Übung 1 – Algorithmen II

Moritz Laupichler, Hans-Peter Lehmann – *{moritz.laupichler, hans-peter.lehmann}@kit.edu*

`http://algo2.iti.kit.edu/AlgorithmenII_WS22.php`

Institut für Theoretische Informatik - Algorithmik II

Organisatorisches

Karlsruher Institut für Technologie

Vorlesungen:

Mo 09:45–11.15 Neue Chemie – Vorlesung
Di 15:45–16:30 Neue Chemie – Vorlesung

Übungen:

Di 16:30-17:15 Neue Chemie – Übung
75% Vertiefung der VL, 25% Besprechung der Übungsblätter

Ilias Forum:

Fragen zum Vorlesungsinhalt

Vorlesungsaufzeichnung:

Vorlesungen der letzten Jahre auf Youtube

↪ Link: *Algorithmen II (WS20/21, Englisch) - Playlist*
↪ Link: *Algorithmen II (WS19/20, Deutsch) - Playlist*

Organisatorisches

Übungsblätter:

- Ausgabe: 14-tägig, jeweils Dienstag
- Musterlösung: 14 Tage nach Veröffentlichung
- Besprechung:
  - 14 Tage nach Veröffentlichung
  - pro Blatt 1-2 Aufgaben ← **Abstimmung auf Ilias**
  - Abstimmen bis Sonntag vor Besprechung
- 1. Blatt:
  - Ausgabe am 08.11.2022
  - Abstimmung bis 20.11.2022
  - Besprechung und Musterlösung am 22.11.2022

Organisatorisches

Sprechstunden:
Corona-bedingt: Ilias Forum oder E-Mail bevorzugt

Letzte Vorlesung:
14.02.2023

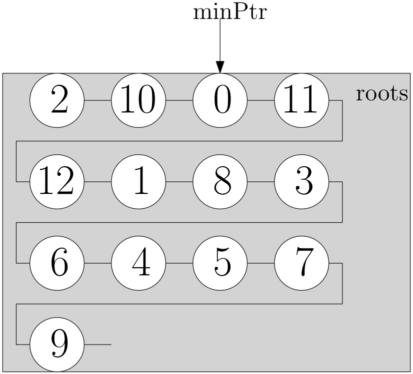
Klausur:
16.03.2023, 11:30 Uhr

Inhalt

Themenübersicht für heute

- Fibonacci Heaps
  - Wiederholung der Operationen
  - Ammortisierte Analyse mit Potentialmethode

Fibonacci Heaps - Insert



Fibonacci Heaps - Insert

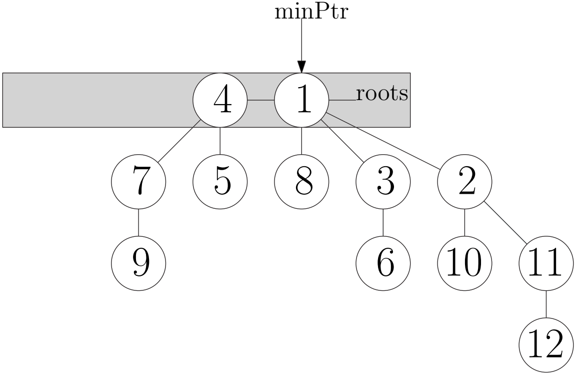
- Insert in konstanter Zeit
- Erzeugt großen Wald einzelner Knoten
- Entspricht ohne **union** nur linearer Liste

Fibonacci Heaps - Delete Min

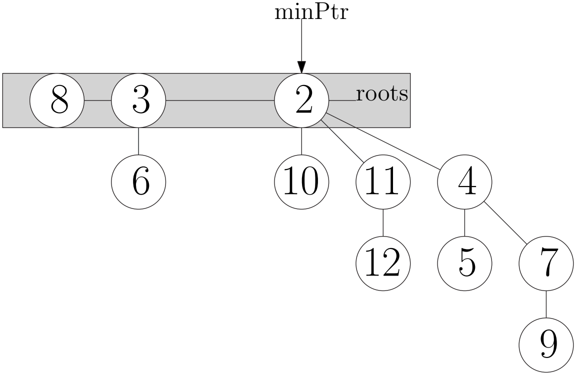
Delete Min:

