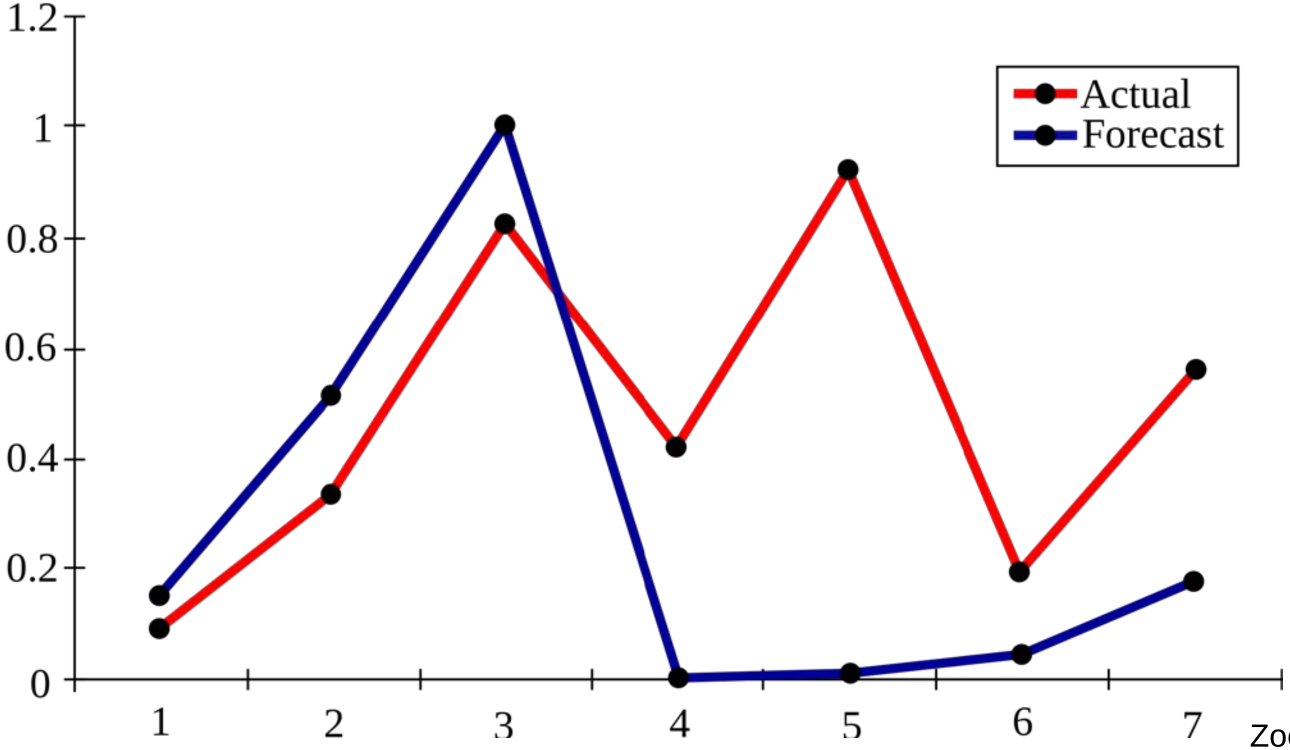
Datenstrukturen, Algorithmen und Programmierung 2 (DAP2)

Vergleich von Zeitreihen

- Zeitreihen treten in vielen Anwendungen auf
- Wie misst man deren Ähnlichkeit?



ZooFari, Wikimedia

Dynamic Time Warping

Idee

- Summe der Abstände eines optimalen “*warp*” auf die andere Zeitreihe
- wobei der *warp* alle Messpunkte zuordnet und *monoton* ist:
  für i>j gibt es k>l so dass (i,k) und (j,l) zugeordnet warden

Definition

Ein *warp* zwischen zwei Zeitreihen $R = (r_1,\dots,r_m)$ und $S = (s_1,\dots,s_n)$ ist eine Folge $C = (C_1,\dots,C_l)$ von Paaren $C_k = (i,j)$ mit

- $C_1 = (1,1)$ und $C_l = (m,n)$
- für $k = 1,\dots,l-1$ ist $C_k = (i,j)$ dann ist $C_{k+1} = \{(i+1,j), (i,j+1), (i+1,j+1)$

