Nachname:

Vorname:

Matrikelnummer:

Lösungsvorschlag

Karlsruher Institut für Technologie
Institut für Theoretische Informatik

Prof. Dr. P. Sanders 25.08.2025

Klausur Algorithmen I

Aufgabe 1.	Kleinaufgaben	6 Punkte
Aufgabe 2.	Graphalgorithmen	10 Punkte
Aufgabe 3.	Hashing	11 Punkte
Aufgabe 4.	Optimierung	11 Punkte
Aufgabe 5.	Suchbäume und Heaps	11 Punkte
Aufgabe 6.	Minimale Spannbäume	11 Punkte

Bitte beachten Sie:

- Als Hilfsmittel ist nur **ein** DIN-A4 Blatt mit Ihren **handschriftlichen** Notizen zugelassen.
- Schreiben Sie Ihre Antworten nur in blau oder schwarz, mit dokumentenechtem Stift.
- **Schreiben** Sie auf **alle** Blätter der Klausur und Zusatzblätter Ihre **Matrikelnummer**.
- Die Klausur enthält **16 Seiten**.

Aufgabe 1. Kleinaufgaben [6 Punkte]

a. Nennen Sie einen Vorteil von Dummy-Elementen bei doppelt verketteten Listen. [1 Punkt]

Lösung

- Invariante `next->prev = prev->next = this` immer erfüllt.
- Vermeidung vieler Sonderfälle, beispielsweise durch Wächter im Dummy.

b. Wir betrachten ein unbeschränktes Array mit n Elementen und Kapazität w. Beschreiben Sie, wann die Worst-Case Laufzeit beim Aufrufen einer `pushBack`-Operation eintritt. [1 Punkt]

Lösung

Worst-Case: Wenn das Unbounded Array voll ist ($w = n$) und die Elemente in ein größeres Array umkopiert werden müssen. Wenn $4n \leq w$ am Ende des `popBacks`, kopiere die n Elemente in ein neues Array der Größe $2n$ um und gebe den Speicher für das alte Array frei.

c. Geben Sie die Lösungen für folgende Rekursionsgleichungen im Θ-Kalkül an.

$$\begin{array}{llll} \text{i)} & \mathrm{r}(1) = 2, & \mathrm{r}(n) = n + 6 \cdot \mathrm{r}(n/6), & n \in \mathbb{N}^+ \\ \text{ii)} & \mathrm{s}(1) = 1, & \mathrm{s}(n) = 5n + 3 \cdot \mathrm{s}(n/5), & n \in \mathbb{N}^+ \end{array}$$

[2 Punkte]

Lösung

1. $r(n) \in \Theta(n \log n)$
2. $s(n) \in \Theta(n)$

d. Nennen Sie einen Sortieralgorithmus aus der Vorlesung, welcher eine Worst-Case Laufzeit von $O(n \log n)$ aufweist und nur $O(1)$ zusätzlichen Speicherplatz benötigt. [1 Punkt]

Lösung

Heapsort, Quicksort (wenn richtig implementiert), Mergesort (wenn richtig implementiert)

e. Nennen Sie einen Vorteil der gleichverteilten, zufälligen Auswahl des Pivot-Elements bei Quicksort gegenüber der Wahl des ersten Elements als Pivot. [1 Punkt]

Lösung

Wären alle Eingaben gleich wahrscheinlich, wäre die Wahl des Pivotelements unkritisch. In der Praxis sind jedoch manche Eingaben wahrscheinlicher, so z. B. sortierte, oder beinahe sortiete Arrays. Würde man immer das erste Element als Pivot wählen, würde in genau diesem Fall jedoch die quadratische Worst-Case Laufzeit von Quicksort auftreten.

Aufgabe 2. Graphalgorithmen [10 Punkte]

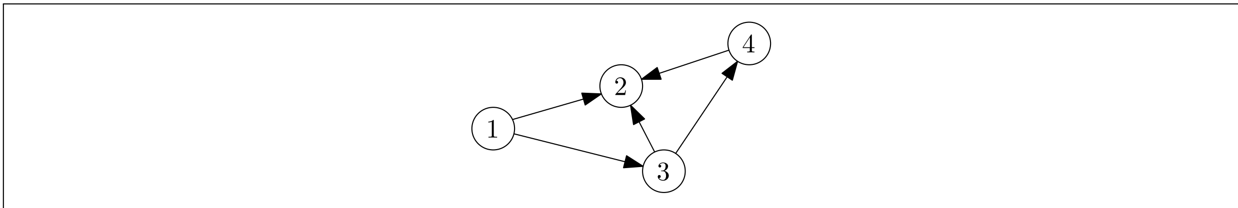
a. Gegeben sei das folgende Adjazenzfeld. Zeichnen Sie den zugehörigen Graphen.

