Lösungsvorschlag

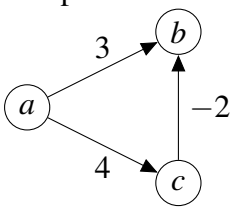
**Aufgabe 1.** Kurzaufgaben [15 Punkte]

Beantworten Sie die folgenden Fragen. Antworten Sie jeweils kurz!

**a.** Geben Sie einen gerichteten Graphen ohne negative Zyklen und einen Startknoten an, so dass der Algorithmus von Dijkstra *kein* korrektes Ergebnis liefert. Begründen Sie kurz. [3 Punkte]

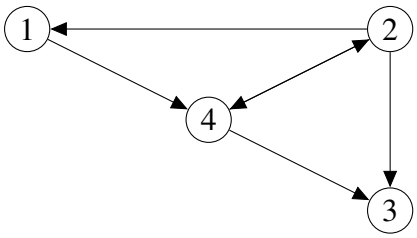
## Lösung

Beispiel:




Der Startknoten sei $a$. Nach Relaxierung der ausgehenden Kanten von $a$ ist $b$ der Knoten mit minimaler vorläufiger Distanz von $a$. Daher wird $b$ als nächstes gescannt, und behält die endgültige Distanz 3. Er wäre aber über $c$ mit Distanz 2 von $a$ erreichbar.

**b.** Geben Sie die Adjazenzfeld-Darstellung des folgenden Graphen an: [3 Punkte]




| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| $V$ | | | | | |

| | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| $E$ | | | | | | |

## Lösung

| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| $V$ | 1 | 2 | 5 | 5 | 7 |

| | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| $E$ | 4 | 1 | 3 | 4 | 2 | 3 |

**c.** Nennen Sie zwei Operationen, die auf einer doppelt verketteten Liste in konstanter Zeit ausgeführt werden können, auf einem unbeschränkten Feld aber nicht in amortisiert konstanter Zeit. [2 Punkte]

## Lösung

| |
|---|
| insert, remove, pushFront, popFront, concat, splice |

(weitere Teilaufgaben zu dieser Aufgabe auf dem nächsten Blatt)

Lösungsvorschlag

**Fortsetzung von Aufgabe 1**

**d.** Nennen Sie zwei Greedy-Algorithmen, die optimale Ergebnisse berechnen. [2 Punkte]

## Lösung

z.B. Dijkstra, Jarník-Prim, Kruskal, Selection-Sort

**e.** Sortierte Listen: Welche Ausgangsgrade kann ein Wurzelknoten bzw. ein innerer Knoten eines $(2,4)$-Baums haben? [2 Punkte]

## Lösung

Wurzelknoten: 1–4, innerer Knoten: 2–4

**f.** Gegeben seien die beiden Rekurrenzen:
$r(1) = 1$, $r(n) = 5n + a \cdot r(n/4)$, $n = 4^k, k \in \mathbb{N}$ $s(1) = 3$, $s(n) = 7n + 8s(n/2)$, $n = 2^k, k \in \mathbb{N}$
Bestimmen Sie $a > 0$, so dass $r(n)$ und $s(n)$ die gleiche asymptotische Laufzeit im $\Theta$-Kalkül besitzen (mit Begründung). [3 Punkte]

## Lösung

Nach Master-Theorem gilt $s(n) = \Theta\left(n^{\log_2 8}\right) = \Theta\left(n^3\right)$. Für $a > 4$ gilt $r(n) = \Theta\left(n^{\log_4 a}\right)$. Somit muss für $a$ gelten: $\log_4 a = 3 \Leftrightarrow a = 4^3 = 64$.