Dynamic Time Warping

Definition

Ein *warp* zwischen zwei Zeitreihen $R = (r_1,\ldots,r_m)$ und $S = (s_1,\ldots,s_n)$ ist eine Folge $C = (C_1,\ldots,C_l)$ von Paaren $C_k = (i,j)$ mit

- $C_1 = (1,1)$ und $C_l = (m,n)$
- für $k = 1,\ldots,l-1$ ist $C_k = (i,j)$ dann ist $C_{k+1} \in \{(i+1,j), (i,j+1), (i+1,j+1)\}$

Für zwei Zeitreihen R,S ist ihr dynamic time warping Abstand

$$\mathrm{dtw}(R,S) = \min_{\text{warp } C} \sum_{(i,j) \in C} \mathrm{dist}(r_i,s_j)$$

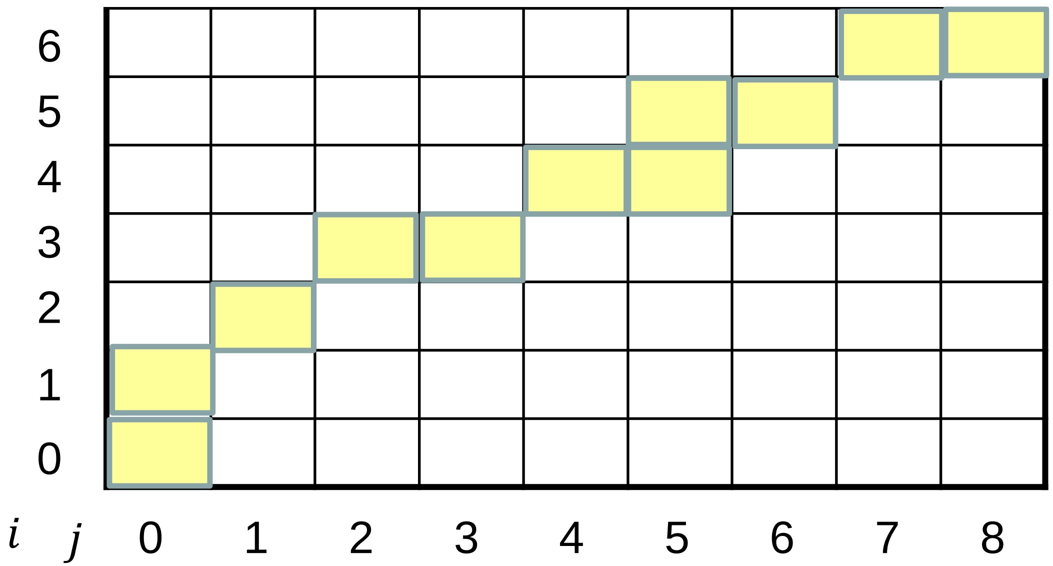
Frage

Wie lässt sich der dynamic time warping Abstand effizient berechnen?

Dynamic Time Warping

Beobachtung

Ein *warp* lässt sich darstellen als monotoner Pfad im mn Diagram.



Problem

Es gibt mehr als 2^m solcher Pfade!

Dynamic Time Warping

Rekursion

$$\mathrm{cost}(i,j) = \begin{cases} 0 & \text{falls } i = j = 0 \\ \infty & \text{falls } ij = 0 \wedge i+j > 0 \\ \mathrm{dist}(i,j) + \min\{\mathrm{cost}(i-1,j),\ \mathrm{cost}(i,j-1),\ \mathrm{cost}(i-1,j-1)\} & \text{sonst} \end{cases}$$

gibt den $\mathrm{dtw}(R_i,S_j)$ an.

Dynamic Time Warping

Algorithmus

$\mathrm{DTW} - \mathrm{Abstand}(R, S)$

$\quad m \leftarrow \mathrm{length}[R]$

$\quad n \leftarrow \mathrm{length}[S]$

$\quad \textbf{new}\ \mathrm{array}\ C[0..m][0..n]$

$\quad \textbf{for}\ i \leftarrow 1\ \textbf{to}\ m\ \textbf{do}\ C[i][0] \leftarrow \infty$

$\quad \textbf{for}\ j \leftarrow 1\ \textbf{to}\ n\ \textbf{do}\ C[0][j] \leftarrow \infty$

$\quad C[0][0] \leftarrow 0$

$\quad \textbf{for}\ i \leftarrow 1\ \textbf{to}\ m\ \textbf{do}$

$\qquad \textbf{for}\ j \leftarrow 1\ \textbf{to}\ n\ \textbf{do}$

$\qquad\quad C[i][j] \leftarrow \mathrm{dist(i,j)} + \min\{C[i-1][j], C[i][j-1], C[i-1][j-1]\}$

$\quad \textbf{return}\ C[m][n]$

Dynamic Time Warping

Satz

Algorithmus $\mathrm{DTW} - \mathrm{Abstand}$ berechnet den dynamic time warping Abstand zweier Zeitreihen in $\mathbf{O}(nm)$ Zeit.

