Name: Matrikelnummer:

Klausur Algorithmen I, 27.07.2010 Blatt 1 von 1

Lösungsvorschlag

Aufgabe 1. Kleinaufgaben [19 Punkte]

a. Geben Sie eine Familie von DAGs an, in der alle DAGs $\Omega(n^2)$ Kanten haben bei n Knoten. [2 Punkte]

Lösung

Für jedes $n \in \mathbb{N}_{>0}$ ein Graph $G_n = (V_n, E_n)$ mit $V_n := \{1, \ldots, n\}$. Für $u, v \in V_n$ sei $(u,v) \in E_n$ genau dann, wenn $u < v$.

Lösungsende

b. Angenommen, Sie wollen eine Folge von n ganzen Zahlen sortieren. Nennen Sie einen vergleichsbasierten Sortieralgorithmus, der im O-Kalkül optimales Worst-Case-Laufzeitverhalten aufweist. Würden Sie diesen Algorithmus auch in der Praxis verwenden? Begründen Sie kurz. [2 Punkte]

Lösung

Mögliche Lösungen sind z.B.:

- Mergesort. In der Praxis erwartet man von Quicksort jedoch ein etwas besseres Laufzeitverhalten.
- Heapsort. Das ist auch in der Praxis sinnvoll, weil Heapsort höchstens $O(1)$ zusätzlichen Speicher benötigt.
- Mergesort. In der Praxis hat Mergesort den Vorteil, dass es stabil ist.

Bemerkung: Im O-Kalkül ist die bestmögliche Worst-Case-Laufzeit eines vergleichsbasierten Sortierverfahrens in $O(n \log n)$. Entsprechend sind Mergesort und Heapsort richtige Antworten, Quicksort jedoch nicht (da es bis zu $O(n^2)$ Zeit braucht). Bucketsort ist nicht vergleichsbasiert und daher auch keine richtige Lösung.

Lösungsende

c. Lösen Sie die beiden folgenden Rekurrenzen im Θ-Kalkül:

$$\begin{aligned} T(n) &= 17n + 2T(n/3), \quad T(1) = 3 \\ S(n) &= 3n + 9S(n/3), \quad S(1) = 23 \end{aligned}$$

mit $n = 3^k$ und $k \in \mathbb{N}_{>0}$. [2 Punkte]

Lösung

$T(n) = \Theta(n)$ und $S(n) = \Theta(n^{\log_3 9}) = \Theta(n^2)$.

Lösungsende

d. Zu einem gerichteten Graph $G = (V,E)$ definieren wir $\overline{G} := (V, \overline{E})$ mit

$$\overline{E} := \left\{ (u,v) \in V^2 \,\middle|\, (u,v) \notin E \right\} .$$

Sei G als Adjazenzfeld gegeben. Nun will man $\overline{G}$ aus G erzeugen, wobei auch $\overline{G}$ am Ende in Form eines Adjazenzfeldes vorliegen soll. Ein solcher Vorgang benötigt mindestens $\Omega(|V|^2)$ Zeit. Erklären Sie kurz warum das so ist. [2 Punkte]

Lösung

Es muss jede Kante von G mindestens einmal betrachtet werden. Zudem muss jedes $(u, v) \in V^2$, das nicht in E ist, ausgegeben werden. Es müssen also alle $\Omega(|V|^2)$ maximal möglichen Kanten eines Graphen mit $|V|$ Knoten mindestens einmal verarbeitet werden.

Lösungsende

(weitere Teilaufgaben auf den nächsten Blättern)

Lösungsvorschlag

Fortsetzung von Aufgabe 1

e. Skizzieren Sie kurz, wie man *locateLocally* in (a,b)-Bäumen so verändern kann, dass die *locate* Operation $O(\log b \log n)$ Zeit braucht statt $O(b \log n)$ Zeit. [2 Punkte]

Lösung

Die Splitter der inneren Knoten bei $(a,b)-$Bäumen sind sortierte Arrays. Die in der Vorlesung vorgestellte Funktion *locateLocally* macht lineare Suche und verursacht Aufwand $O(b)$. Diese kann auf *binäre Suche* umgestellt werden und man erhält insgesamt die geforderte Laufzeit.