Lösungsvorschlag

**Aufgabe 2.** Algorithmen-Entwurf: Summanden finden [14 Punkte]

Gegeben sei ein Array $A = A[1], \ldots, A[n]$ mit $n$ Zahlen in beliebiger Reihenfolge. Für eine gegebenen Zahl $x$ soll ein Paar $(A[i], A[j])$, $1 \leq i, j \leq n$ gefunden werden, für das gilt: $A[i] + A[j] = x$.
**a.** Geben Sie eine Lösung für $x = 33$ und $A = (7, 15, 21, 14, 18, 3, 9)$ an. [2 Punkte]

## Lösung

$A[2] = 15, A[5] = 18$

**b.** Geben Sie einen effizienten Algorithmus an, der das Problem in Zeit $O(n \log n)$ löst, und bei Erfolg ein Paar $(A[i], A[j])$ ausgibt, ansonsten NIL. Hinweis: Für einen Algorithmus mit quadratischer Komplexität gibt es höchstens 3 Punkte. [6 Punkte]

## Lösung

```
sort(A)                                   // sortiere Feld aufsteigend
i=1                                       // linker Rand
j=n                                       // rechter Rand
while i ≤ j do
    if A[i] + A[j] < x then i++           // Summe größer machen
    else
        if A[i] + A[j] > x then j--       // Summe kleiner machen
        else return (A[i], A[j])          // Lösung gefunden
return NIL
```

Eine alternative Lösung mit Hashing läuft sogar in linearer Zeit: Im ersten Schritt alle Elemente in die Hash-Tabelle einfügen, im zweiten Schritt zu jeder Zahl das "Gegenstück" suchen.

(weitere Teilaufgaben zu dieser Aufgabe auf dem nächsten Blatt)

Lösungsvorschlag

**Fortsetzung von Aufgabe 2**

**c.** Skizzieren Sie einen Beweis für die Korrektheit Ihres Algorithmus (z. B. eine aussagekräftige Schleifeninvariante und eine kurze Begründung). [Für den Beweis eines Algorithmus mit quadratischer Komplexität gibt es höchstens 3 Punkte.] [6 Punkte]

## Lösung

Schleifeninvariante: Die Lösung befindet sich entweder innerhalb von $i$ und $j$ im sortierten Array $A$, oder es gibt keine Lösung.
Terminierung (optional): Entweder der Algorithmus terminiert in einem Schleifendurchlauf mittels Return, oder das Intervalls $[i, j]$ wird um ein Element kleiner, d. h. nach endlich viel Schritten $i > j$ und damit Ende der Schleife und Return.
Gültigkeit der Invariante: Zu Beginn gilt die Invariante trivialerweise. Wird die Schleife durch Return verlassen, gilt die Invariante offensichtlich immer noch, da sich $i$ und $j$ nicht ändern. Wir $i$ auf $i+1$ erhöht, so schließen wir nur die möglichen Lösungen $(A[i], A[k]), k \in [i, j]$ neu aus. Für diese gilt aber $A[i]+A[k] \leq A[i]+A[j]$ wegen der Sortierung, und damit auch $A[i]+A[k] < x$, sie sind also nicht gültig. Der Beweis, dass durch $j--$ keine gültigen Lösungen verloren gehen, ist symmetrisch.
Korrektheit insgesamt: Falls die Schleife durch Return verlassen wurde, so gab es offensichtlich die Lösung $A[i]+A[j] = x$, da dies aus $\neg(A[i]+A[j] < x)$ und $\neg(A[i]+A[j] > x)$ folgt. Diese wurde korrekt zurückgegeben. Falls die Schleife aufgrund der Bedingung verlassen wurde, so gilt $i > j$, und es kann sich nach der Invariante keine Lösung mehr zwischen $i$ und $j$ befinden. Auch dies wird dann nach Spezifikation zurückgegeben.