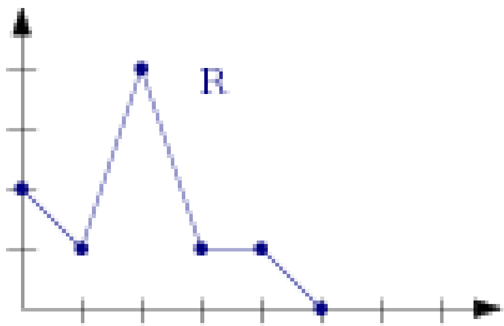
Beweisskizze

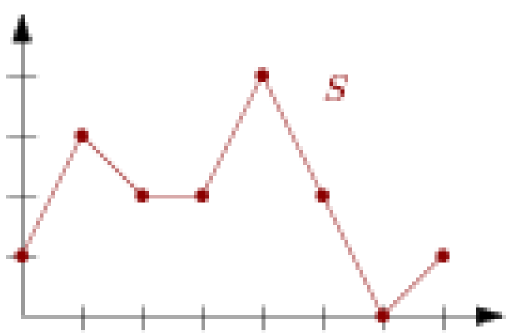
- Die Korrektheit folgt per Induktion über die Rekursion
- Laufzeit folgt sofort.

Dynamic Time Warping

Beispiel

R = (2, 1, 4, 1, 1, 0) und S = (1, 3, 2, 2, 4, 2, 0, 1)



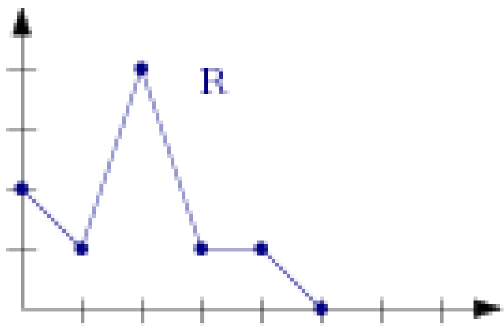


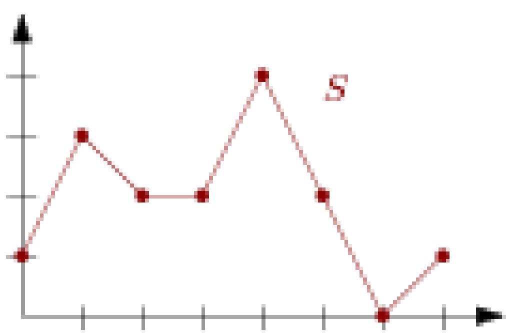
i \ j	0	1	2	3	4	5	6	7	8
6									
5									
4									
3									
2									
1									
0									

Dynamic Time Warping

Beispiel

R = (2, 1, 4, 1, 1, 0) und S = (1, 3, 2, 2, 4, 2, 0, 1)