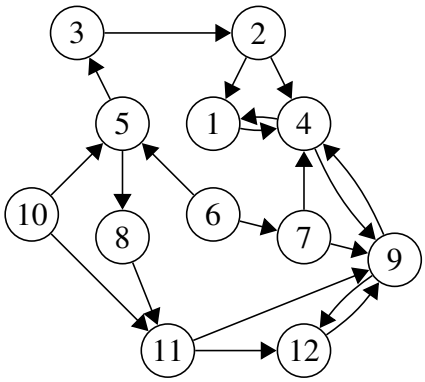
Lösungsende

f. Gegeben sei der unten abgebildete gerichtete Graph. Auf diesem Graphen soll eine Breitensuche durchgeführt werden. Wählen Sie den Startknoten so, dass bei der Breitensuche **keine Rückwärtskanten** auftreten.

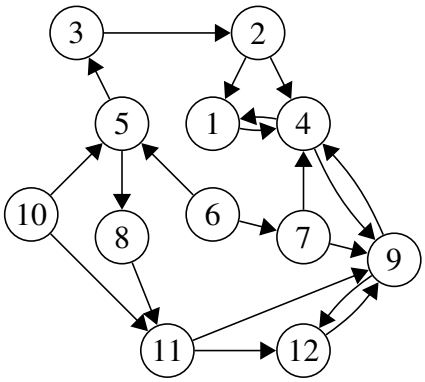
Markieren Sie den Startknoten und den zugehörigen Breitensuchbaum im Bild.

Hinweis: Der Startknoten darf auch so gewählt werden, dass die Breitensuche nicht alle Knoten des Graphen erreicht.

Für die **Lösung:**



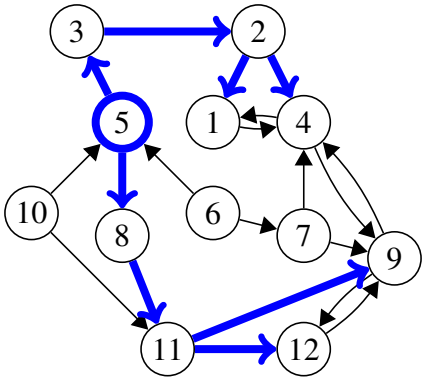
Kopie zum **Rechnen:**



[3 Punkte]

Lösung

Der Startknoten ist 5.



Lösungsende

g. Zeigen Sie oder widerlegen Sie, dass $2^{2^{n+1}} = \mathrm{O}\left(2^{2^n}\right)$ gilt. [2 Punkte]

Lösung

Die Behauptung gilt nicht. Andernfalls gäbe es $c, n_0 > 0$ mit $2^{2^{n+1}} = 2^{2 \cdot 2^n} = 4^{2^n} \leq c 2^{2^n}$ für alle $n > n_0$. Dann wäre aber

$$\frac{4^{2^n}}{2^{2^n}} = 2^{2^n} < c \quad \text{für all } n > n_0 \,,$$

im Widerspruch zur Unbeschränktheit von 2^{2^n}.

Lösungsende

(weitere Teilaufgaben auf dem nächsten Blatt)

Lösungsvorschlag

Fortsetzung von Aufgabe 1

h. Zeigen oder widerlegen Sie, dass $n^{\log_2 \log_2 n} = \Omega(n^2)$ gilt. [2 Punkte]

Lösung

Die Behauptung gilt. Es gilt nämlich die Äquivalenz

$$\log\log n > 2 \;\Leftrightarrow\; 2^{\log\log n} > 2^2 \;\Leftrightarrow\; \log n > 4 \;\Leftrightarrow\; 2^{\log n} > 2^4 \;\Leftrightarrow\; n > 16\,.$$

Für alle $n > 16$ hat man also $n^{\log\log n} > n^2$. Daraus folgt die Behauptung.

Lösungsende

i. Nennen Sie eine Operation, die auf unbeschränkten Arrays im besten Fall $\Omega(n)$ Zeit braucht und auf doppelt verketten Listen im schlimmsten Fall $O(1)$ Zeit braucht (n sei die Anzahl der jeweils enthaltenen Elemente). [1 Punkte]

Lösung

Mögliche Lösungen: *popFront*, *pushFront*

Lösungsende

j. Nennen Sie eine Operation, die auf einfach verketteten Listen im schlimmsten Fall $\Omega(n)$ Zeit braucht und auf unbeschränkten Arrays im schlimmsten Fall $O(1)$ Zeit braucht (n sei die Anzahl der jeweils enthaltenen Elemente). [1 Punkte]

Lösung

Wahlfreier Zugriff auf das i-te Element.

