Algorithmen 1 SS 2013 – Tutorium 7

9. Tutorium

Sarah Lutteropp

18. Juni 2013

Übersicht

1. Korrekturen
2. (a,b)-Bäume
3. Exkurs: AVL-Bäume
4. Graphenrepräsentation
5. Graphtraversierung
   - Topologisches Sortieren
   - Tiefensuche
   - Breitensuche
   - BFS vs. DFS
   - Exkurs: Iterative Tiefensuche
6. Labyrinth
7. Kreativaufgabe

Probeklausur

- Lest die Aufgabenstellungen genauer
- Auf **ALLE** Blätter Name + Matrikelnummer schreiben
- Hinweis auf Rückseite

7. und 8. Übungsblatt

Keine besonderen Anmerkungen.

Motivation (a,b)-Baum

Szenario

Datenmenge ist so groß, dass sie auf der Festplatte gespeichert werden muss.

Anforderung

Wir wollen die Daten so auf der Festplatte organisieren, dass möglichst wenig Paging[1] notwendig wird. (I/O-effiziente Datenstrukturen)

[1] siehe TI

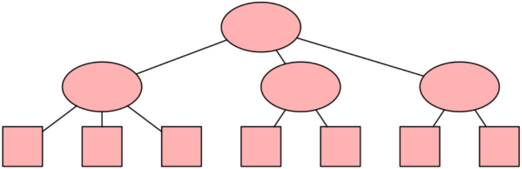
(a,b)-Bäume

Seien $a, b \in \mathbb{N}, a \geq 2, b \geq 2a - 1$.

Definition (a,b)-Baum

Ein Baum heißt (a,b)-Baum, falls gilt:

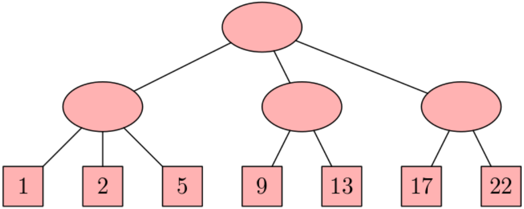
- Jeder innere Knoten hat mindestens a und höchstens b Kinder.
- Alle Blätter haben die gleiche Tiefe.
- Jeder Knoten mit m Kindern enthält genau $m - 1$ Schlüssel.



Ein (2,3)-Baum

(a,b)-Bäume als assoziatives Array

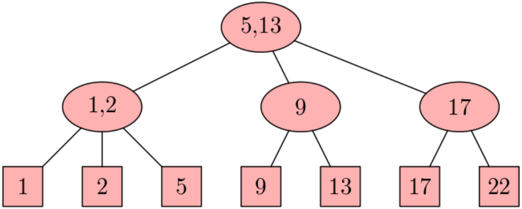
Wir speichern Schlüssel und Datenelemente nur in den Blättern:



Die Schlüssel sind von links nach rechts geordnet.

(a,b)-Bäume als assoziatives Array

Als Suchhilfe erhält ein innerer Knoten mit m Kindern genau $m-1$ Hilfsschlüssel. (Diese sind innerhalb des Knotens sortiert.)



Jetzt kann effizient nach einem Element gesucht werden (wie bei binärem Suchbaum, nur jetzt mit Mehrwege-Entscheidung).

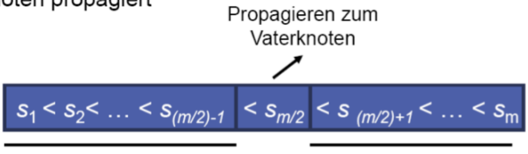
(a,b)-Bäume – Einfügen

Die Blätter enthalten die Schlüssel in aufsteigender Reihenfolge.

- Neues Blatt an richtiger Stelle einfügen
- Problem, falls Elternknoten mehr als b Kinder hat (Überlauf)
- $\Rightarrow$ In zwei Knoten mit $\lfloor (b+1)/2 \rfloor$ und $\lceil (b+1)/2 \rceil$ Kindern teilen
- Jetzt kann Vater überfüllt sein etc.

Bei Uberlauf eines Knotens → Split

- Einträge des Knotens werden auf zwei Knoten verteilt
- Das Objekt, das in der Mitte des übergelaufenen Knotens liegt, wird zum Vaterknoten propagiert



(a,b)-Bäume – Löschen

Die Blätter enthalten die Schlüssel in aufsteigender Reihenfolge.

- Blatt mit Schlüssel entfernen
- Problem, falls Elternknotenweniger als *a* Kinder hat (Unterlauf)
- $\Rightarrow$ Mit einem Geschwisterknoten vereinigen
- Dieser muss vielleicht wieder geteilt werden
- Der nächste Elternknoten kann nun wieder unterbelegt sein

B-Bäume und Datenbanken

Ein B-Baum ist ein $(m, 2m)$-Baum.
Wir wählen m so groß, dass ein Knoten soviel Platz wie eine Seite im Hintergrundspeicher benötigt (z.B. 4096 Byte).
Blätter eines Elternknotens gemeinsam speichern.
Zugriffszeit:
Nur $\mathcal{O}(\log_m(n))$ Zugriffe auf den Hintergrundspeicher.
Wurzel kann immer im RAM gehalten werden.
Falls $m \approx 500$, dann enthält ein B-Baum der Höhe 3 bereits mindestens $500 \cdot 500 \cdot 500 = 125000000$ Schlüssel und Datenelemente.
Jede Suche greift auf nur zwei Seiten zu!