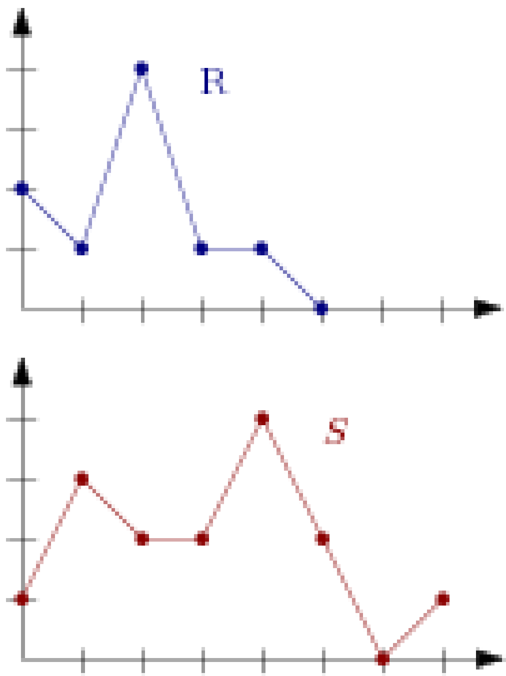


6	∞	5	7	6	6	8	7	5	6
5	∞	4	6	4	4	9	5	5	5
4	∞	4	4	3	4	6	4	5	5
3	∞	4	2	4	5	3	5	9	8
2	∞	1	3	3	3	5	5	5	5
1	∞	1	2	2	2	4	4	6	7
0	0	∞	∞	∞	∞	∞	∞	∞	∞
i *j*	0	1	2	3	4	5	6	7	8

Dynamic Time Warping

Beispiel

R = (2, 1, 4, 1, 1, 0) und S = (1, 3, 2, 2, 4, 2, 0, 1)



i \ j	0	1	2	3	4	5	6	7	8
6	∞	5	7	6	6	8	7	5	6
5	∞	4	6	4	4	9	5	5	5
4	∞	4	4	3	4	6	4	5	5
3	∞	4	2	4	5	3	5	9	8
2	∞	1	3	3	3	5	5	5	5
1	∞	1	2	2	2	4	4	6	7
0	0	∞	∞	∞	∞	∞	∞	∞	∞

Zusammenfassung der Vorlesung

Laufzeitanalyse

Maschinenmodell

- Eine Pseudocode-Instruktion braucht einen Zeitschritt
- Wird eine Instruktion r-mal aufgerufen, werden r Zeitschritte benötigt
- Formales Modell: Random Access Machines (RAM Modell)

Idee

- Ignoriere rechnerabhängige Konstanten
- Betrachte Wachstum von $T(n)$ für $n \to \infty$

„Asymptotische Analyse"

Laufzeitanalyse

O-Notation

- $\mathbf{O}\big(f(n)\big) = \{g(n)\colon \exists c > 0, n_0 > 0, \text{so dass für alle } n \geq n_0 \text{ gilt } g(n) \leq c \cdot f(n)\}$
- (wobei $f, g\colon \mathbb{N} \rightarrow \mathbb{R}^+$)

Interpretation

- $g(n) \in \mathbf{O}\big(f(n)\big)$ bedeutet, dass $g(n)$ für $n \rightarrow \infty$ höchstens genauso stark wächst wie $f(n)$
- Beim Wachstum ignorieren wir Konstanten

Korrektheitsbeweise

Ein triviales Beispiel

$\text{EinfacherAlgorithmus}(n)$

1. $X \leftarrow 10$
2. $Y \leftarrow n$
3. $X \leftarrow X + Y$
4. $\textbf{return } X$

Ein Korrektheitsbeweis vollzieht also das Programm Schritt für Schritt nach.

Behauptung

Der Algorithmus gibt den Wert $10 + n$ zurück.

Beweis:
Zu Beginn des Algorithmus sind alle Variablen bis auf den Parameter n undefiniert.
Der Befehl in Zeile 1 weist X den Wert 10 zu.
Der Befehl in Zeile 2 weist Y den Wert n zu.
Der Befehl in Zeile 3 weist X den Wert $X + Y$ zu. Da X vor der Zuweisung den Wert 10 enthielt und Y den Wert n, wird X auf $10 + n$ gesetzt. Der Befehl in Zeile 4 gibt X zurück. Da X zu diesem Zeitpunkt den Wert $10 + n$ hat, folgt die Behauptung.

Korrektheitsbeweise

Ein erstes nichttriviales Beispiel

MaxSearch(Array A)

1. $\max \leftarrow 1$
2. **for** $j \leftarrow 2$ **to** length$[A]$ **do**
3. **if** $A[j] > A[\max]$ **then** $\max \leftarrow j$
4. **return** max

Definition (Schleifeninvariante)

Eine Schleifeninvariante ist eine i.a. von der Anzahl i der Schleifendurchläufe abhängige Aussage $A(i)$, die zu Beginn des i-ten Schleifendurchlauf gilt. Mit $A(1)$ beziehen wir uns also auf den Zustand zu Beginn des ersten Durchlaufs. Dieser wird auch als Initialisierung bezeichnet.