V	1	3	3	5	6

E	2	3	2	4	2
Index	~~0~~ 1	~~1~~ 2	~~2~~ 3	~~3~~ 4	~~4~~ 5

Hinweis: Bei dem gestrichelt dargestellten, letzten Element in V handelt es sich um ein Dummy-Element wie in der Vorlesung beschrieben. [1 Punkt]

Lösung



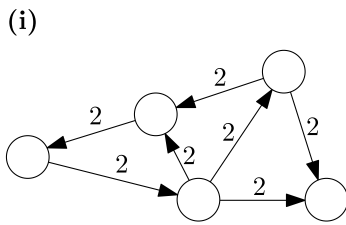
b. Handelt es sich bei dem Graphen aus Teilaufgabe **a.** um einen DAG (Directed Acyclic Graph)? Begründen Sie Ihre Antwort. [1 Punkt]

Lösung

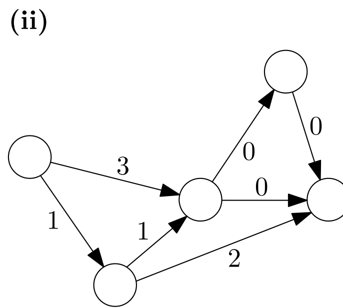
Der Graph ist ein DAG, weil er keine gerichteten Kreise enthält.

c. Gegeben seien die folgenden drei gerichteten Graphen **(a)** bis **(c)**.

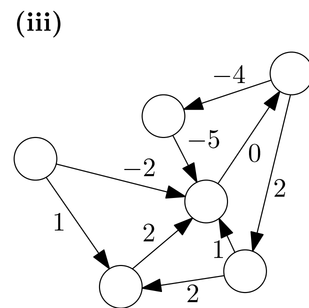
(i)



(ii)



(iii)



Welcher Kürzeste-Wege Algorithmus liefert auf welchem Graphen den korrekten kürzesten Weg für *alle* Start- und Zielknoten? Markieren Sie jede Zelle entsprechend mit ✓ (alle Ergebnisse korrekt) oder ✗ (mindestens ein inkorrektes Ergebnis).

Algorithmus	**Graph (i)**	**Graph (ii)**	**Graph (iii)**
BFS			
Dijkstra			
Bellman-Ford			

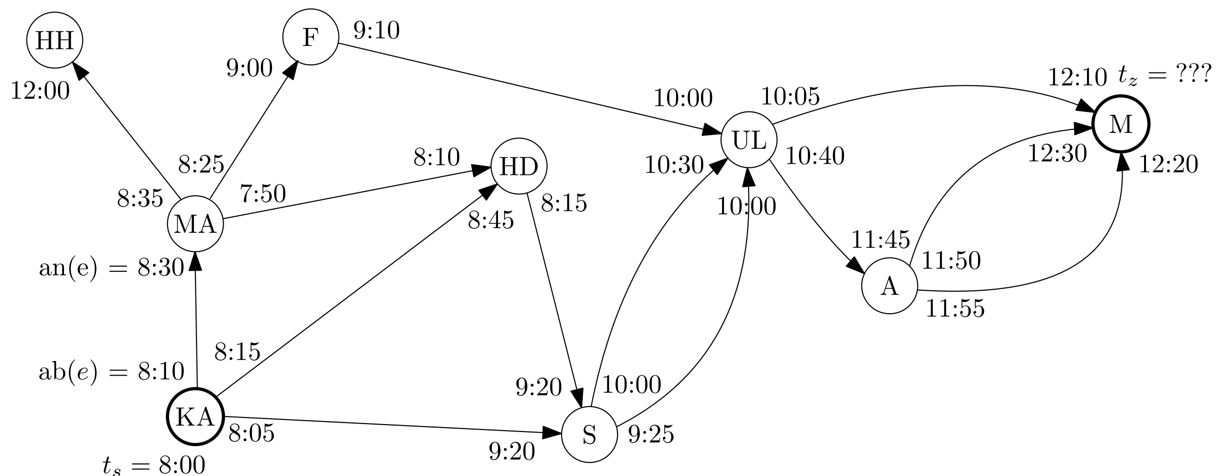
[3 Punkte]