Exkurs: AVL-Bäume

Kommt nächstes Tut.

Graphenrepräsentation

Welche Arten der Graphenrepräsentation kennt ihr?

Graphenrepräsentation

Welche Arten der Graphenrepräsentation kennt ihr?

- Adjazenzmatrix

Graphenrepräsentation

Welche Arten der Graphenrepräsentation kennt ihr?

- Adjazenzmatrix
- Adjazenzliste

Graphenrepräsentation

Welche Arten der Graphenrepräsentation kennt ihr?

- Adjazenzmatrix
- Adjazenzliste
- Adjazenzfeld

Graphenrepräsentation

Welche Arten der Graphenrepräsentation kennt ihr?

- Adjazenzmatrix
- Adjazenzliste
- Adjazenzfeld
- Inzidenzmatrix

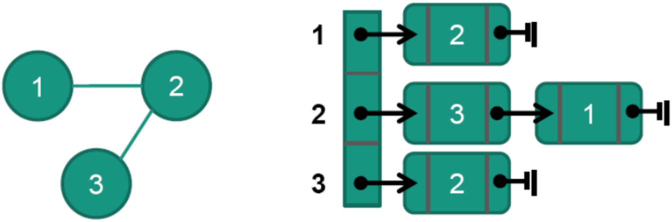
Graphenrepräsentation

Welche Arten der Graphenrepräsentation kennt ihr?

- Adjazenzmatrix
- Adjazenzliste
- Adjazenzfeld
- Inzidenzmatrix
- Inzidenzliste

Graphenrepräsentation – Adjazenzliste

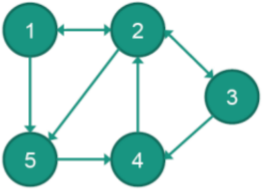
- Feld *Adj* der Länge $|V|$
- $Adj[u]$ enthält Liste mit Knoten v für die eine Kante $(u, v) \in E$ existiert



Graphenrepräsentation – Adjazenzmatrix

- Adjazenzmatrizenrepräsentation eines Graphen $G = (V, E)$
  - Adjazenzmatrix ist eine $|V| \times |V|$ Matrix $A = (a_{ij})$ so, dass
  - $a_{ij} = \begin{cases} 1, & \text{wenn } (i,j) \in E \\ 0, & \text{sonst} \end{cases}$

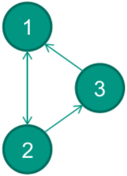
- Beispiel: Gerichteter Graph



	1	2	3	4	5
1	0	1	0	0	1
2	1	0	1	0	1
3	0	1	0	1	0
4	0	1	0	0	0
5	0	0	0	1	0

Graphenrepräsentation – Adjazenzfeld

$V[n]$ speichert den Index in E, ab dem die ausgehenden Kanten von Knoten n in E aufgelistet sind



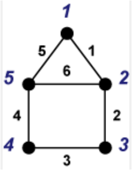
V

Index	1	2	3
Wert	1	2	4

E

Index	1	2	3	4
Wert	2	1	3	1

Graphenrepräsentation – Exkurs: Inzidenz-{liste, matrix}



Inzidenzliste

$e_1(v_1, v_2)$ $e_2(v_2, v_3)$ $e_3(v_3, v_4)$ $e_4(v_4, v_5)$ $e_5(v_1, v_5)$ $e_6(v_2, v_5)$

Inzidenzmatrix

$$\begin{pmatrix} e_1 & e_2 & e_3 & e_4 & e_5 & e_6 & \\ 1 & 0 & 0 & 0 & 1 & 0 & v_1 \\ 1 & 1 & 0 & 0 & 0 & 1 & v_2 \\ 0 & 1 & 1 & 0 & 0 & 0 & v_3 \\ 0 & 0 & 1 & 1 & 0 & 0 & v_4 \\ 0 & 0 & 0 & 1 & 1 & 1 & v_5 \end{pmatrix}$$

Topologisches Sortieren

Topologische Sortierung eines Graphen $G = (V, E)$

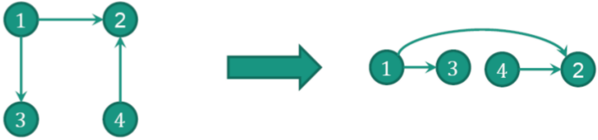
Ordnung aller Knoten in G, so dass $u < v$, falls $(u, v) \in E$.

Topologisches Sortieren

Topologische Sortierung eines Graphen $G = (V, E)$

Ordnung aller Knoten in G, so dass $u < v$, falls $(u, v) \in E$.
Falls G Zyklen enthält, ist keine Ordnung möglich.

