

Grundlagen: Algorithmen und Datenstrukturen

Prof. Dr. Hanjo Täubig

Lehrstuhl für Effiziente Algorithmen
(Prof. Dr. Ernst W. Mayr)
Institut für Informatik
Technische Universität München

Sommersemester 2010



Übersicht

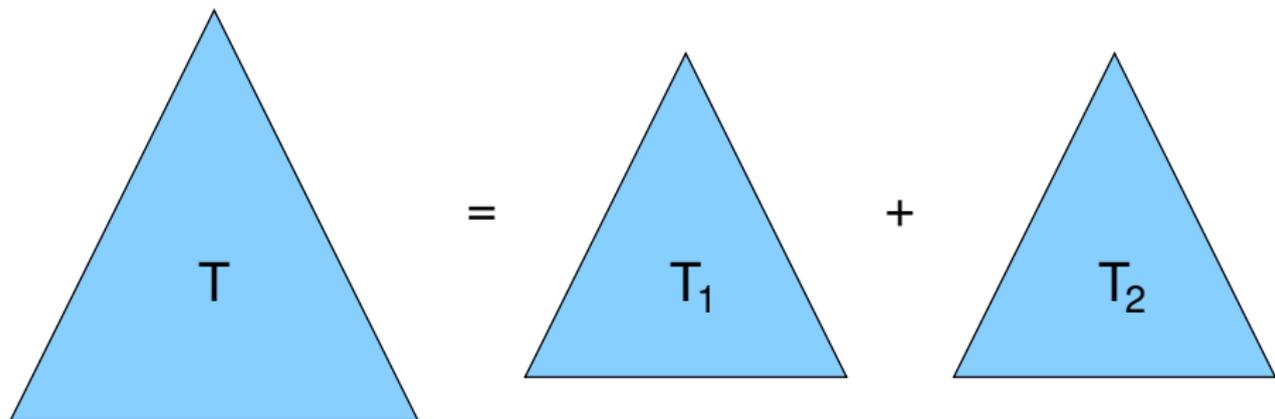
- 1 Suchstrukturen
 - (a, b) -Bäume
- 2 Graphen

(a, b)-Baum

Weitere Operationen:

- **Aufspaltung**

spalte (a, b)-Baum T bei Schlüssel k in zwei (a, b)-Bäume T_1 und T_2 auf



(a, b)-Baum

split

- Sequenz $q = \langle w, \dots, x, y, \dots, z \rangle$ soll aufgespalten werden bei Schlüssel y in Sequenzen $q_1 = \langle w, \dots, x \rangle$ und $q_2 = \langle y, \dots, z \rangle$
- betrachte Pfad von Wurzel zum Blatt y
- spalte auf diesem Pfad jeden Knoten v in zwei Knoten v_ℓ und v_r
- v_ℓ bekommt Kinder links vom Pfad, v_r bekommt Kinder rechts vom Pfad (evt. gibt es Knoten ohne Kinder)
- Knoten mit Kind(ern) werden als Wurzeln von (a, b)-Bäumen interpretiert
- Konkatenation der linken Bäume zusammen mit einem neuen ∞ -Dummy ergibt einen Baum für die Elemente bis x
- Konkatenation von $\langle y \rangle$ zusammen mit den rechten Bäumen ergibt einen Baum für die Elemente ab y

