Technische Universität Darmstadt
Fachgebiet Theoretische Informatik

Prof. Johannes Buchmann
Nabil Alkeilani Alkadri
Nina Bindel
Patrick Struck

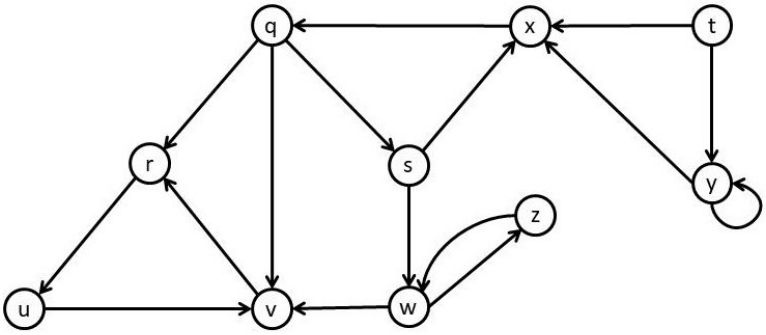
Algorithmen und Datenstrukturen

SoSe 2018

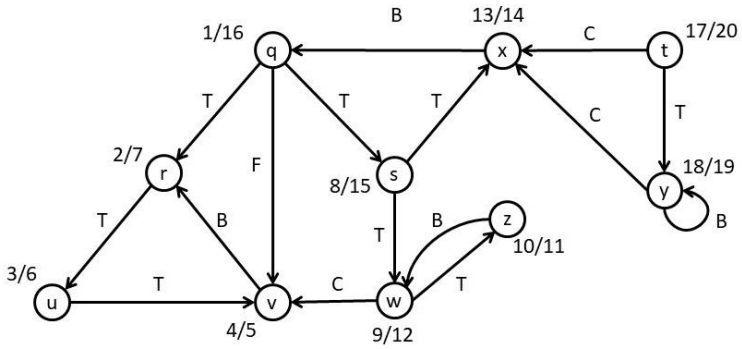
9. Lösungsblatt — 11.06.2018
v1.0

P1 Tiefensuche

Wenden Sie die Tiefensuche auf den folgenden Graphen an. Beginnen Sie bei dem Knoten q und gehen Sie davon aus, dass die Adjazenzlisten alphabetisch sortiert sind. Geben Sie für jeden Knoten die *discovery time* (Entdeckungszeit) und die *finishing time* (Abschlusszeit) an. Geben Sie außerdem bei jeder Kante an, ob es sich um eine *Baumkante* (T), *Vorwärtskante* (F), *Rückwärtskante* (B) oder *Querkante* (C) handelt.



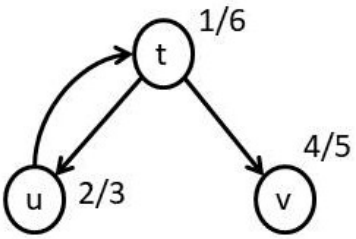
Lösung.



P2 Waldstruktur der Tiefensuche

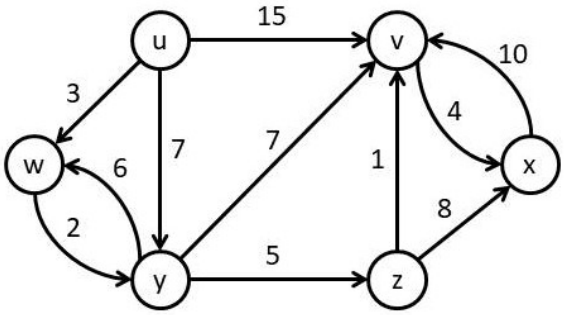
Zeigen Sie oder widerlegen Sie folgende Aussage: “Wenn ein gerichteter Graph G einen Pfad von u nach v enthält und in einer Tiefensuche $d[u] < d[v]$ gilt, dann ist v im Tiefensuchwald ein Nachfahre von u.”