- Anschaulich werden alle Knoten auf einer Linie angeordnet
  - Alle Kanten gehen von links nach rechts



Aufgabe

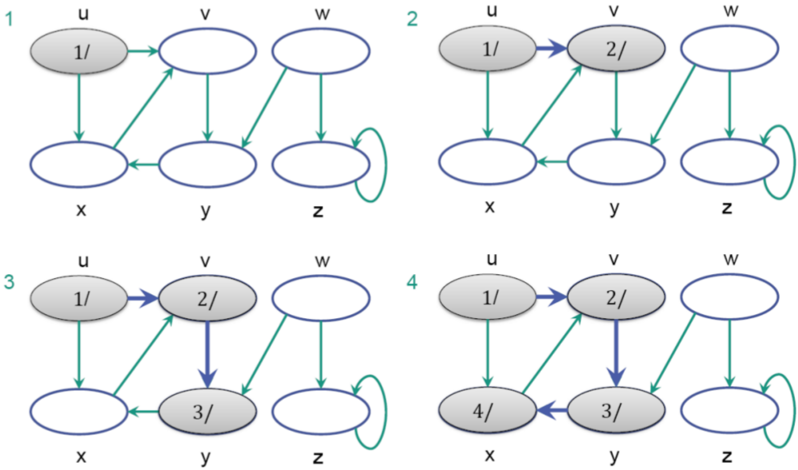
Sortiere den folgenden Graphen topologisch

Tiefensuche – Grundlagen

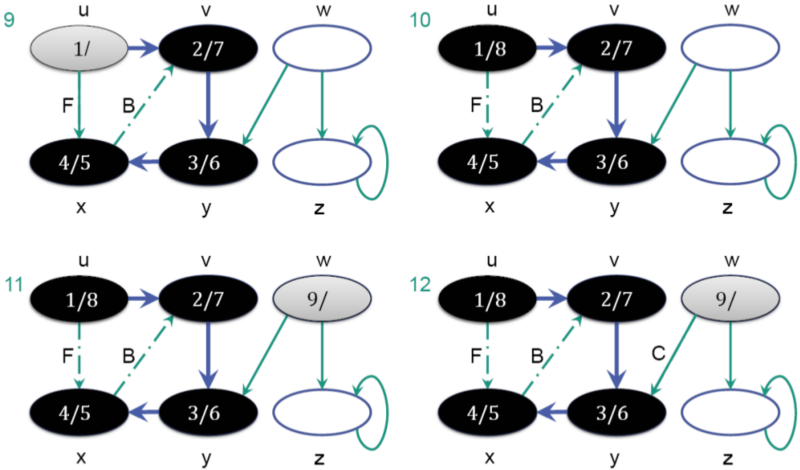
- Weiß = unbesuchter Knoten
- Grau = entdeckter Knoten 1/ (Mit discovered-Timestamp „1“)
- Schwarz = finalisierter Knoten 1/7 (Mit finalized-Timestamp „7“)

- Baumkante
  - Zielknoten ist weiß
- Rückwärtskante B
  - Zielknoten ist ein Vorgänger (entlang der bisherigen Baumkanten)
  - Schleifen
- Vorwärtskante F
  - Zielknoten ist ein Nachfolger (entlang der bisherigen Baumkanten)
- Querkanten C
  - Sonstige Kanten

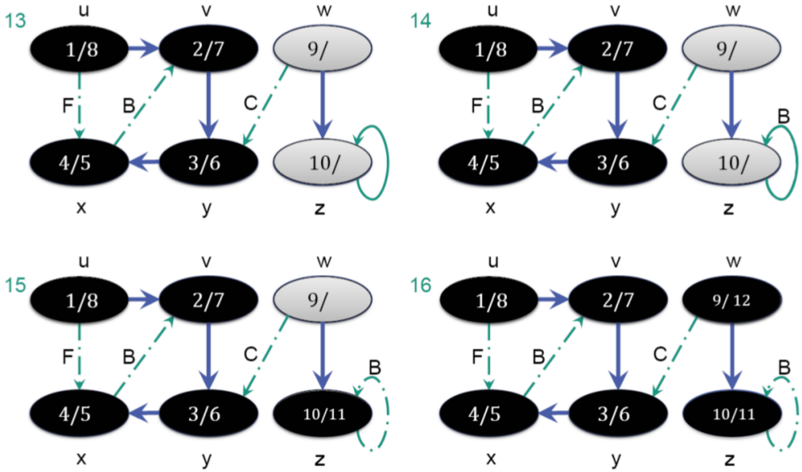
Tiefensuche – Beispiel



Tiefensuche – Beispiel

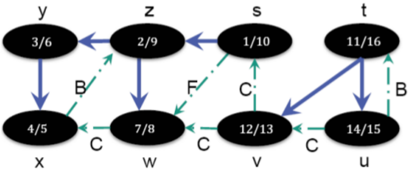


Tiefensuche – Beispiel

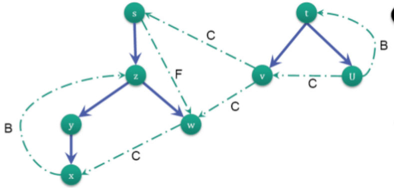


Tiefensuche – Klammerausdruck

- Timestamps haben Klammerstruktur
  - Entdecken einen Knotens u wird mit „(u“ repräsentiert
  - Abschließen einen Knotens u wird mit „u)“ repräsentiert
- Beispiel
- Baumdarstellung (unten)