Korrektheitsbeweise

Rekursion

Algorithmus Sum(A,n)

1. **If** n=1 **then** return A[1]
2. **else**
3. W= Sum(A,n-1)
4. **return** A[n] + W

<u>Beweis (Induktion über n):</u>
(I.A.) Ist $n = 1$, so gibt der Algorithmus in Zeile 1 den Wert $A[1]$ zurück. Dies ist korrekt.
(I.V.) Für $n - 1 > 0$ berechnet Sum$(A, n-1)$ die Summe der ersten $n-1$ Einträge von A.
(I.S.) Wir betrachten den Aufruf von Sum(A, n). Da $n > 1$ ist, wird der **else**-Fall der ersten **if**-Anweisung aufgerufen. Dort wird W auf Sum$(A, n-1)$ gesetzt. Nach I.V. ist dies die Summe der ersten $n-1$ Einträge von A. Nun wird in Zeile 4 $A[n] + W$, also die Summe der ersten n Einträge von A zurückgegeben.

Satz 3

Algorithmus Sum(A, n) berechnet die Summe der ersten n Einträge des Feldes A.

Teile & Herrsche

Teile & Herrsche (Divide & Conquer)

- Teile Eingabe in mehrere Teile auf
- Löse das Problem rekursiv auf den Teilen
- Füge die Teillösungen zu einer Gesamtlösung zusammen

Beispiel(Sortieren)

15	**7**	**6**	**13**	**25**	**4**	**9**	**12**

6	**7**	**13**	**15**

4	**9**	**12**	**25**

Teile & Herrsche

Laufzeiten der Form

$$T(n) = a \cdot T(n/b) + f(n)$$

Anzahl Unterprobleme

Aufwand für Aufteilen und Zusammenfügen

Größe der Unterprobleme
(bestimmt Höhe des Rekursionsbaums)

- (und $T(1)$ = const)

Gierige Algorithmen

Gierige Algorithmen

- Konstruiere Lösung Schritt für Schritt
- In jedem Schritt: Optimiere ein einfaches, lokales Kriterium

Beobachtung

- Man kann viele unterschiedliche gierige Algorithmen für ein Problem entwickeln
- Nicht jeder dieser Algorithmen löst das Problem korrekt

Beispiele

- Scheduling Probleme

Dynamische Programmierung

Beobachtung

- Die Funktionswerte werden bottom-up berechnet

Grundidee der dynamischen Programmierung

- Berechne die Funktionswerte iterativ und bottom-up

$\mathrm{FibDynamischeProgrammierung}(n)$

1. Initialisiere Feld $F[1..n]$
2. $F[1] \leftarrow 1$
3. $F[2] \leftarrow 1$
4. **for** $i \leftarrow 3$ **to** n **do**
5. $\quad F[i] \leftarrow F[i-1] + F[i-2]$
6. **return** $F[i]$

Dynamische Programmierung

Dynamische Programmierung

- Formuliere Problem rekursiv
- Löse die Rekursion „bottom-up“ durch schrittweises Ausfüllen einer Tabelle der möglichen Lösungen

Wann ist dynamische Programmierung effizient?

- Die Anzahl unterschiedlicher Funktionsaufrufe (Größe der Tabelle) ist klein
- Bei einer „normalen Ausführung“ des rekursiven Algorithmus ist mit vielen Mehrfachausführungen zu rechnen

Datenstrukturen

Drei grundlegende Datenstrukturen

- Feld
- sortiertes Feld
- doppelt verkettete Liste

Diskussion

- Alle drei Strukturen haben gewichtige Nachteile (beim Suchen/Updaten)
- Zeiger/Referenzen helfen beim Speichermanagement
- Sortierung hilft bei Suche ist aber teuer aufrecht zu erhalten

Datenstrukturen

Binäre Suchbäume

- Aufzählen der Elemente mit Inorder-Tree-Walk in $\mathbf{O}(n)$ Zeit
- Suche in $\mathbf{O}(h)$ Zeit
- Minimum/Maximum in $\mathbf{O}(h)$ Zeit
- Vorgänger/Nachfolger in $\mathbf{O}(h)$ Zeit

