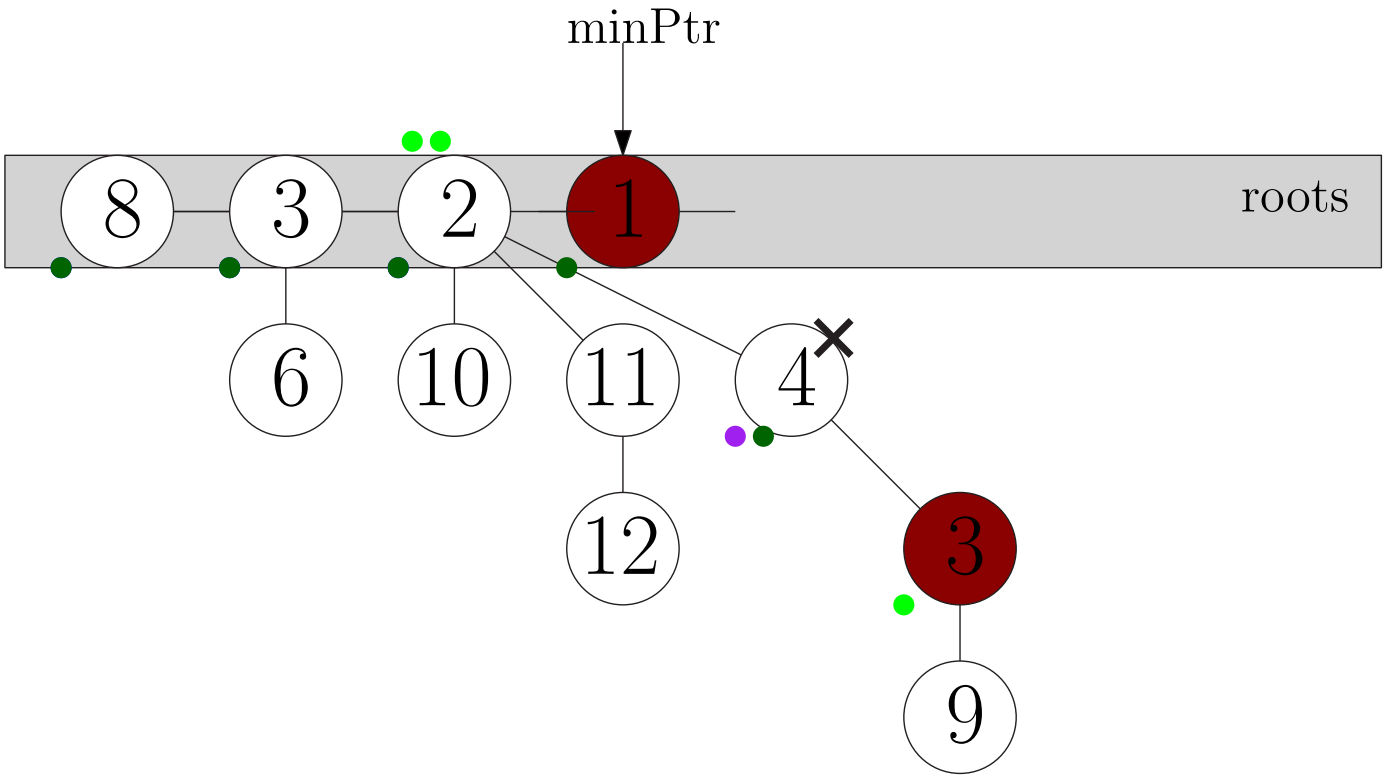
Fibonacci Heaps - Decrease Key



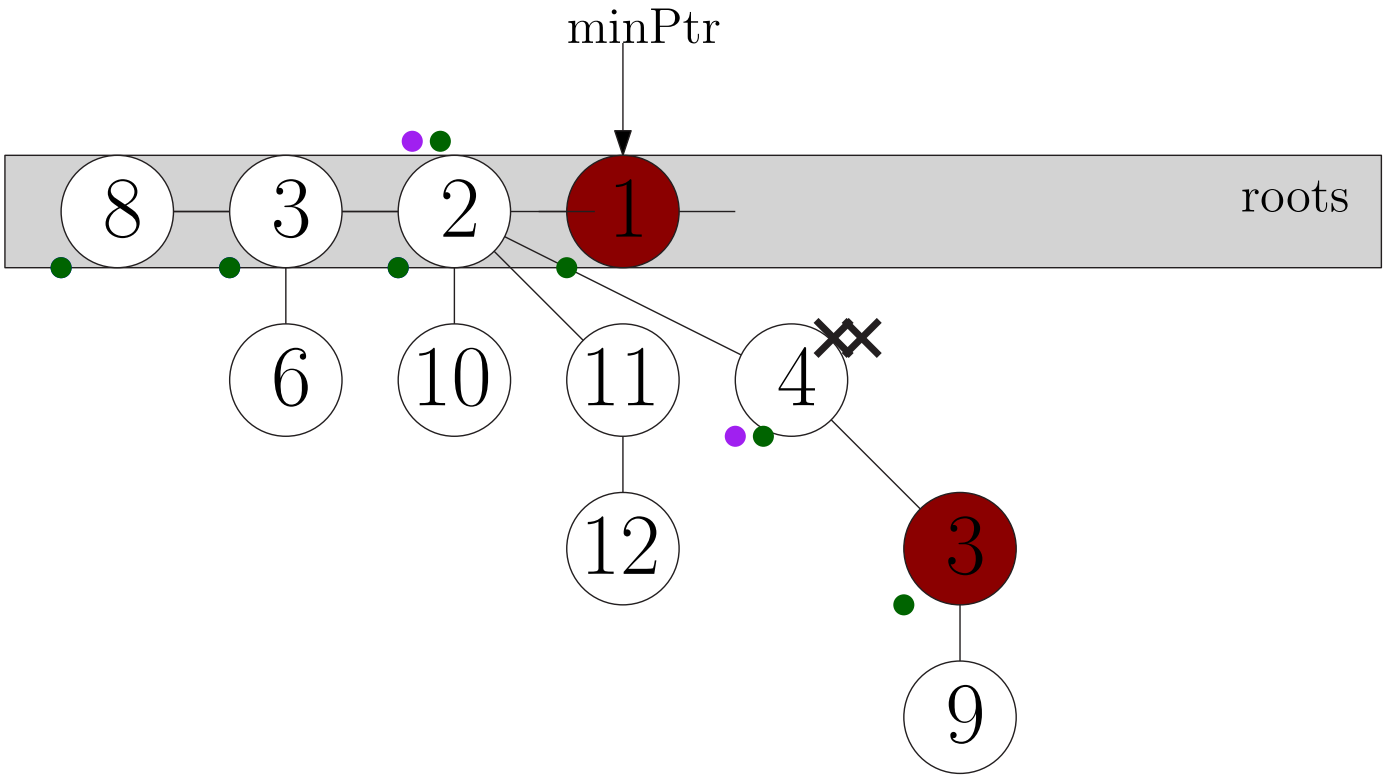
Fibonacci Heaps - Decrease Key



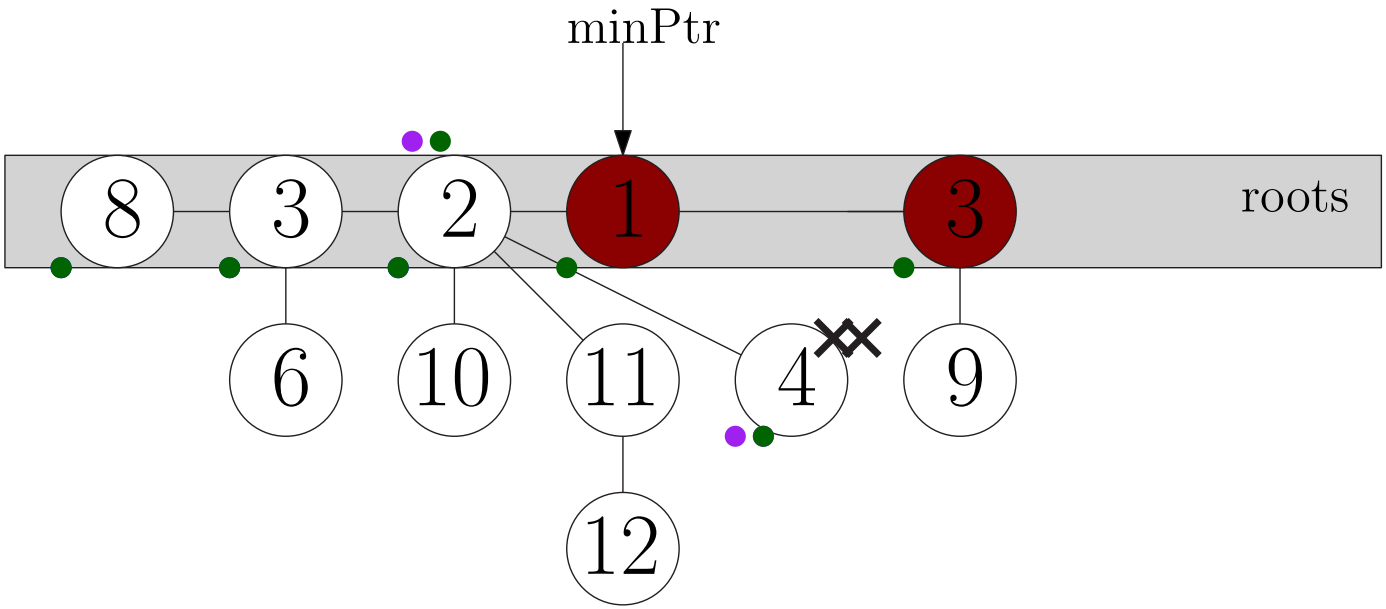
Fibonacci Heaps - Decrease Key



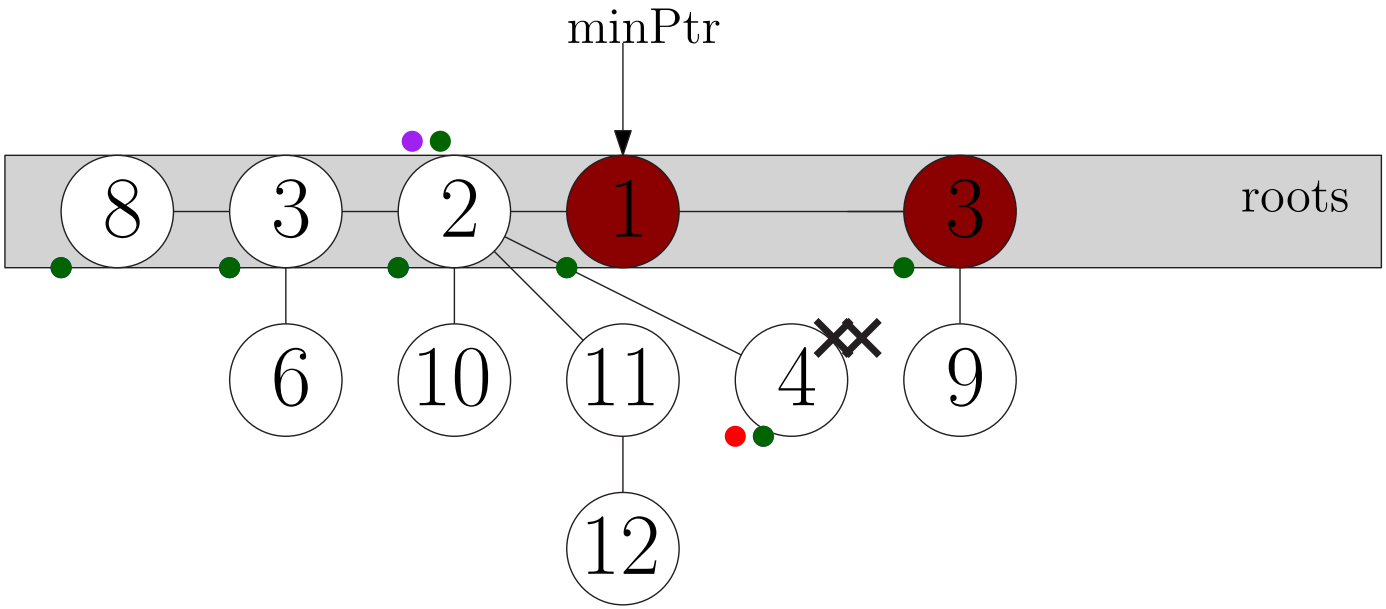