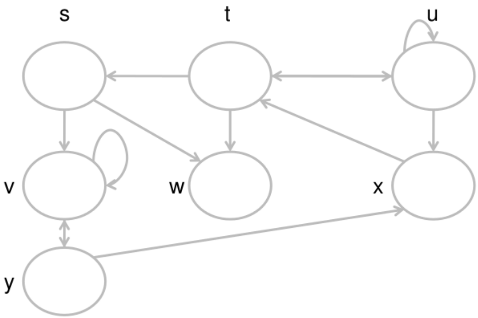


1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
(s	(z	(y	(x	x)	y)	(w	w)	z)	s)	(t	(v	v)	(u	u)	t)

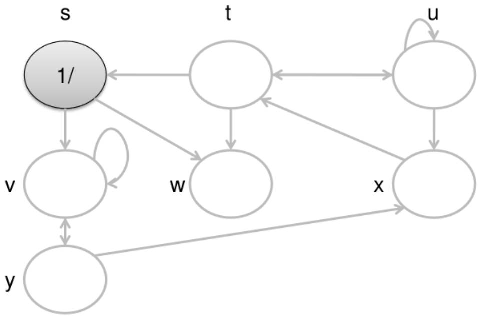


Aufgabe

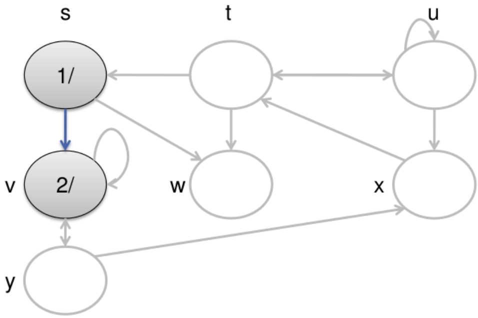
Tiefensuche, starte bei s



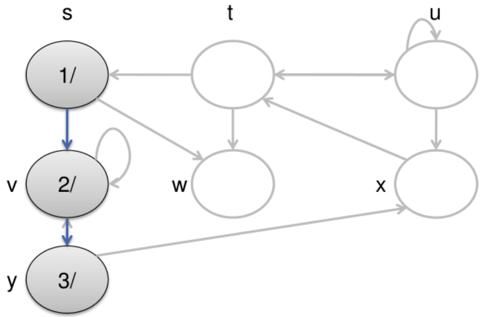
Lösung



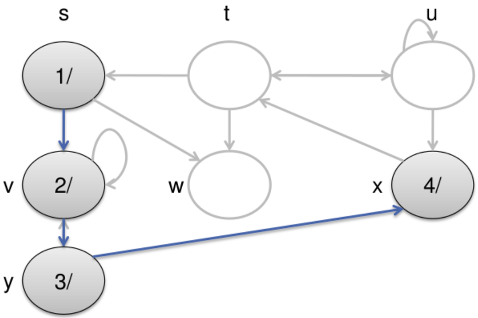
Lösung



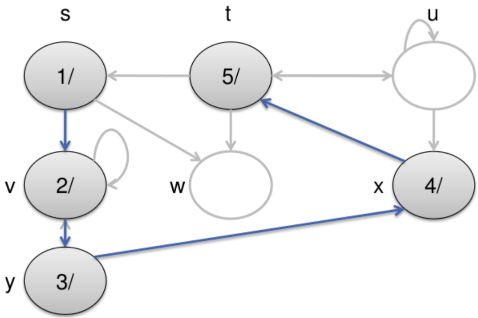
Lösung



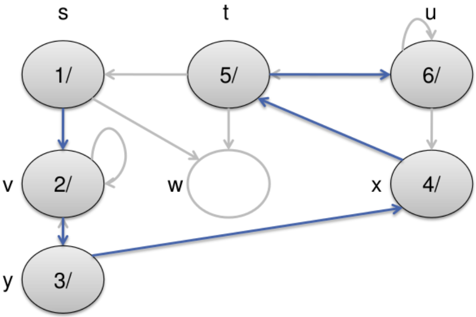
Lösung



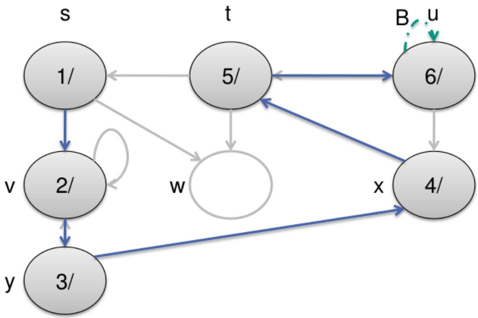
Lösung



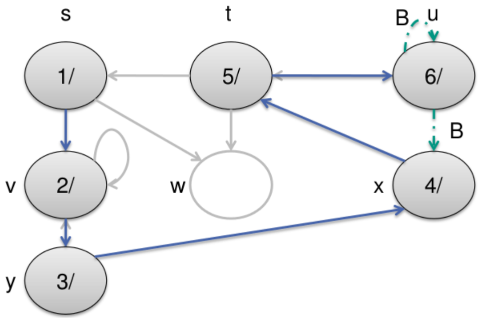