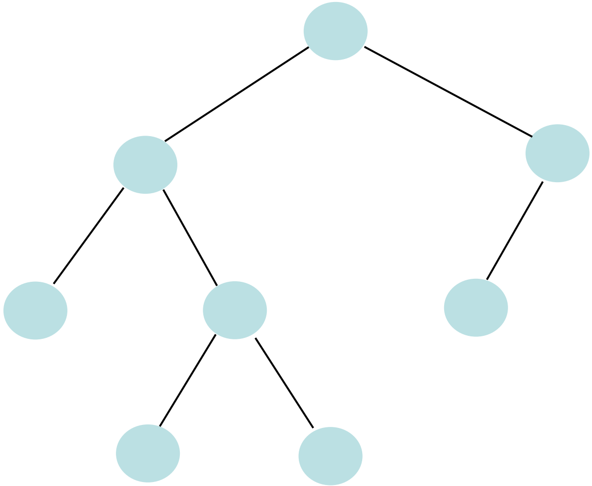
Dynamische Operationen?

- Einfügen und Löschen
- Müssen Suchbaumeigenschaft aufrecht erhalten
- Auswirkung auf Höhe des Baums?

Datenstrukturen

AVL-Bäume [Adelson-Velsky und Landis]

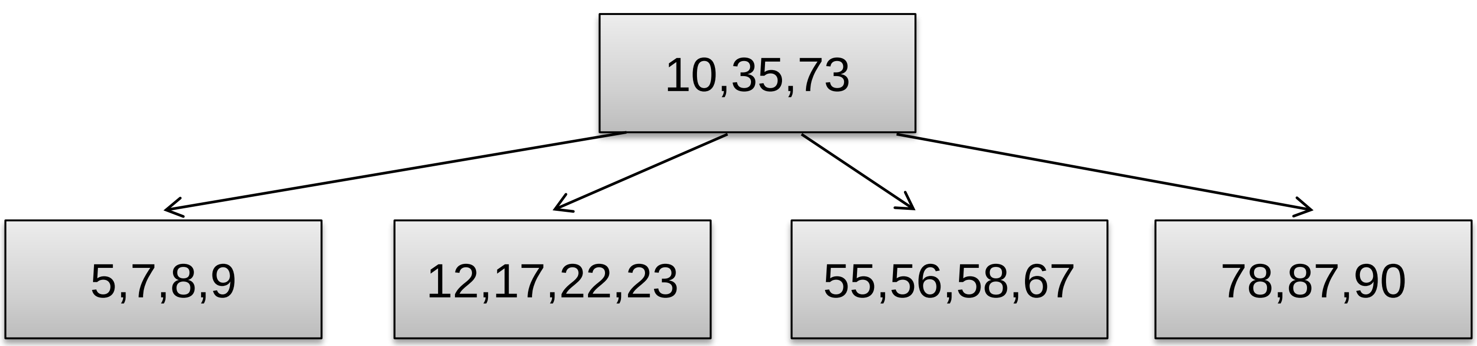
Ein Binärbaum heißt AVL-Baum, wenn für jeden Knoten gilt: Die Höhe seines linken und rechten Teilbaums unterscheidet sich höchstens um 1.



Big Data

B-Bäume - Grundidee

- Neuer Suchbaumstruktur mit „größeren Knoten“ und höherem Verzweigungsgrad
- Jeder Knoten enthält aufsteigend sortierte Folge von Schlüsseln
- k Schlüssel an einem Knoten partitionieren das Schlüsseluniversum in $k+1$ Intervalle
- für jedes solche Intervall gibt es einen Unterbaum, der alle Knoten des Intervalls enthält



Big Data

Datenströme

- Sehr große Datenmengen, die als Sequenz auftreten

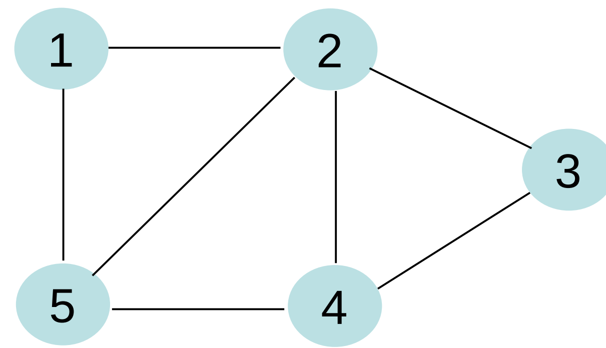
Datenstrommodell

- Daten treten als Sequenz auf
- Wenig Speicher
- Die Sequenz kann nur einmal und nur in der vorgegebenen Reihenfolge gelesen werden