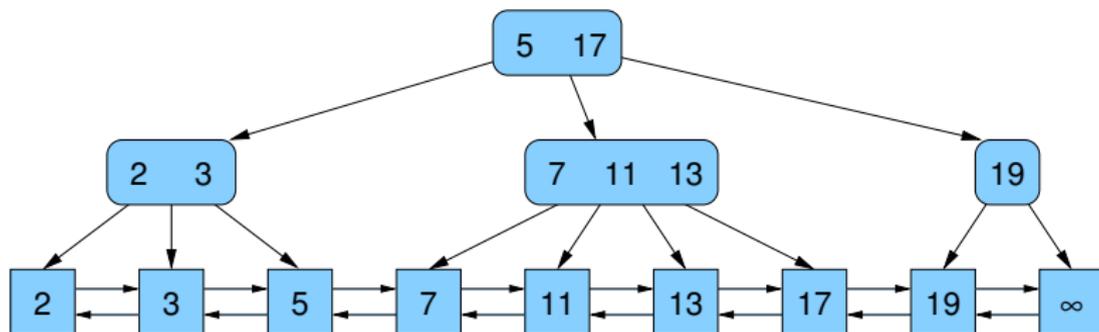
(a, b)-Baum

split

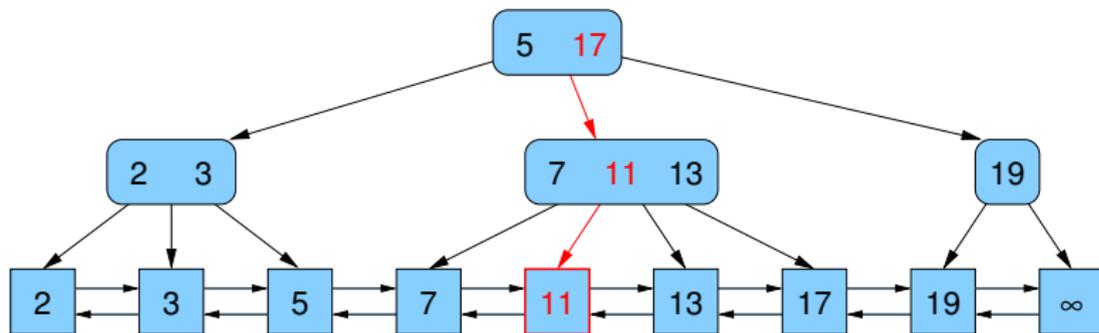
- diese $O(\log n)$ Konkatenationen können in Gesamtzeit $O(\log n)$ erledigt werden
- Grund: die linken / rechten Bäume haben echt monoton fallende / wachsende Höhe
- Seien z.B. r_1, r_2, \dots, r_k die Wurzeln der linken Bäume und h_1, h_2, \dots, h_k deren Höhen
- verbinde zuerst r_{k-1} und r_k in Zeit $O(1 + h_{k-1} - h_k)$, dann r_{k-2} mit dem Ergebnis in Zeit $O(1 + h_{k-2} - h_{k-1})$, dann r_{k-3} mit dem Ergebnis in Zeit $O(1 + h_{k-3} - h_{k-2})$ usw.
- Gesamtzeit:

$$O\left(\sum_{1 \leq i < k} (1 + h_i - h_{i+1})\right) = O(k + h_1 - h_k) \in O(\log n)$$