Lösung



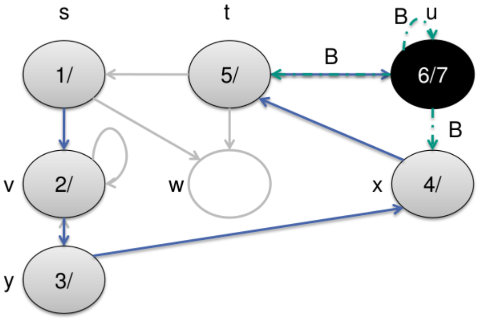
Lösung



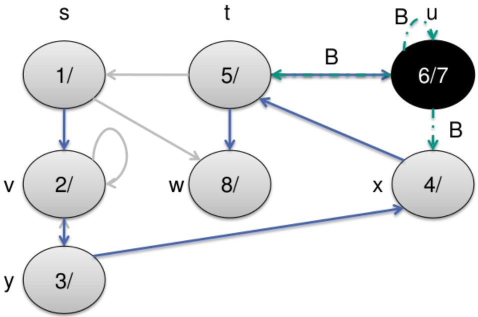
Lösung



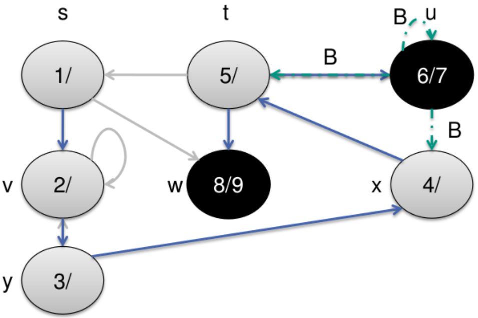
Lösung



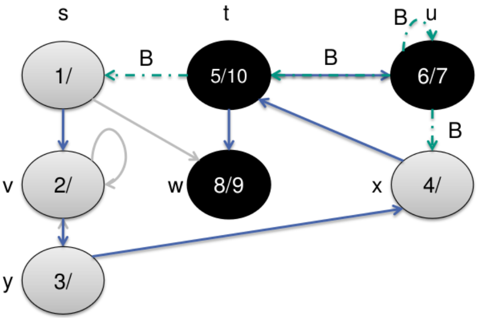
Lösung



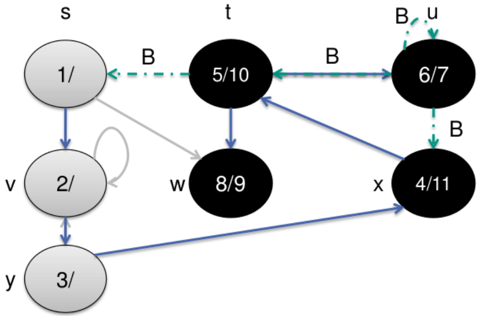
Lösung



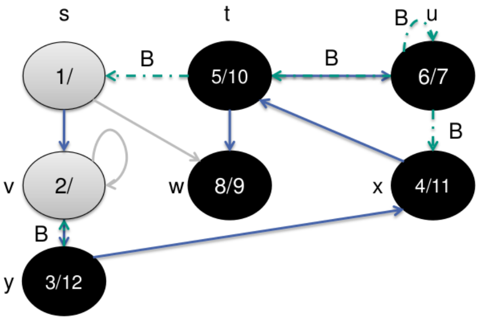
Lösung



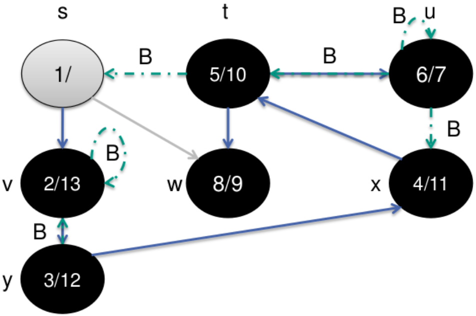
Lösung



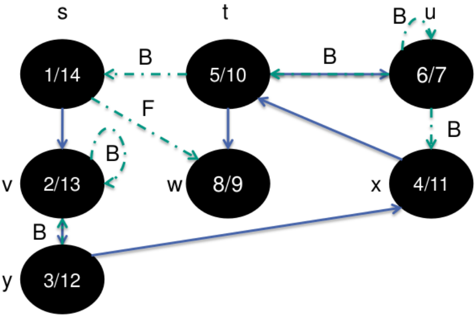
Lösung



Lösung

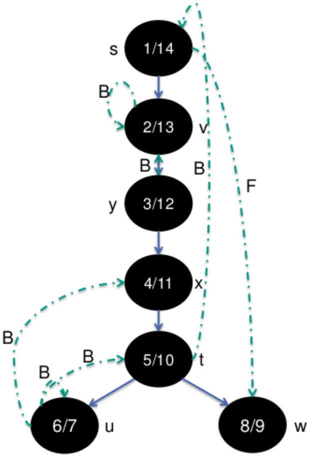


Lösung

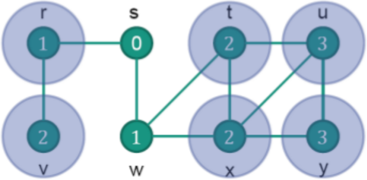


Extraaufgabe: Zeichne nun den resultierenden Baum.