Lösungsende

Lösungsvorschlag

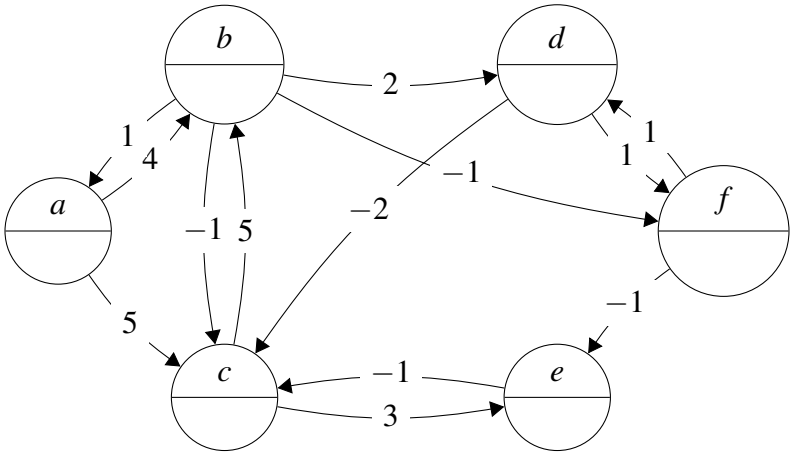
Aufgabe 2. Bellman-Ford Algorithmus [6 Punkte]

Gegeben sei der abgebildete gerichtete Graph mit Kantengewichten. Auf diesem Graph soll der Bellman-Ford Algorithmus mit *a* als Startknoten ausgeführt werden.

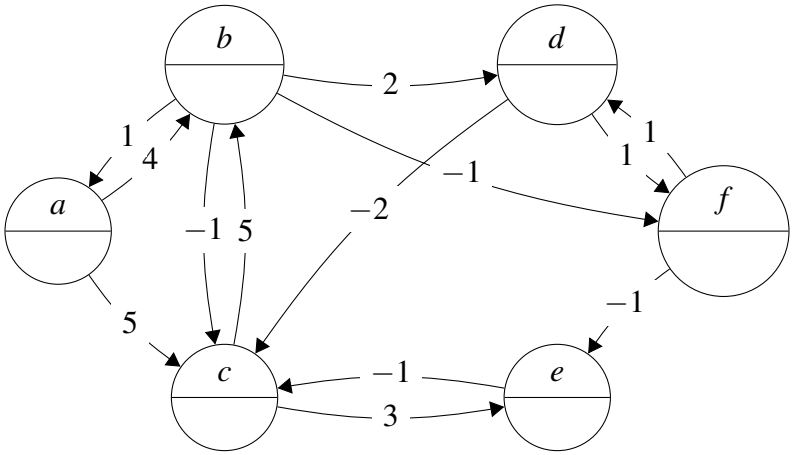
a. Tragen Sie in jeden Knoten jeweils die kürzeste Distanz von *a* zu diesem Knoten ein (in der unteren Hälfte jedes Knoten wurde dafür Platz frei gelassen). [2 Punkte]

b. Zeichnen Sie den vom Bellman-Ford Algorithmus berechneten Baum kürzester Wege in den Graph ein. [2 Punkte]

Tragen Sie die **Lösungen** von **a.** und **b.** bitte in **diesen** Graphen ein:

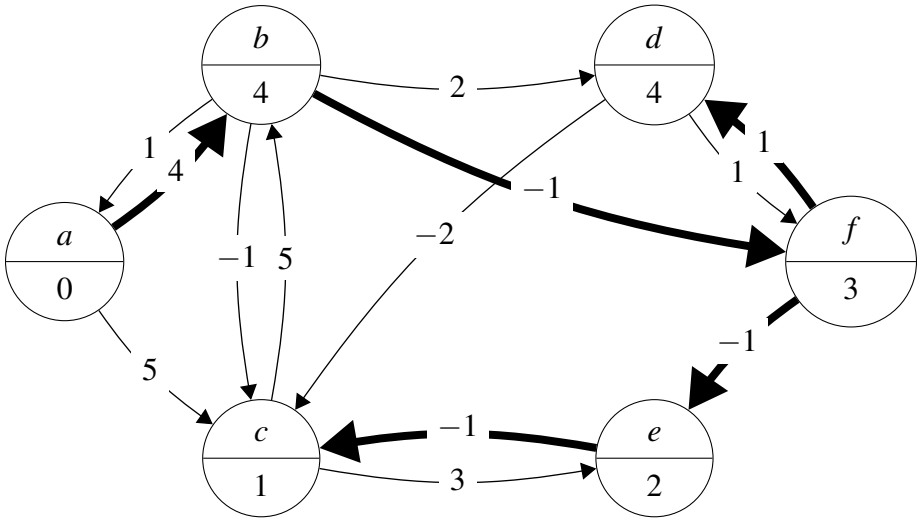


Kopie des obigen Graphen zum **Rechnen:**



Lösung

Lösung für Teilaufgaben **a.** und **b.**:



Lösungsende

(Teilaufgabe **c.** auf dem nächsten Blatt)

Lösungsvorschlag

Fortsetzung von Aufgabe 2

c. Geben Sie für den Bellman-Ford Algorithmus eine Bearbeitungsreihenfolge der Kanten an, so dass für den obigen Graph mit a als Startknoten eine einzige Runde des Algorithmus ausreicht, um die kürzesten Distanzen von a zu jedem Knoten zu berechnen. [2 Punkte]

Lösung

$(a,b),(b,a),(a,c),(b,c),(c,b),(b,d),(b,f),(d,f),(f,d),(d,c),(f,e),(e,c),(c,e)$

Bemerkung: In einer gültigen Lösung spielt nur die Reihenfolge der Kanten des Baumes kürzester Wege eine Rolle. Eine Kante dieses Baumes darf erst dann in der Folge auftreten, wenn ihr Startknoten a ist oder der Zielknoten einer anderen Kante, die in der Folge schon vorher aufgetreten ist. Die Reihenfolge der anderen Kanten ist egal.

Lösungsende

Lösungsvorschlag

Aufgabe 3. Entwurf einer Datenstruktur [8 Punkte]

Eine Datenstruktur D soll Paare der Form $(\mathit{ElementID},\ \mathit{Bewertung})$ speichern (sowohl *ElementID* und *Bewertung* seien Zahlen aus $\mathbb{Z}$). Paare (x,c) und (y,c') mit $x=y$ dürfen in D **nicht gleichzeitig** vorkommen. Weiter soll D folgende Operationen mit jeweils gegebenem Laufzeitverhalten unterstützen (n bezeichne dabei die Anzahl der in D enthaltenen Paare):

- $\mathit{insert}(x : \mathit{ElementID},\ c : \mathit{Bewertung})$
  fügt (x,c) in D ein. Ist schon ein Paar (y,c') mit $y=x$ in D vorhanden, so wird D nicht verändert. Der Zeitbedarf sei erwartet $O(\log n)$.
- $\mathit{removeMin}() : \mathit{ElementID} \times \mathit{Bewertung}$
  entfernt aus D ein Paar (x,c) mit **minimaler Bewertung** c und liefert das entfernte Paar als Ergebnis. Der Zeitbedarf sei erwartet $O(\log n)$.
- $\mathit{contains}(x : \mathit{ElementID}) : \mathit{boolean}$
  stellt fest, ob D ein Paar (y,c) mit $y=x$ enthält. Der Zeitbedarf sei erwartet $O(1)$.

Anwendungsbedingt sei bekannt, dass in D zu jedem Zeitpunkt höchstens m Paare gleichzeitig vorhanden sind, d. h. es gilt stets $n \leq m$. Über die Beschaffenheit der auftretenden Paare wisse man im Voraus aber nichts.

a. Skizzieren Sie, wie Sie diese Datenstruktur und die drei beschriebenen Operationen realisieren würden. [6 Punkte]