- delete minimum m from forest
- for each child h of m:
  - make new tree with root h
- while $\exists a, b \in$ forest with rank$(a) = $ rank(b) do:
  - union(a, b)
- return m

Fibonacci Heaps - Delete Min



Fibonacci Heaps - Delete Min



Fibonacci Heaps - Decrease Key

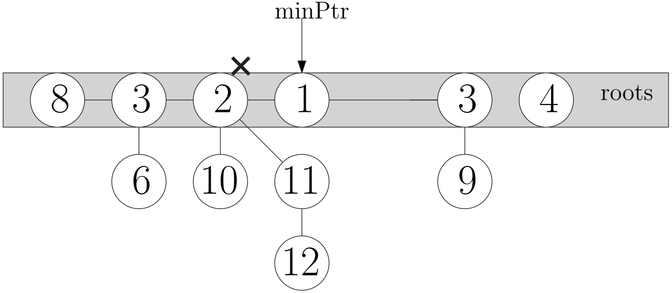
Decrease Key(h : Handle, k: Key):

- key(h) $\leftarrow k$
- cascadingCut(h):

cascadingCut(h: Handle)

- if h is not a root:
  - $p \leftarrow$ parent(h)
  - unmark h
  - cut(h)
  - if p is marked:
    - cascadingCut(p)
  - else:
    - mark p

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

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

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.

Einfaches Beispiel: Wachsendes Array

Datenstruktur

- Dynamisches Array A, das nur wächst
  - $\mathit{size}(s)$: Anzahl Elemente im Array in Zustand s
  - $\mathit{cap}(s)$: Kapazität in Zustand s (= Anzahl Elemente, für die Speicher allokiert ist)
- $\texttt{push\_back}(s, e : \text{Element}) : s'$
  - if $\mathit{size}(s) = \mathit{cap}(s)$ then
    - allocate memory for $\mathit{cap}(s') := 2 \cdot \mathit{cap}(s)$ elements
    - copy contents of A to allocated memory
  - A[$\mathit{size}(s)$] := e
  - $\mathit{size}(s') := \mathit{size}(s) + 1$
- $T_{\texttt{push\_back}}(s) = \mathcal{O}(1 + \mathit{copy}(s))$
  - $\mathit{copy}(s) :=$ #kopierte Elemente für `push_back` (0 oder $\mathit{cap}(s)$)

Einfaches Beispiel: Wachsendes Array

Konto-Methode

- nach “Umkopieren”: $\mathit{cap}/2$ Plätze frei
- `push_back`-Operationen für diese Plätze zahlen nächstes Umkopieren
- Jede `push_back`-Operation zahlt 2 Token $(\mathcal{O}(1))$
- Bei nächstem Umkopieren genau *cap* Token angespart
- $\Rightarrow$ amortisierte Laufzeit $A_{\texttt{push\_back}}(s) = \mathcal{O}(1)$

Einfaches Beispiel: Wachsendes Array

Potential-Methode

$$\Phi(s) = 2size(s) - cap(s) + 1$$

- `push_back` überführt Array von Zustand s nach s'
- Erinnerung: $T_{\texttt{push\_back}}(s) = \mathcal{O}(1 + copy(s))$
- Es gilt $copy(s) = cap(s') - cap(s)$
- Amortisierte Kosten: $A_{\texttt{push\_back}}(s) = T_{\texttt{push\_back}}(s) + \Delta\Phi(s)$
- $\Delta\Phi(s) = 2\Delta size(s) - \Delta cap(s) = 2 - copy(s)$
  - $size(s') = size(s) + 1 \Rightarrow \Delta size(s) = 1$
  - $cap(s') = cap(s) + copy(s) \Rightarrow \Delta cap(s) = copy(s)$

$\Rightarrow$ $A_{\texttt{push\_back}}(s) = \mathcal{O}(1 + copy(s)) + 2 - copy(s) = \mathcal{O}(1)$

Potentialfunktion für Fibonacci-Heaps

Karlsruher Institut für Technologie

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

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

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

Fibonacci Heaps

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

Ende!

Feierabend!