Lösung



Breitensuche

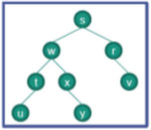


	Q	s				
s ←	Q	w	r			
w ←	Q	r	t	x		
r ←	Q	t	x	v		
t ←	Q	x	v	u		
x ←	Q	v	u	y		
v ←	Q	u	y			
u ←	Q	y				
y ←	Q					

$visited$ = false, $distance = \infty$ ∞

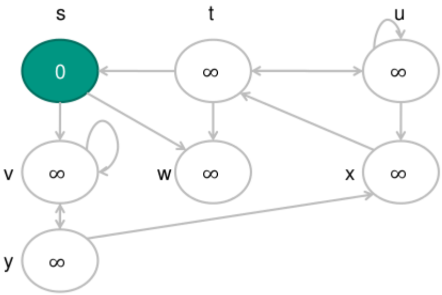
$visited$ = true, $distance = 0$ 0

Breadth-First-Tree



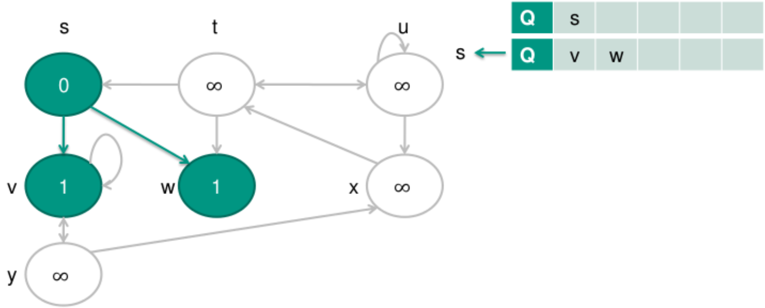
Aufgabe

- Übungsaufgabe
  - Wende Breitensuche bei folgendem Graphen an
  - Beginne bei „s" und notiere die Queue sowie die Distanzen in jedem Schritt