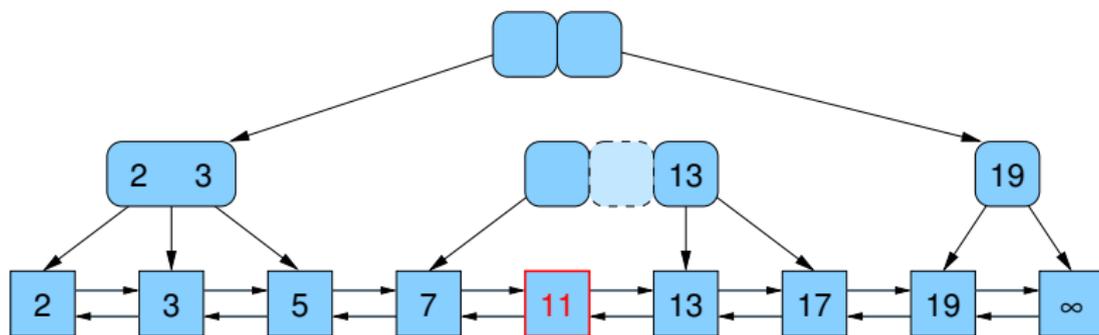
(a, b)-Baum / split



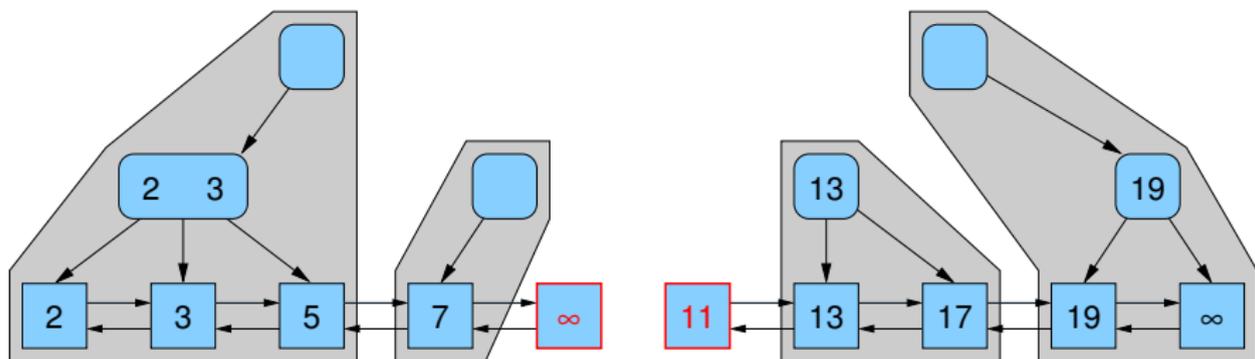
(a, b)-Baum / split



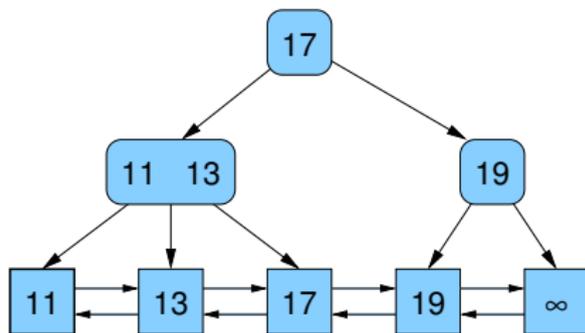
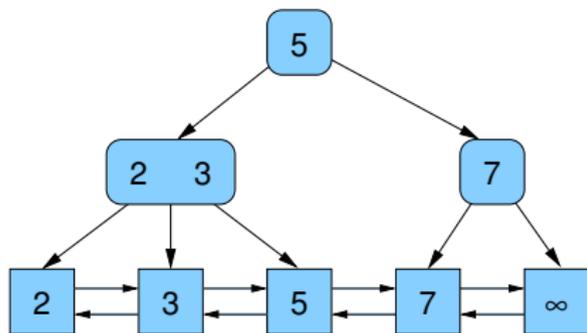
(a, b)-Baum / split



(a, b)-Baum / split



(a, b)-Baum / split



Effizienz von insert/remove-Folgen

Satz

Es gibt eine Folge von n insert- und remove-Operationen auf einem anfangs leeren $(2, 3)$ -Baum, so dass die Gesamtanzahl der Knotenaufspaltungen und -verschmelzungen in $\Omega(n \log n)$ ist.

Beweis: siehe Übung

Effizienz von insert/remove-Folgen

Satz

Für (a, b) -Bäume, die die erweiterte Bedingung $b \geq 2a$ erfüllen, gilt:

Für jede Folge von n insert- und remove-Operationen auf einem anfangs leeren (a, b) -Baum ist die Gesamtanzahl der Knotenaufspaltungen und -verschmelzungen in $O(n)$.

Beweis: amortisierte Analyse, nicht in dieser Vorlesung

Übersicht

1 Suchstrukturen

2 Graphen

- Netzwerke und Graphen
- Graphrepräsentation

Netzwerk

Objekt bestehend aus

- **Elementen** und
- Interaktionen bzw. **Verbindungen** zwischen den Elementen

eher *informales* Konzept

- keine exakte Definition
- Elemente und ihre Verbindungen können ganz unterschiedlichen Charakter haben
- manchmal manifestiert in **real** existierenden Dingen, manchmal nur gedacht (**virtuell**)

Beispiele für Netzwerke

- Kommunikationsnetze: Internet, Telefonnetz
- Verkehrsnetze: Straßen-, Schienen-, Flug-, Nahverkehrsnetz
- Versorgungsnetzwerke: Strom, Wasser, Gas, Erdöl
- wirtschaftliche Netzwerke: Geld- und Warenströme, Handel
- biochemische Netzwerke: Metabolische und Interaktionsnetzwerke
- biologische Netzwerke: Gehirn, Ökosysteme
- soziale / berufliche Netzwerke: virtuell oder explizit (Communities)
- Publikationsnetzwerke: Zitationsnetzwerk, Koautor-Netzwerk

Graph

formales / abstraktes Objekt bestehend aus

- Menge von **Knoten** V (engl. vertices, nodes)
- Menge von **Kanten** E (engl. edges, lines, links), die jeweils ein Paar von Knoten verbinden
- Menge von Eigenschaften der Knoten und / oder Kanten

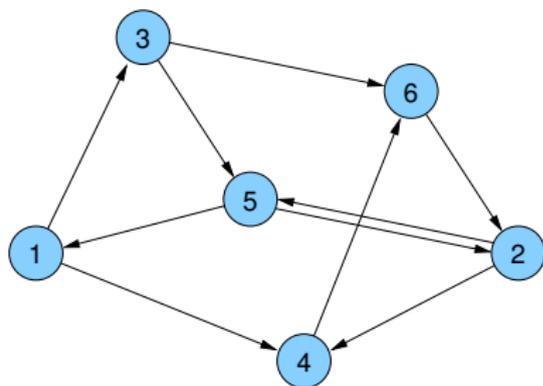
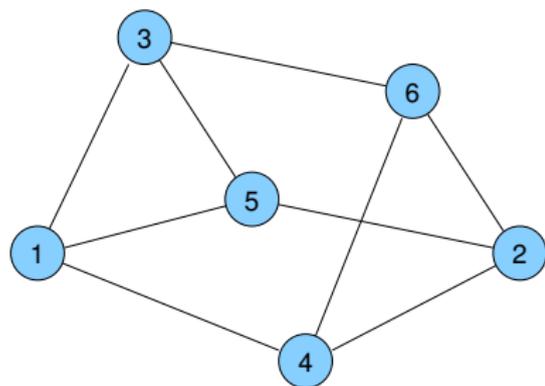
Notation:

- $G = (V, E)$
manchmal auch $G = (V, E, w)$ im Fall gewichteter Graphen
- Anzahl der Knoten: $n = |V|$
Anzahl der Kanten: $m = |E|$

Gerichtete und ungerichtete Graphen

Kanten bzw. Graphen

- ungerichtet: $E \subseteq \{\{v, w\} : v \in V, w \in V\}$
(ungeordnetes Paar von Knoten bzw. 2-elementige Teilmenge)
- gerichtet: $E \subseteq \{(v, w) : v \in V, w \in V\}$, also $E \subseteq V \times V$
(geordnetes Paar von Knoten)



Gerichtete und ungerichtete Graphen

Anwendungen:

- Ungerichtete Graphen:

symmetrische Beziehungen (z.B. $\{v, w\} \in E$ genau dann, wenn Person v und Person w verwandt sind)

- Gerichtete Graphen:

asymmetrische Beziehungen (z.B. $(v, w) \in E$ genau dann, wenn Person v Person w mag)

kreisfreie Beziehungen (z.B. $(v, w) \in E$ genau dann, wenn Person v Vorgesetzter von Person w ist)

hier:

- Modellierung von ungerichteten durch gerichtete Graphen
- Ersetzung ungerichteter Kanten durch je zwei antiparallele gerichtete Kanten

Nachbarn: Adjazenz, Inzidenz, Grad

- Sind zwei Knoten v und w durch eine Kante e verbunden, dann nennt man
 - ▶ v und w **adjazent** bzw. benachbart
 - ▶ v und e **inzident** (ebenso w und e)

- Anzahl der Nachbarn eines Knotens v : **Grad $\deg(v)$**
bei gerichteten Graphen:
 - ▶ Eingangsgrad: **$\deg^-(v) = |\{(w, v) \in E\}|$**
 - ▶ Ausgangsgrad: **$\deg^+(v) = |\{(v, w) \in E\}|$**

Annahmen

- Graph (also Anzahl der Knoten und Kanten) ist **endlich**
- Graph ist **einfach**, d.h. E ist eine Menge und keine Multimenge (anderenfalls heißt G Multigraph)
- Graph enthält **keine Schleifen** (Kanten von v nach v)

Gewichtete Graphen

In Abhängigkeit vom betrachteten Problem wird Kanten und / oder Knoten oft eine Eigenschaft (z.B. eine Farbe oder ein numerischer Wert, das **Gewicht**) zugeordnet (evt. auch mehrere), z.B.

- Distanzen (in Längen- oder Zeiteinheiten)
- Kosten
- Kapazitäten / Bandbreite
- Ähnlichkeiten
- Verkehrsdichte

Wir nennen den Graphen dann

- **knotengewichtet** bzw.
- **kantengewichtet**

Beispiel: $w : E \mapsto \mathbb{R}$

Schreibweise: $w(e)$ für das Gewicht einer Kante $e \in E$

Wege, Pfade und Kreise

- **Weg** (engl. *walk*) in einem Graphen $G = (V, E)$: alternierende Folge von Knoten und Kanten $x_0, e_1, \dots, e_k, x_k$, so dass
 - ▶ $\forall i \in [0, k] : x_i \in V$ und
 - ▶ $\forall i \in [1, k] : e_i = \{x_{i-1}, x_i\}$ bzw. $e_i = (x_{i-1}, x_i) \in E$.
- **Länge** eines Weges: Anzahl der enthaltenen Kanten
- Ein Weg ist ein **Pfad**, falls er (in sich) kantendisjunkt ist, falls also gilt: $e_i \neq e_j$ für $i \neq j$.
- Ein Pfad ist ein **einfacher Pfad**, falls er (in sich) knotendisjunkt ist, falls also gilt: $x_i \neq x_j$ für $i \neq j$.
- Ein Weg heißt **Kreis** (engl. *cycle*), falls $x_0 = x_k$.