Lösung. Wir geben ein Gegenbeispiel an, welches die Aussage widerlegt:
Bei dem folgenden gerichteten Graphen existiert ein Pfad von u nach v und bei einer Tiefensuche, die beim Knoten t beginnt, ist $d[u] < d[v]$. Allerdings ist v im entstehenden Baum kein Nachfahre von u.



P3 Dijkstra

Führen Sie den Algorithmus von Dijkstra auf folgendendem Graphen aus. Nutzen Sie als Startknoten den Knoten u. Tragen Sie in der Tabelle für jeden Schleifendurchlauf die Werte d, π und S ein.



d[u]	d[v]	d[w]	d[x]	d[y]	d[z]	π[u]	π[v]	π[w]	π[x]	π[y]	π[z]	S
0	∞	∞	∞	∞	∞	*nil*	*nil*	*nil*	*nil*	*nil*	*nil*	$\emptyset$

Lösung.

d[u]	d[v]	d[w]	d[x]	d[y]	d[z]	π[u]	π[v]	π[w]	π[x]	π[y]	π[z]	S
0	∞	∞	∞	∞	∞	*nil*	*nil*	*nil*	*nil*	*nil*	*nil*	$\emptyset$
0	15	3	∞	7	∞	*nil*	u	u	*nil*	u	*nil*	$\{u\}$
0	15	3	∞	5	∞	*nil*	u	u	*nil*	w	*nil*	$\{u,w\}$
0	12	3	∞	5	10	*nil*	y	u	*nil*	w	y	$\{u,w,y\}$
0	11	3	18	5	10	*nil*	z	u	z	w	y	$\{u,w,y,z\}$
0	11	3	15	5	10	*nil*	z	u	v	w	y	$\{u,w,y,z,v\}$
0	11	3	15	5	10	*nil*	z	u	v	w	y	$\{u,w,y,z,v,x\}$

P4 MIN-PRIORITY-QUEUE

Um effizient den Knoten mit der geringsten Distanz zur Abarbeitung auswählen zu können, verwendet der Algorithmus von Dijkstra eine MIN-PRIORITY-QUEUE. Diese Datenstruktur unterstützt die folgenden Operationen:

- INSERT(S, x) fügt das Element x der Menge S hinzu, was äquivalent zu $S = S \cup x$ ist.
- MINIMUM(S) gibt das Element mit dem kleinsten Schlüssel aus S zurück.
- EXTRACT-MIN(S) entfernt das Element mit dem kleinsten Schlüssel aus S und gibt es zurück.
- DECREASE-KEY(S, x, k) verringert den Wert des Schlüssels vom Element x auf den Wert k, welcher höchstens so groß wie der alte Wert sein darf.

Erweitern Sie die Datenstruktur MIN-HEAP so, dass sie als MIN-PRIORITY-QUEUE verwendet werden kann. Implementieren Sie dazu in Pseudocode die Funktion HEAP-DECREASE-KEY(S, i, k) wobei i der Index des Elements x ist, dessen Schlüsselwert verringert werden soll.

Lösung.

```
HEAP-DECREASE-KEY(S, i, k)
1 if k > S[i]
2     error "new key is bigger than current key"
3 S[i] = k
4 while i > 1 and S[PARENT(i)] > S[i]
5     tmp = S[i]
6     S[i] = S[PARENT(i)]
7     S[PARENT(i)] = tmp
8     i = PARENT(i)
```

H1 Kantentypen

Zeigen Sie, dass bei einer Tiefensuche auf einem ungerichteten Graphen $G = (V, E)$ jede Kante $(u, v) \in E$ entweder eine *Baum-* oder *Rückwärtskante* ist.