Lösung

Man realisiere D mit Hilfe eines binären Heaps B mit maximaler Größe m und einer Hashtabelle H mit Kollisionsauflösung mittels linearer Listen, die über $2m$ Slots verfügt. Bei jedem „Systemstart“ werde für H eine Hashfunktion aus einer universellen Familie zufällig gewählt. Für jedes (x,c) in D müssen B und H einen Eintrag enthalten, c ist *Key* bzgl. des binären Heap B und x ist *Key* bzgl. der Hashtabelle H.
Beim *insert* von (x,c) führt man zuerst $H.\mathit{find}(x)$ aus. Ist für x bereits ein Eintrag vorhanden, so tut man nichts weiter. Sonst führe man noch $B.\mathit{insert}(x,c)$ und $H.\mathit{insert}(x)$ aus.
Bei *removeMin* führt man zuerst $B.\mathit{deleteMin}()$ aus und gibt das vormals minimale Element x und seinen Heap-Schlüssel c als (x,c) zurück. Danach führt man dann $H.\mathit{remove}(x)$ aus.
Bei $\mathit{contains}(x : \mathit{ElementID})$ führe man $H.\mathit{find}(x)$ aus um herauszufinden, ob es ein Paar (y,c) mit $y=x$ gibt in D.

Lösungsende

(Mehr Platz zum Schreiben und Teilaufgabe **b.** auf dem nächsten Blatt)

Lösungsvorschlag

Fortsetzung von Aufgabe 3

b. Begründen Sie kurz, warum die drei Operationen in Ihrer Realisierung das geforderte Laufzeitverhalten aufweisen. [2 Punkte]

Lösung

Verwendete Heap-Operationen: Benötigen alle $O(\log n)$ Worst-Case-Zeit.
Hashtable-Operationen: Da H über $2m \geq 2n = \Omega(n)$ Slots verfügt und eine Hashfunktion aus einer universellen Familie zufällig gewählt wurde, dauern *find* und *remove* erwartet konstante Zeit, *insert* braucht sowieso deterministisch konstante Zeit.
Datenstruktur D: *insert* und *removeMin* brauchen erwartete Zeit $O(1)$ plus Worst-Case-Zeit $O(\log n)$, also insgesamt erwartet $O(\log n)$ Zeit. Die Operation *contains* braucht nur erwartet $O(1)$ Zeit, da ja nur $H.find(x)$ ausgeführt wird.

Lösungsende

Lösungsvorschlag

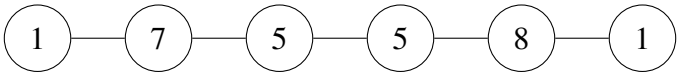
Aufgabe 4. Dynamisches Programmieren [12 Punkte]

Definition: Zu einem ungerichteten Graph $G = (V, E)$ ist eine Menge $U \subset V$ eine *unabhängige Menge* genau dann, wenn keine Knoten aus U in G benachbart sind, d. h. $\forall u, v \in U : \{u, v\} \notin E$.

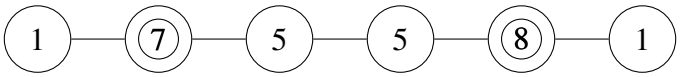
Definition: Zu einem ungerichteten Graph $G = (V, E)$ mit Knotengewichtsfunktion $c : V \to \mathbb{N}_{>0}$ ist eine Menge $U \subset V$ eine *maximale unabhängige Menge*, wenn U unabhängige Menge ist und unter allen unabhängigen Mengen das größtmögliche Gesamtgewicht hat, d. h. es existiert keine unabhängige Menge $U' \subset V$ mit $\sum_{u \in U'} c(u) > \sum_{u \in U} c(u)$

Festlegung: Im Folgenden sei $G := v_1 - v_2 - \cdots - v_k$ ein Pfad, d. h. G bestehe ausschließlich aus Kanten $\{v_\ell, v_{\ell+1}\}$ für $\ell \in \{1..k-1\}$. Für $\ell \leq k$ sei weiter $G_\ell := v_1 - \cdots - v_\ell$ der Teilpfad von G, der beim Knoten v_1 beginnt und bis einschließlich Knoten v_ℓ geht. G_0 soll als leerer Pfad interpretiert werden.

a. Geben Sie zu dem abgebildeten Pfad die *maximale unabhängige Menge* U an, indem Sie die Knoten markieren, die zu U gehören. Geben Sie außerdem das Gesamtgewicht von U an. Die Knotengewichte stehen in den Knoten. [2 Punkte]