Operationen

Graph G : Datenstruktur (Typ / Klasse, Variable / Objekt) für Graphen

Node: Datenstruktur für Knoten, **Edge**: Datenstruktur für Kanten

Operationen:

- **G.insert**(Edge e): $E := E \cup \{e\}$
- **G.remove**(Key i , Key j): $E := E \setminus \{e\}$
für Kante $e = (v, w)$ mit $\text{key}(v) = i$ und $\text{key}(w) = j$
- **G.insert**(Node v): $V := V \cup \{v\}$
- **G.remove**(Key i): sei $v \in V$ der Knoten mit $\text{key}(v) = i$
 $V := V \setminus \{v\}$, $E := E \setminus \{(x, y) : x = v \vee y = v\}$
- **G.find**(Key i): gib Knoten v mit $\text{key}(v) = i$ zurück
- **G.find**(Key i , Key j): gib Kante (v, w) mit $\text{key}(v) = i$ und $\text{key}(w) = j$ zurück

Operationen

Anzahl der Knoten **konstant**

$$\Rightarrow V = \{0, \dots, n - 1\}$$

- (Knotenschlüssel durchnummeriert)

Anzahl der Knoten **variabel**

- Hashing kann verwendet werden für ein Mapping der n Knoten in den Bereich $\{0, \dots, O(n)\}$

\Rightarrow nur konstanter Faktor der Vergrößerung gegenüber statischer Datenstruktur

Graphrepräsentation

Darstellung von Graphen im Computer?

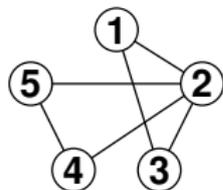
Vor- und Nachteile bei z.B. folgenden Fragen:

- Sind zwei gegebene Knoten v und w **adjazent**?
- Was sind die **Nachbarn** eines Knotens?
- Welche Knoten sind (direkte oder indirekte) **Vorgänger** bzw. **Nachfolger** eines Knotens v in einem gerichteten Graphen?
- Wie aufwendig ist das **Einfügen** oder **Löschen** eines Knotens bzw. einer Kante?

Graphrepräsentationen

- Kantenliste
- Adjazenzmatrix
- Inzidenzmatrix
- Adjazenzarray
- Adjazenzliste
- implizit

Kantenliste



$\{1, 2\}, \{1, 3\}, \{2, 3\}, \{2, 4\}, \{2, 5\}, \{4, 5\}$

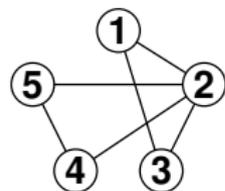
Vorteil:

- Speicherbedarf $O(m + n)$
- Einfügen von Knoten und Kanten in $O(1)$
- Löschen von Kanten per Handle in $O(1)$

Nachteil:

- $G.find(\text{Key } i, \text{Key } j)$: im worst case $\Theta(m)$
- $G.remove(\text{Key } i, \text{Key } j)$: im worst case $\Theta(m)$
- Nachbarn nur in $O(m)$ feststellbar

Adjazenzmatrix



$$\begin{pmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \end{pmatrix}$$

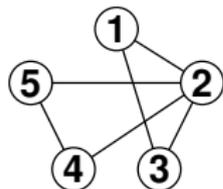
Vorteil:

- in $O(1)$ feststellbar, ob zwei Knoten Nachbarn sind
- ebenso Einfügen und Löschen von Kanten

Nachteil:

- kostet $O(n^2)$ Speicher, auch bei Graphen mit $o(n^2)$ Kanten
- Finden aller Nachbarn eines Knotens kostet $O(n)$
- Hinzufügen neuer Knoten ist schwierig

Inzidenzmatrix

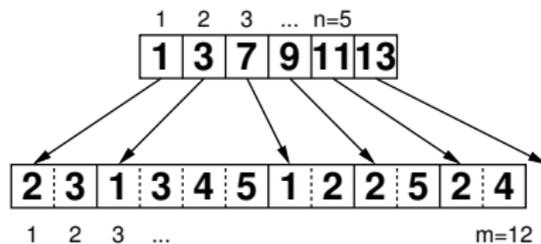
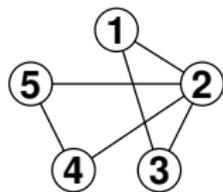


$$\begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 \end{pmatrix}$$

Nachteil:

- kostet $O(mn)$ Speicher

Adjazenzarray



Vorteil:

- Speicherbedarf:

gerichtete Graphen: $n + m + \Theta(1)$

(hier noch kompakter als Kantenliste mit $2m$)

ungerichtete Graphen: $n + 2m + \Theta(1)$

Nachteil:

- Einfügen und Löschen von Kanten ist schwierig, deshalb nur für *statische* Graphen geeignet