Lösung. Sei $(u, v) \in E$. Wir nehmen ohne Beschränkung der Allgemeinheit an, dass $d[u] < d[v]$ gilt (Falls $d[u] > d[v]$ gilt, kann man u und v tauschen, da der Graph ungerichtet ist). Da v in der Adjazenzliste von u enthalten ist, ist $d[v] < f[u]$ und aufgrund der Klammerstruktur muss deshalb $f[v] < f[u]$ gelten. Deshalb ist u grau, wenn v beendet wird. Die Kante (u, v) kann entweder in Richtung u nach v oder in Richtung v nach u erkundet werden. Falls sie in Richtung u nach v erkundet wird, ist v zum Zeitpunkt der Erkundung noch weiß, da die Kante ansonsten bereits in die andere Richtung erkundet worden wäre. Somit handelt es sich in diesem Fall um eine *Baumkante*. Falls die Kante in Richtung v nach u erkundet wird, ist zu diesem Zeitpunkt v grau gefärbt. Da auch u grau gefärbt ist, handelt es sich in diesem Fall um eine Rückwärtskante.

H2 Negative Kantengewichte

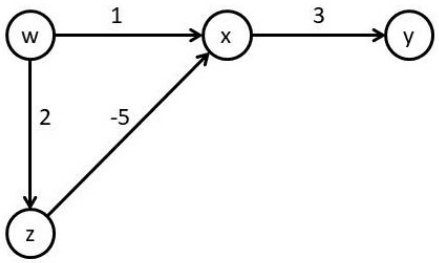
(a) Geben Sie ein Beispiel für einen Graphen mit negativen Kantengewichten an, so dass der Algorithmus von Dijkstra nicht das richtige Ergebnis liefert.

(b) Argumentieren Sie, warum der Algorithmus von Dijkstra trotz negativer Kantengewichte korrekt funktioniert, wenn ausschließlich vom Startknoten ausgehende Kanten negative Gewichte besitzen und keine negativen Zyklen im Graphen existieren.

(c) Ist das Problem des kürzesten Pfades wohldefiniert, wenn der Graph negative Zyklen enthält, die vom Startknoten s aus erreichbar sind? Begründen Sie Ihre Antwort.

Lösung.

(a) Bei folgendem Graph kommt der Algorithmus von Dijkstra mit dem Startknoten w zu einem falschen Ergebnis. Zuerst wird der Knoten w, dann x und dann z abgearbeitet. Dabei ergibt sich beim Abarbeiten von x die Distanz 4 für y. Wenn nun x beim Abarbeiten von z die Distanz -3 erhält, müsste infolge dessen die Distanz von y zu 0 schrumpfen. Dies geschieht jedoch nie, da x bereits abgearbeitet ist.



(b) Wenn nur ausgehende Kanten vom Startknoten negativ sind, werden nach der Abarbeitung des Startknotens keine negativen Kanten beim Abarbeiten der anderen Knoten mehr aufgefunden. Da keine negativen Zyklen existieren, kann die Distanz des Startknoten nicht kleiner als 0 werden. Da beim Abarbeiten der restlichen Knoten keine negativen Kanten mehr aufgefunden werden, kann es auch dort nicht vorkommen, dass sich die Distanz zu einem Knoten nach dem Abarbeiten noch verringert. Denn es wird immer von den nicht abgearbeiteten Knoten derjenige abgearbeitet, der die geringste Distanz hat. Mit den positiven Gewichten kann beim Abarbeiten kein anderer Knoten eine kleinere Distanz als die des aktuellen Knotens, und somit auch nicht kleiner als die, eines abgearbeiteten Knotens, erhalten. Somit ist es bei keinem Knoten möglich, dass sich die Distanz nach dem Abarbeiten noch verkleinern kann, womit die Annahme des Algorithmus erfüllt ist und er folglich korrekt arbeitet.

(c) Das Problem des kürzesten Pfades ist nicht wohldefiniert, wenn der Graph negative Zyklen enthält, die vom Startknoten s aus erreichbar sind. Kein Pfad vom Knoten s zu einem der Knoten in dem Zyklus kann ein kürzester Pfad sein. Es ist immer möglich einen kürzeren Pfad zu konstruieren, indem man den Zyklus einmal an den Pfad anfügt.