KSP-Algorithmus

- Findet „Heavy Hitter“

Graphen

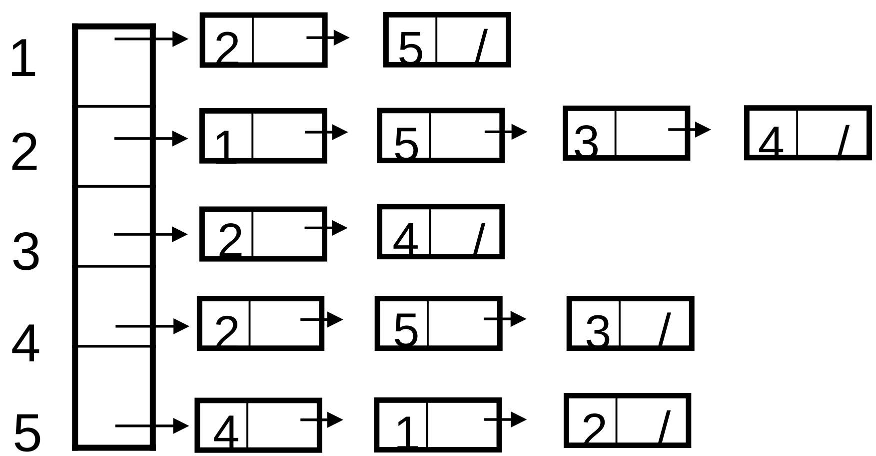


Arten von Graphen

- ungerichtet, gerichtet
- ungewichtet, gewichtet

Darstellung von Graphen

- Adjazenzlisten (dünne Graphen, $|E| \ll |V|^2$)
- Adjazenzmatrix (dichte Graphen, $|E|$ nah an $|V|^2$)

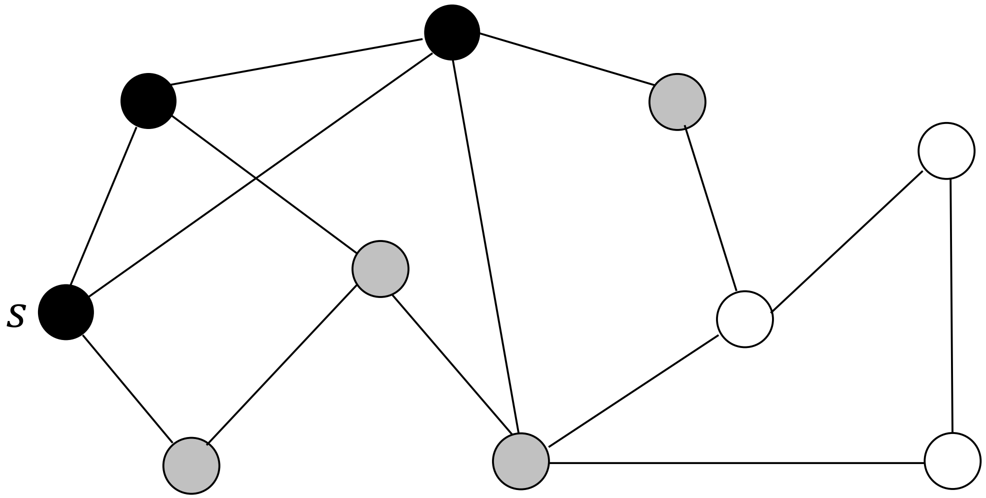


0	1	0	0	1
1	0	1	1	1
0	1	0	1	0
0	1	1	0	1
1	1	0	1	0

Graphalgorithmen

Tiefensuche und Breitensuche

- Tiefensuche bietet andere Möglichkeit (neben Breitensuche) zur Graphtraversierung in Laufzeit $\mathbf{O}(|V| + |E|)$
- Die Sortierung der Tiefensuche kann zum topologischen Sortieren benutzt werden



Graphalgorithmen

SSSP (pos. Kantengewichte):