Lösungsvorschlag

**Aufgabe 3.** Algorithmen-Entwurf: Anagramme finden [10 Punkte]

Ein *Anagramm* ist eine Zeichenkette, welche durch Permutation der Zeichen aus einer anderen Zeichenkette entstanden ist. Beispielsweise ist "ABCDE" ein Anagramm von "DCEBA" (und umgekehrt). Auch ist jede Zeichenkette ein Anagramm von sich selbst.
Annahme: Alle Zeichenketten sind endlich und die Zeichen aus der Menge $\{A, B, C, \ldots, Z\}$.
**a.** Geben Sie eine Funktion $f(s)$ auf Zeichenketten an, die zwei Wörtern genau dann das gleiche Bild zuordnet, wenn sie Anagramme sind. Genau wenn die Zeichenkette $s_1$ ein Anagramm von $s_2$ ist, soll gelten $f(s_1) = f(s_2)$. [3 Punkte]

## Lösung

Es gibt viele Lösungen, generell muss $s$ normalisiert werden.
Beispiel 1: $f(s) :=$ Zeichen von $s$ alphabetisch sortiert, z.B. ANAGRAMM $\rightarrow$ AAAGMMNR.
Beispiel 2: $f(s) :=$ Zeichenkette aus Anzahl der einzelnen Zeichen in $s$,
z.B. ANAGRAMM $\rightarrow$ A3G1M2N1R1 (oder andere Kodierungen)

(weitere Teilaufgaben zu dieser Aufgabe auf dem nächsten Blatt)

Lösungsvorschlag

**Fortsetzung von Aufgabe 3**

**b.** Gegeben sei eine Menge $S$ von Zeichenketten. Geben Sie einen effizienten Algorithmus an, der zu einer Anfragezeichenkette $q$ feststellt, ob sich ein Anagramm davon in $S$ befindet. Der Anfragealgorithmus soll eine erwartete Laufzeit besitzen, die unabhängig von $|S|$ ist.
Um diese Laufzeit zu erreichen, müssen Sie eine geeignete Datenstruktur aus $S$ *vorberechnen*. Die Laufzeit der Vorberechnung wird nicht zur Laufzeit des Anfragealgorithmus gezählt, allerdings ist zur Zeit der Vorberechnung die Anfragezeichenkette $q$ noch unbekannt.
Nehmen Sie die Existenz einer korrekten, wie in Teilaufgabe **a.** beschriebenen Funktion $f$ an. Begründen Sie die Korrektheit und dass die Laufzeit Ihres Algorithmus unabhängig von $|S|$ ist.
[7 Punkte]

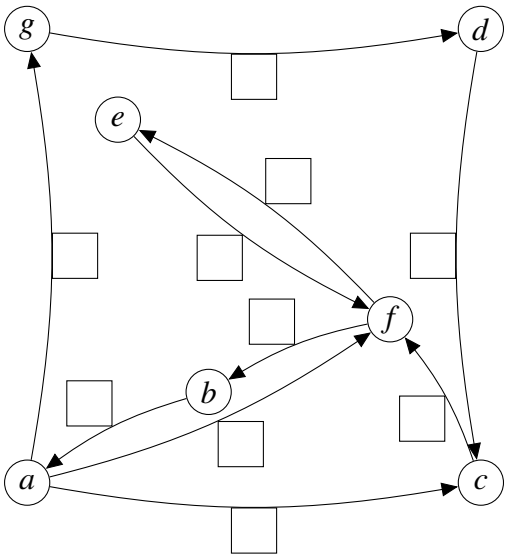
## Lösung

Lösung mit Hashtabelle: Berechne eine Hashtabelle $H$ der Größe $\Omega(|S|)$ mit verketteten Listen vor, welche eine zufällige Hashfunktion aus einer Familie universeller Hashfunktion verwendet. Speichere in $H$ alle Bilder von $S$ unter $f$. Somit gibt es genau dann ein Anagramm von $q$ in $S$, wenn $f(q)$ in $H$ ist. Eine Anfrage ist somit eine Anfrage nach $f(q)$ an $H$.
Die erwartete Laufzeit einer Anfrage ist damit unabhängig von $|S|$, da nur eine konstante Anzahl Kollisionen erwartet wird (Satz 1 aus Vorlesung, Folie 121).
Auch andere Lösungen, die nicht auf den Datenstrukturen der Vorlesung aufbauen, z.B. *trie*, werden als korrekt bewertet, allerdings muss dort näher auf diese Datenstrukturen eingegangen werden.