Lösung



Gesamtgewicht: 15

Lösungsende

b. Es sei $m(\ell)$ das Gesamtgewicht einer *maximalen unabhängigen Menge* U_ℓ auf G_ℓ. Geben Sie eine Rekurrenz für $m(\ell)$ an, indem Sie auf die Werte der $m(i)$ für $i < \ell$ zurückgreifen! Begründen Sie kurz! [3 Punkte]

$$\begin{aligned} m(0) &= 0 \\ m(1) &= c(v_1) \\ m(\ell) &= \end{aligned}$$

Lösung

$$
\begin{aligned}
m(0) &= 0 \\
m(1) &= c(v_1) \\
m(\ell) &= \max\{m(\ell-1), m(\ell-2)+c(v_\ell)\}
\end{aligned}
$$

Begründung (ausführlich): Es gilt hier, dass sich optimale Lösungen aus optimalen Teillösungen zusammensetzen. Es ergeben sich zwei Fälle. Im ersten Fall gehört der Knoten v_ℓ zur maximalen unabhängigen Menge auf G_ℓ. In diesem Fall kann der direkte Nachbar $v_{\ell-1}$ nicht in dieser unabhängigen Menge sein, denn sonst wäre es keine unabhängige Menge. Weiterhin ist der Wert der maximalen unabhängigen Menge auf G_ℓ gerade durch den Wert der maximalen unabhängigen Menge auf $G_{\ell-2}$ plus das Gewicht des Knotens v_ℓ, der hinzugenommen wurde, gegeben. Daher ergibt sich als Gesamtgewicht dieser Menge $m(\ell-2)+c(v_\ell)$.
Im zweiten Fall gehört der Knoten v_ℓ nicht zur maximalen unabhängigen Menge. Dann ist der Wert der maximalen unabhängigen Menge gerade durch den Wert der maximalen unabhängige Menge $m(\ell-1)$ auf $G_{\ell-1}$ gegeben.

Lösungsende

(Teilaufgaben **c. und d.** auf dem nächsten Blatt)

Lösungsvorschlag

Fortsetzung von Aufgabe 4

c. Skizzieren Sie einen Algorithmus, der eine *maximale unabhängige Menge U* auf einem Pfad $G = v_1 - \cdots - v_k$ mit Knotengewichtsfunktion c **berechnet** und **ausgibt.** Der Algorithmus darf die Laufzeit $O(k)$ nicht überschreiten. [3 Punkte]

Lösung

Zur Lösung der Aufgabe verwenden wir dynamisches Programmieren. Die Werte $m(i)$ werden in linearer Zeit bottom up berechnet. Um die Lösung rekonstruieren zu können, führt man zusätzlich eine Entscheidungsvariable $x(i)$ mit, die angibt, ob v_i zu dieser Teillösung gehört oder nicht. Diese wird bei der Berechnung der $m(i)$ mitberechnet. Die Ausgabe kann dann mit folgender Prozedur erfolgen (Aufruf von output(k)):

```
1: procedure output(ℓ ∈ ℕ)
2:     if ℓ = 0 then return
3:     if x(ℓ) then
4:         print v_ℓ
5:         output(ℓ − 2)
6:     else
7:         output(ℓ − 1)
```

Lösungsende

d. Skizzieren Sie kurz einen Linearzeit-Algorithmus, der eine maximale unabhängige Menge auf einem ungerichteten **Baum** mit Knotengewichtsfunktion c **berechnet** und **ausgibt.** [4 Punkte]

Lösung

Hier ergeben sich die optimalen Lösungen eines Baums durch die optimalen Lösungen der Teilbäume. Man überlegt sich folgende Rekurrenz: im Fall, dass der Wurzelknoten des Baumes in der maximalen unabhängigen Menge ist, können die direkten Nachfolger von v im Baum nicht in der maximalen unabhängigen Menge enthalten sein. Im anderen Fall wählt man immer die optimalen Lösungen der Teilbäume aus, da die direkten Nachfolger sowohl enthalten sein können als auch nicht enthalten sein können.
Genauer: Zu einer Wurzel v, sei $m_{\text{in}}(v)$ der Wert der maximalen unabhängigen Menge die v enthält und $m_{\text{out}}(v)$ der Wert der maximalen unabhängigen Menge die v nicht enthält. Dann gilt die folgende Rekurrenz:

$$m_{in}(v) = c(v) + \sum_{u \in \text{children}(v)} m_{\text{out}}(u)$$
$$m_{out}(v) = \sum_{u \in \text{children}(v)} \max\{m_{\text{out}}(u), m_{\text{in}}(u)\}$$

Diese Werte können dann im Baum schichtweise bottom up berechnet werden. Die Werte der Blätter v werden initialisiert mit $m_{\text{in}}(v) = c(v)$, $m_{\text{out}}(v) = 0$. Um die Lösung dann auszugeben, kann man den Baum top down traversieren (z.B. levelweise).