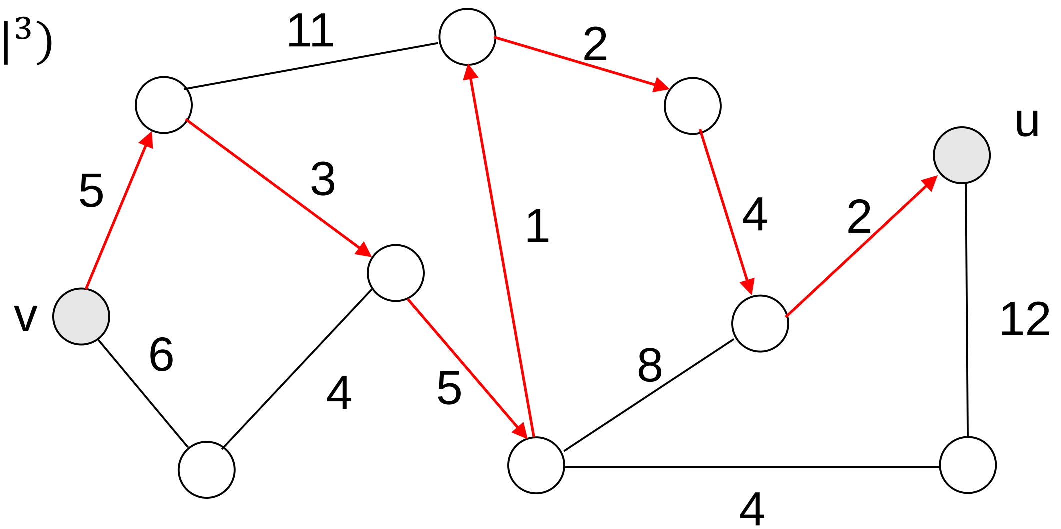
- Dijkstra; Laufzeit $\mathbf{O}\big((|V| + |E|)\log|V|\big)$

SSSP (allgemeine Kantengewichte)

- Bellman-Ford; Laufzeit $\mathbf{O}(|V|^2 + |V||E|)$

APSP (allgemeine Kantengewichte, keine negativen Zyklen)

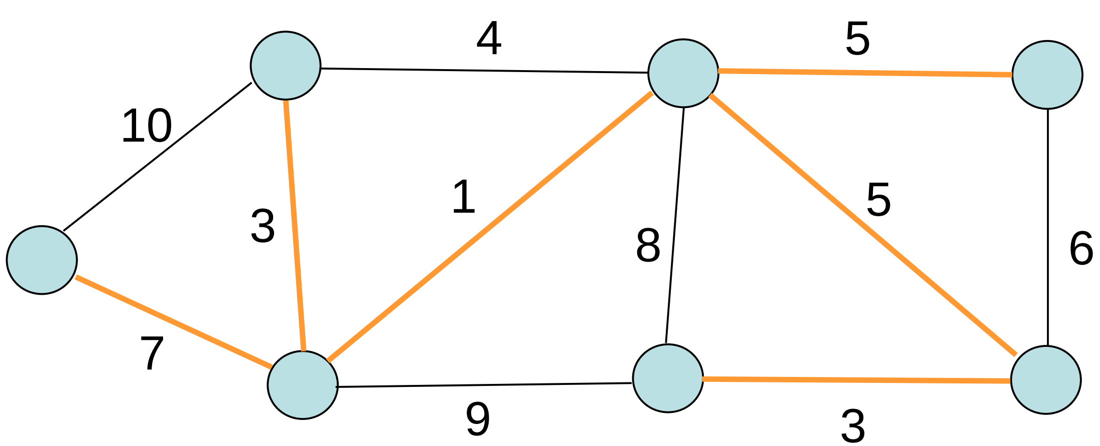
- Floyd-Warshall; Laufzeit $\mathbf{O}(|V|^3)$



Graphalgorithmen

Minimale Spannbäume

- Gegeben: Gewichteter, ungerichteter, zusammenhängender Graph $G = (V, E)$
- Gesucht: Ein aufspannender Baum mit minimalem Gewicht
- Aufspannender Baum: Inklusionsmaximaler kreisfreier Teilgraph mit Knotenmenge V



Approximationsalgorithmen

Was kann man tun, wenn man Problem nicht effizient lösen kann?

- Die Aufgabenstellung vereinfachen!

Approximationsalgorithmen

- Löse Problem nicht exakt, sondern nur approximativ
- Qualitätsgarantie in Abhängigkeit von optimaler Lösung
- Z.B.: jede berechnete Lösung ist nur doppelt so teuer, wie eine optimale Lösung

Beispiele

- Vertex Cover und TSP
- Last Balancierung und k-Center Clustering

Viel Erfolg bei der Klausur und schöne Semesterferien!