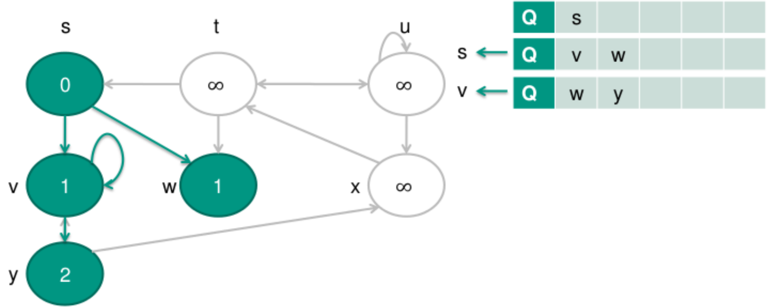


Q	s				

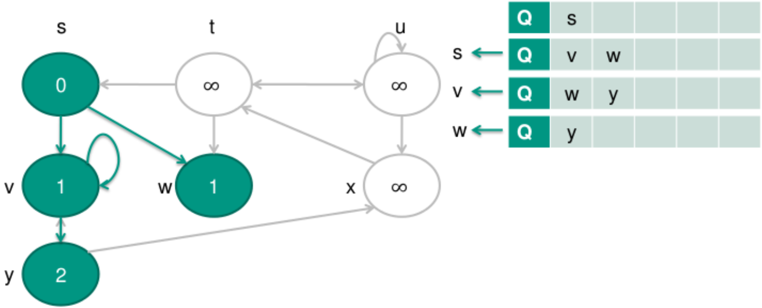
Lösung



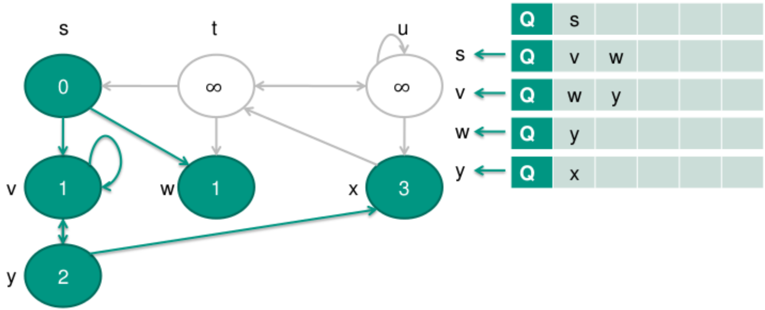
Lösung



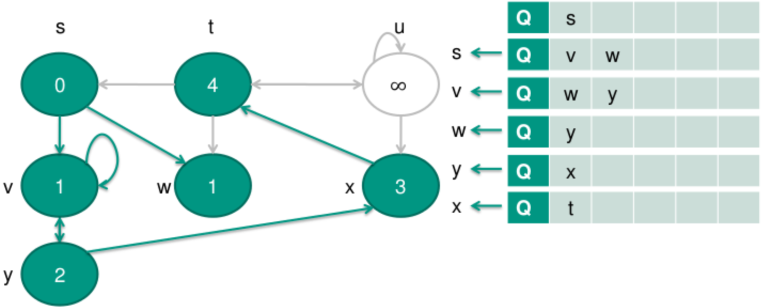
Lösung



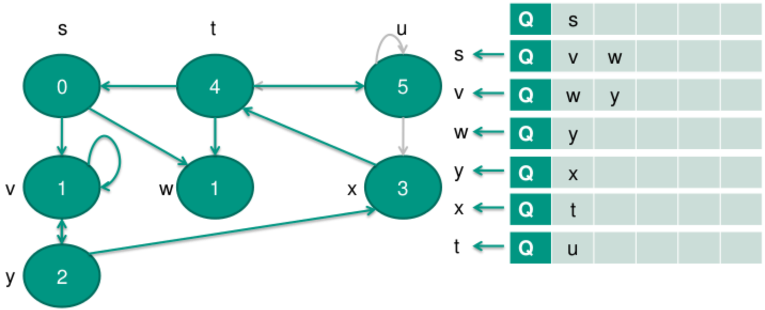
Lösung



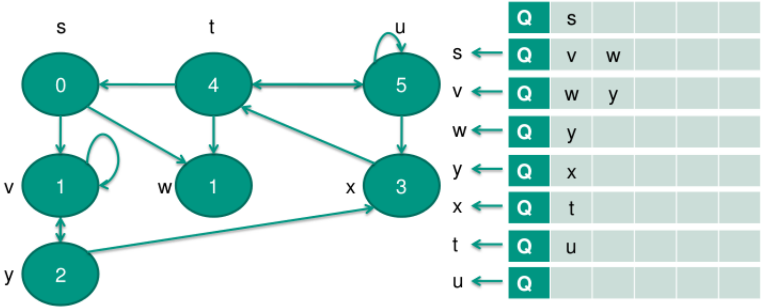
Lösung



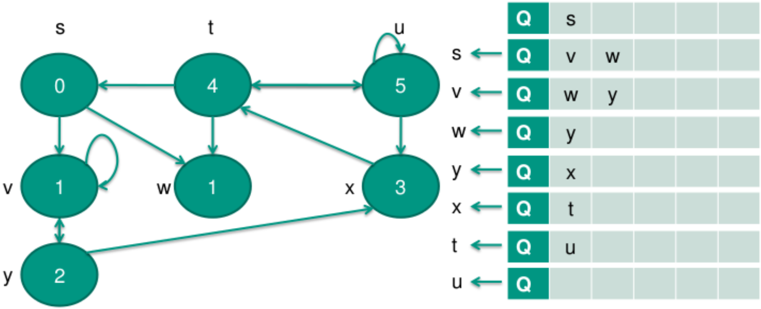
Lösung



Lösung

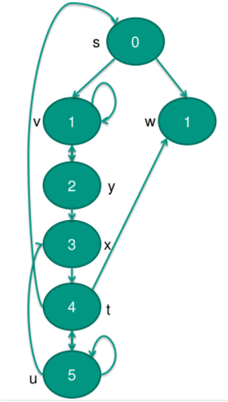


Lösung



Wie sieht der zugehörige Baum aus?

Lösung



Vergleich Tiefensuche und Breitensuche

Vorteile Breitensuche

- keine Rekursion nötig
- Vollständigkeit: Falls Lösung existiert, wird diese gefunden (auch bei einem unendlichen Graphen, nur endlich viele Alternativen pro Knoten)
- Optimalität: Im Allgemeinen optimal, da immer das Ergebnis mit dem kürzesten Pfad zum Anfangsknoten gefunden wird

Vorteile Tiefensuche

- Geringer Speicherverbrauch, da keine Warteschlange nötig

Vergleich Tiefensuche und Breitensuche

Gleiche Laufzeit

- Graph als Adjazenzliste: $\mathcal{O}(|V| + |E|)$
- Graph als Adjazenzmatrix: $\mathcal{O}(|V|^2)$

Denkaufgabe

Kleiner Trick

Was passiert, wenn man eine Breitensuche codet und dann die Queue durch einen Stack ersetzt? ;-)

Exkurs: Iterative Tiefensuche

Kombiniert geringen Speicherverbrauch von Tiefensuche mit Optimalität von Breitensuche

Idee

Rufe wiederholt beschränkte Tiefensuche auf, erhöhe Suchtiefe immer um 1 $\rightsquigarrow$ Dadurch kein "Verlaufen" in unendlich langen Pfaden mehr möglich

Aufgabe

Hier stand mal eine nette Aufgabe. Allerdings wurde sie (heute um 01:53 Uhr) auf nächstes Tut verschoben.

Kreativaufgabe

Dynamisiertes Adjazenzfeld

Gesucht ist eine Datenstruktur für gerichtete Graphen $G = (V, E)$, mit folgenden Eigenschaften:

- Stabile und eindeutige KnotenIDs
- Eindeutige KantenIDs
- Effizienter Wahlfreier Zugriff auf Knoten und Kanten
- Effiziente Navigation
- Amortisiert konstantes Einfügen von Knoten und Kanten
- Amortisiert konstantes Entfernen von Knoten und Kanten

Kreativaufgabe

Stabile und eindeutige KnotenIDs

Knoten sollen durch IDs eindeutig identifiziert werden. Diese IDs sollen Zahlen aus $\mathbb{N}_{\geq 0}$ sein. Dabei seien die KnotenIDs stabil, d.h. die ID eines Knotens ändere sich nie solange dieser Knoten existiert (nach Entfernen eines Knotens darf dessen ID jedoch neu vergeben werden).