Lösungsende

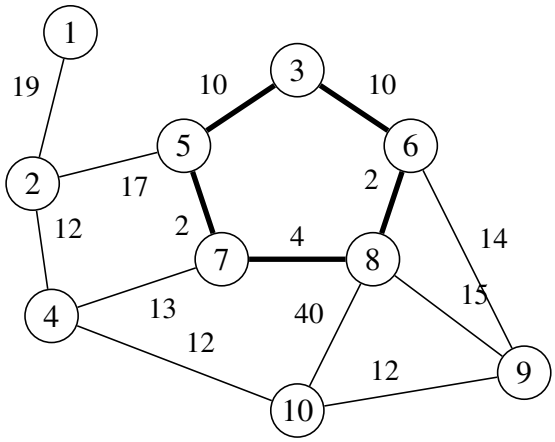
Lösungsvorschlag

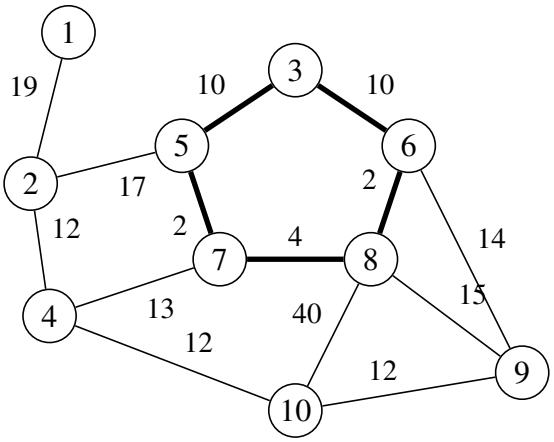
Aufgabe 5. Minimale Spannbäume [9 Punkte]

Gegeben sei ein ungerichteter zusammenhängender Graph G, in dem jede Kante mit einem positiven Gewicht versehen ist. In G sei ein einfacher Kreis markiert, so dass alle Kreiskanten ein Gewicht $< r$ und alle Kanten außerhalb des Kreises ein Gewicht $> r$ haben für ein $r \in \mathbb{R}_+$.
Hinweis: In einem einfachen Kreis haben alle Knoten Grad 2.

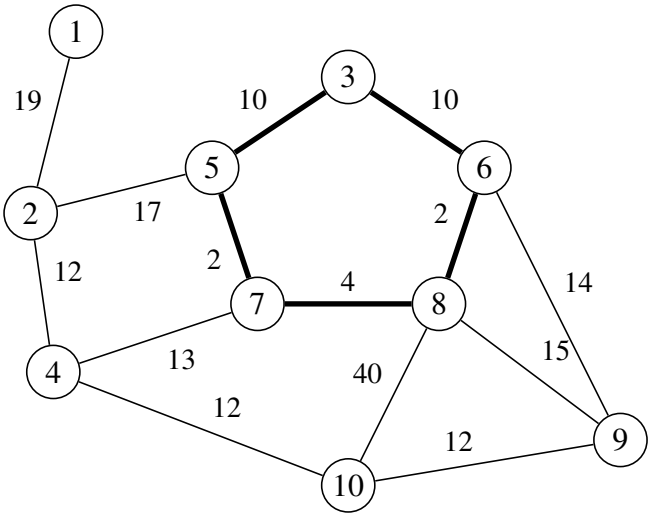
a. Zeichnen Sie in die folgenden beiden identischen Beispielgraphen jeweils einen MST ein. Dabei soll jeder MST mindestens eine Kante des markierten Kreises enthalten, die der andere MST nicht enthält. [3 Punkte]