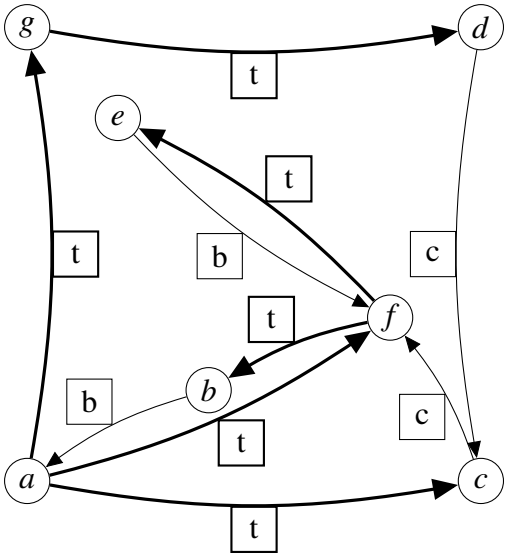
Lösungsvorschlag

**Aufgabe 4.** Graphtraversierung [8 Punkte]

**a.** Auf folgendem Graphen werde eine Breitensuche von $a$ aus durchgeführt. Klassifizieren Sie die Kanten, indem Sie in die Kästchen entweder $t$ (tree), $b$ (backward) und $c$ (cross) eintragen. [5 Punkte]




## Lösung




**b.** Zeigen Sie, dass bei Breitensuche keine Vorwärtskanten entstehen. Hinweis: Eine Vorwärtskante ist eine nicht im Breitensuchbaum enthaltene Kante, die zwei Knoten verbindet, die auf einem Pfad im Baum liegen, wobei der Startknoten näher an der Wurzel ist als der Zielknoten. [3 Punkte]

## Lösung

Die Entfernung zur Wurzel ist bei der Breitensuche direkt durch die Schicht gegeben. Annahme: Es gibt ein Vorwärtskante $e = (v, w)$. Somit gilt $\text{level}(\text{v}) < \text{level}(w)$. $w$ ist direkt von $v$ erreichbar, also muss es direkt in der nächsten Schicht liegen: $\text{level}(\text{v}) + 1 = \text{level}(w)$. Damit ist $e$ aber eine Baumkante.

Lösungsvorschlag

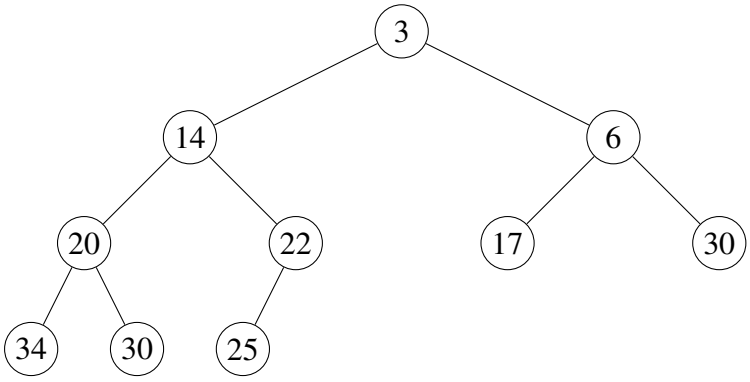
**Aufgabe 5.** Prioritätswarteschlangen [7 Punkte]

**a.** Zeichnen Sie den Binärbaum (mit Knoten und Kanten) zu dem in impliziter Baum-Darstellung gegebenen binären Heap:

| 3 | 14 | 6 | 20 | 22 | 17 | 30 | 34 | 30 | 25 |
|---|---|---|---|---|---|---|---|---|---|

[2 Punkte]

## Lösung




**b.** Der nachfolgende Binärbaum in impliziter Baum-Darstellung verletzt die Heap-Eigenschaft. Bitte markieren Sie die beiden Knoten mit einem X, zwischen denen die Heap-Eigenschaft verletzt ist.