Kreativaufgabe

Eindeutige KantenIDs

Die Kanten sollen ebenfalls durch IDs eindeutig identifiziert werden. Allerdings müssen diese nicht unbedingt Zahlen aus $\mathbb{N}_{\geq 0}$ sein und sie müssen auch nicht stabil sein.

Kreativaufgabe

Effizienter Wahlfreier Zugriff auf Knoten und Kanten

Es gibt die Operationen

- *node(u : NodeID) : Handle of Node* und
- *edge(e : EdgeID) : Handle of Edge*,

die in $\mathcal{O}(1)$ Zeit einen Handle auf das Knoten bzw. Kantenobjekt zu einer Knoten- bzw. Kanten-ID liefern.

Kreativaufgabe

Effiziente Navigation

Es gibt die Operationen

- *firstEdge(v : NodeID) : EdgeID* $\cup\{\perp\}$ und
- *nextEdge(e : EdgeID) : EdgeID* $\{\cup\perp\}$,

mit deren Hilfe wie folgt über alle ausgehenden Kanten eines Knoten *v* iteriert werden kann in einem Graph *G*:

for ($EdgeID\ \ e := graph.firstEdge(v)\ ;\ e \neq \perp\ ;\ e := nextEdge(e)$)
  $h_e := G.edge(e) : Handle$ **of** $Edge$
  $/*\ do\ something\ */$
end for

Sowohl *firstEdge* als auch *nextEdge* dürfen höchstens $\mathcal{O}(1)$ Zeit

Kreativaufgabe

Amortisiert konstantes Einfügen von Knoten und Kanten

Es gibt die Operationen

- *insertNode : NodeID* und
- *insertEdge(u, v : NodeID) : EdgeID*,

die in amortisiert konstanter Zeit einen neuen Knoten bzw. eine neue Kanten von u nach v einfügen und jeweils die ID des neu erzeugten Elementes zurückliefern. Beide Operationen dürfen höchstens *amortisiert* konstante Zeit kosten.

Kreativaufgabe

Amortisiert konstantes Entfernen von Knoten und Kanten

Es gibt die Operationen

- *deleteNode(v : NodeID)* und
- *deleteEdge(e : EdgeID)*,

die einen Knoten bzw eine Kante entfernen. Der Einfachheit halber darf ein Knoten dabei nur entfernt werden, wenn bereits alle seine Kanten entfernt worden sind. Beide Operationen dürfen höchstens *amortisiert* konstante Zeit kosten.

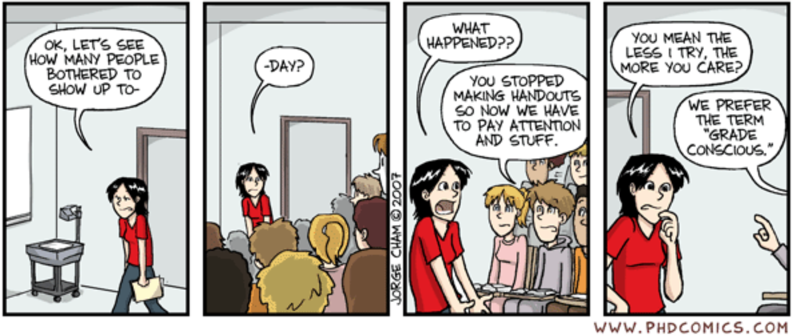
Kreativaufgabe

Dynamisiertes Adjazenzfeld

Gesucht ist eine Datenstruktur für gerichtete Graphen $G = (V, E)$, mit obigen Eigenschaften.

1. Überlegen Sie sich, wie Sie diese Datenstruktur realisieren.
2. Begründen Sie, warum die beschriebenen Operationen in Ihrer Realisierung das geforderte Laufzeitverhalten aufweisen.
3. Wieviel Speicher kann ein Graph mit Ihrer Realisierung im schlimmsten Fall belegen (abhängig von aktuellen oder zwischenzeitlichen Werten von $|V|$ und $|E|$ und das nicht nur im O-Kalkül)? Wieviel im besten Fall? Vergleichen Sie mit dem Speicherverbrauch des statischen Adjazenzfeldes aus der Vorlesung.

Bis zum nächsten Mal.



Disclaimer: Folien zu (a,b)-Bäumen zusammengeklaut aus

- `http://tcs.rwth-aachen.de/lehre/DA/SS2011/handout-2011-05-03.pdf`
- `http://www2.cs.uni-paderborn.de/cs/ag-madh/vorl/DaStrAlg01/folien/DA_6.pdf`

Disclaimer 2: Folien zu Tiefensuche/Breitensuche teils zusammengeklaut aus

- Benjamin Brandmüller, Algorithmen I Tutorium, 09.06.2011