Fibonacci Heaps - Decrease Key



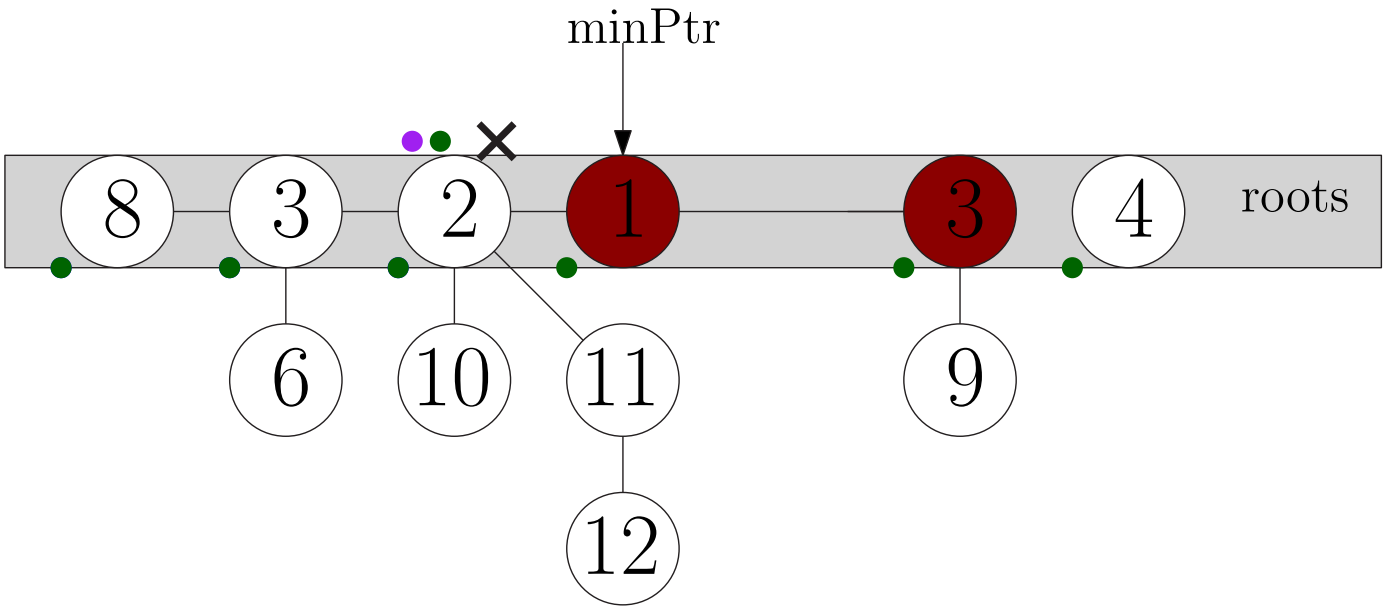
Fibonacci Heaps - Decrease Key



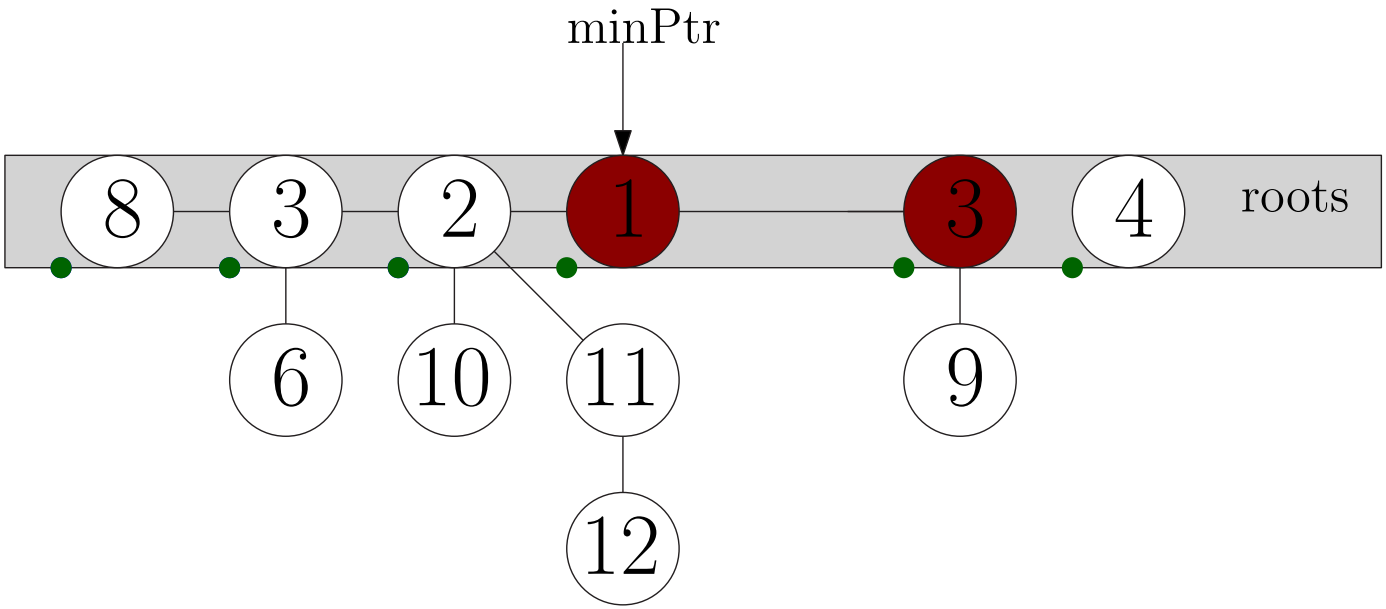
Fibonacci Heaps - Decrease Key



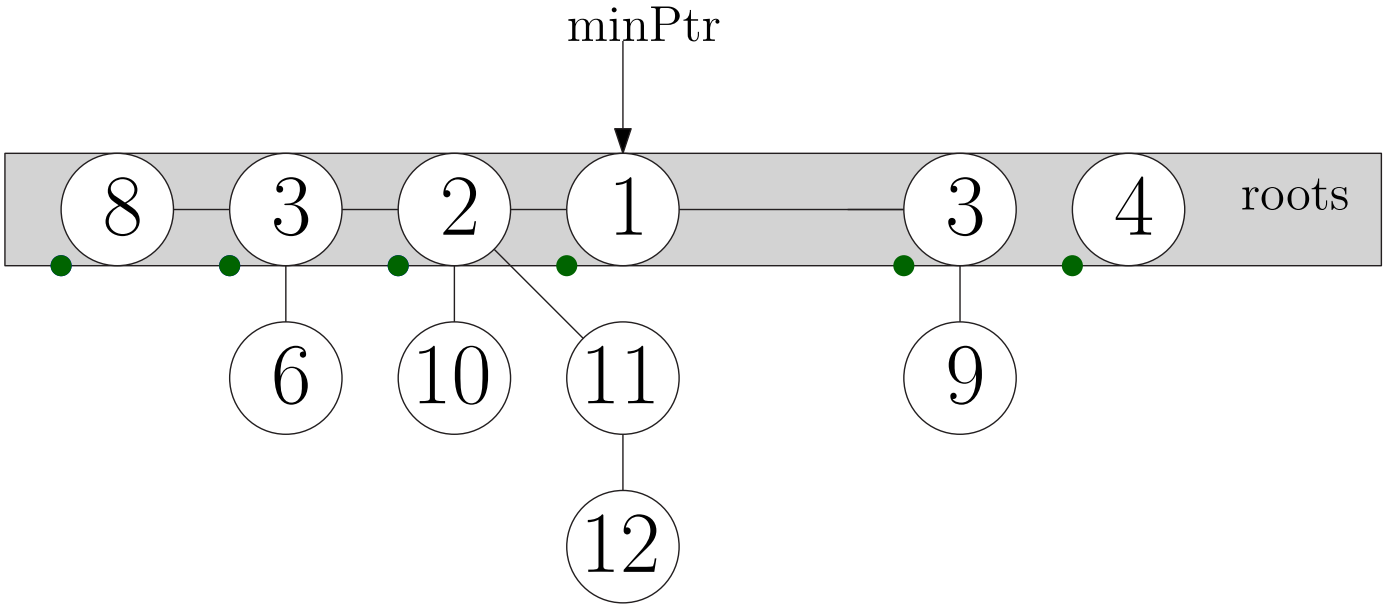
Fibonacci Heaps - Decrease Key



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

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

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

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

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

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

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

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

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

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

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

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

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

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

Fibonacci Heaps

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

Ende!

Feierabend!