Für die **Lösung:**





Kopie des Graphen zum **Rechnen:**

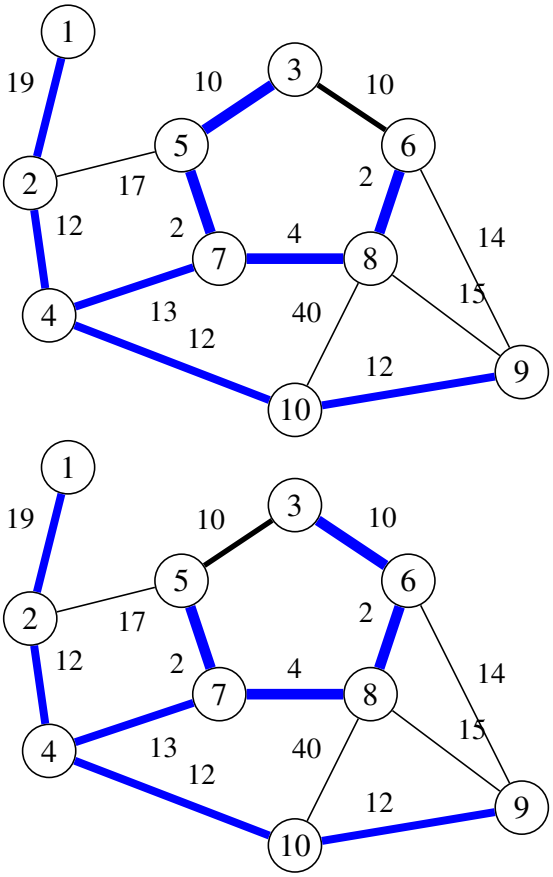


(Teilaufgabe **b.** auf dem nächsten Blatt)

Lösungsvorschlag

Fortsetzung von Aufgabe 5

Lösung



Lösungsende

b. Sei G ein Graph wie in der Einleitung zu dieser Aufgabe beschrieben. Sei k die Anzahl der Kanten auf dem markierten Kreis in G. Zeigen Sie: Jeder MST in G enthält genau $k-1$ Kanten des markierten Kreises. [6 Punkte]

Lösung

Falls ein MST alle Kanten des Kreises enthalten würde, könnte man sein Gewicht durch das Streichen einer schwersten Kante auf dem markierten Kreis reduzieren. Daher enthält ein MST niemals alle Kanten des markierten Kreises.

Annahme: Es existiert ein MST der mindestens zwei Kanten des markierten Kreises nicht enthält. Es sei $\{a,b\}$ eine Kante des markierten Kreises, die nicht in diesem MST enthalten ist. Da mindestens eine weitere Kante des Kreises nicht im MST enthalten ist, führt der eindeutige Pfad im MST von a nach b über mindestens eine Kante $\{u,v\}$, die nicht auf dem markierten Kreis liegt. Löscht man $\{u,v\}$ aus dem MST so zerfällt dieser in zwei Bäume, die sich durch $\{a,b\}$ wieder zu einem Baum zusammenfügen lassen. Dieser neue Baum hat wegen $c(\{a,b\}) < r < c(\{u,v\})$ ein geringeres Gesamtgewicht als der ursprüngliche MST, was zu einem Widerspruch führt.

Lösungsende

Lösungsvorschlag

Aufgabe 6. Hashing [6 Punkte]

Betrachten Sie die folgende Hashtabelle:

0	1	2	3	4	5	6	7	8	9
73	75	23	10	18		62	61	83	86

Zur Kollisionsauflösung wird offenes Hashing mit linearer Suche verwendet. Die Hashfunktion, $h(x) = x$ DIV 10, bildet eine Zahl $x \in \{1..99\}$ auf ihre höchstwertige Ziffer ab. Beispiel: $h(62) = 6$, $h(6) = 0$.

a. Sei die Tabelle nun leer. Geben Sie eine Folge von *insert*-Operationen an, so dass die Tabelle nach Ausführen dieser Operationsfolge den obigen Zustand hat. [3 Punkte]

Lösung

Eine mögliche Lösung: 62, 61, 83, 86, 73, 75, 23, 10, 18

Lösungsende

b. Sei die Tabelle in dem oben angegebenen Zustand. Geben Sie den Zustand der Tabelle nach dem Entfernen von 75 an. [3 Punkte]

0	1	2	3	4	5	6	7	8	9

Lösung

0	1	2	3	4	5	6	7	8	9
73	10	23	18			62	61	83	86

Lösungsende