| 3 | 14 | 17 | 15 | 20 | 82 | 17 | 16 | 17 | 18 |
|---|---|---|---|---|---|---|---|---|---|

[2 Punkte]

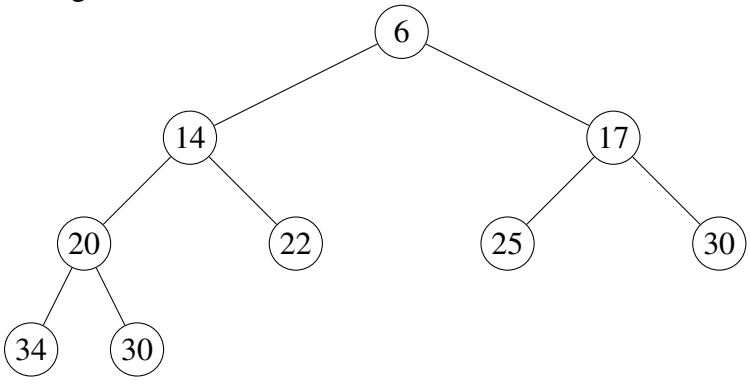
## Lösung

| | | | | X | | | | | X |
|---|---|---|---|---|---|---|---|---|---|
| 3 | 14 | 17 | 15 | 20 | 82 | 17 | 16 | 17 | 18 |

**c.** Führen Sie die Funktion `deleteMin` auf dem binären Heap von Teilaufgabe **a** aus und geben sie den Rückgabewert der Funktion sowie den resultierenden Heap als Binärbaum (mit Knoten und Kanten) an. [3 Punkte]

## Lösung

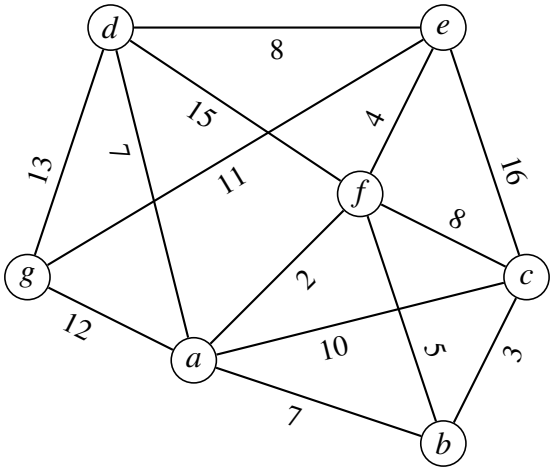
Rückgabewert: 3

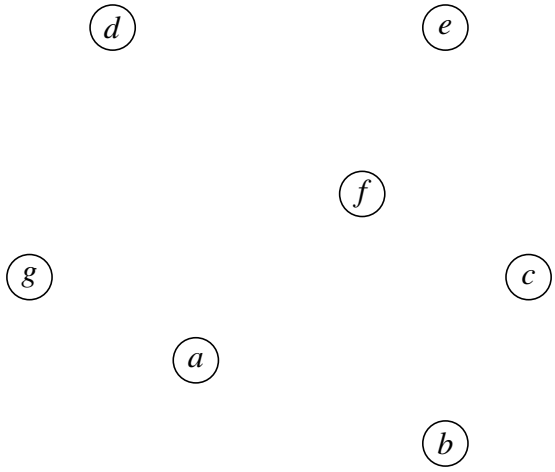


Lösungsvorschlag

**Aufgabe 6.** Minimale Spannbäume [6 Punkte]

Benutzen Sie den Algorithmus von Kruskal aus der Vorlesung, um für den links gegebenen Graphen einen minimalen Spannbaum (MST) zu berechnen. Zeichnen Sie den MST in das rechte Knotengerüst ein und NUMMERIEREN Sie die Kanten in der Reihenfolge, in der sie hinzugefügt werden.




## Lösung