Lösung

Algorithmus	**Graph (i)**	**Graph (ii)**	**Graph (iii)**
BFS	✓	✗	✗
Dijkstra	✓	✓	✗
Bellman-Ford	✓	✓	✓

d. Gegeben sei das folgende Problem auf einem Graphen: Jeder Knoten v stelle einen Bahnhof dar und jede Kante e eine Fahrt. Zudem hat jede Fahrt eine Abfahrts- und eine Ankunftszeit $\text{ab}(e)$ und $\text{an}(e)$. Ein Fahrgast startet an einem Startknoten s zu einer gegebenen Zeit t_s und kann in jedem Knoten v mit frühester Ankunftszeit t_v lediglich Fahrten nehmen, welche um oder nach t_v abfahren ($\text{ab}(e) \geq t_v$). Der Fahrgast kann beliebig lange an einem Knoten auf eine Fahrt warten, muss also nicht die nächste Fahrt nehmen. Es kann mehrere Kanten mit identischen Abfahrts- und Zielbahnhöfen geben, die sich jedoch in ihrer Abfahrts- und Ankunftszeit unterscheiden.

Beispiel: Im folgenden Graph fährt ein Fahrgast von $s = \text{KA}$ mit frühester Abfahrtszeit $t_s = 8{:}00$ ab. Die frühest mögliche Ankunftszeit in $z = \text{M}$ ist $t_\text{M} = 12{:}10$ Uhr (KA $\to$ S $\to$ UL $\to$ M).



Entwerfen Sie einen Algorithmus, der für einen gegebenen Graphen, Abfahrtsknoten s und Abfahrtszeitpunkt t_s in $O((m+n)\log(n))$ Zeit die früheste Ankunftszeit t_z für alle Ziele sowie die zugehörigen Pfade dorthin berechnet. Hierbei sei n die Anzahl an Knoten und m die Anzahl an Kanten im gegebenen Graphen. [5 Punkte]

Lösung

Verwende einen modifizierten Dijkstra mit einer Binary Heap oder Fibonacci Prioritätswarteschlange.

function PUBLICTRANSITROUTING(s, z, t_s): t_z, parent
  $t \leftarrow \langle \infty, \ldots, \infty \rangle$
  $\text{parent}[s] \leftarrow s$
  $t[s] \leftarrow t_s$
  $Q.\text{insert}(s)$
  while $Q \neq \emptyset$ **do**
    $u \leftarrow Q.\text{deleteMin}()$
    for all edge $e = (u,v) \in E$ **do**
      if $t[u] \leq \text{ab}(e) \wedge \text{an}(e) < t[v]$ **then** ▷ Unterschied zu Dijkstra
        $t[v] \leftarrow \text{an}(e)$ ▷ Unterschied zu Dijkstra
        $\text{parent}[v] \leftarrow u$
        if $v \in Q$ **then**
          $Q.\text{decreaseKey}(v)$
        else
          $Q.\text{insert}(v)$
  return (t, parent)

Aufgabe 3. Hashing [11 Punkte]

a. Betrachten Sie die unten abgebildete Linear Probing Hashtabelle mit Kapazität $m = 10$. Die Elemente wurden mit der Hashfunktion $h(n) = n \bmod 10$ eingefügt. Geben Sie eine mögliche Sequenz an Einfügeoperationen an, die eine leere Hashtabelle in den gezeigten Zustand überführt. [1 Punkt]

Hashtabelle		31	11		44		66	16	7	6
Index	0	1	2	3	4	5	6	7	8	9

Lösung

Zum Beispiel: 31, 11, 44, 66, 16, 7, 6.

b. Nennen Sie jeweils einen Vor- und Nachteil von Hashing mit verketteten Listen gegenüber offenem Hashing mit linearer Suche. [2 Punkte]

Lösung

Vorteile:

- Insert worst-case in $O(1)$
- Für dichte Hashtabelle besser
- Referentielle Integrität
- Leistungsgarantien mit universellem Hashing

Nachteile:

- Weniger Platz-effizient
- Weniger Cache-effizient

c. Für $n \in \mathbb{N}$ seien eine n-elementige Menge $Z = \{z_1, \dots, z_n\} \subseteq \mathbb{Z}$ sowie $C \in \mathbb{Z}$ geben. Geben Sie einen Algorithmus mit erwarteter Laufzeit $O(n)$ an, der alle Paare $a, b \in Z$ mit $a + b = C$ bestimmt. Begründen Sie die Laufzeit Ihres Algorithmus. [4 Punkte]

Lösung

Erstelle eine Hashtabelle H. Iteriere über Z. Falls für das aktuelle Element $z_i \in Z$ das Element $C - z_i$ bereits in H enthalten ist, füge $(z_i, C - z_i)$ zum Ergebnis hinzu. Andernfalls füge z_i in H ein.
Laufzeit: Der Algorithmus iteriert einmal über alle Elemente. Für jedes Element wird höchstens eine `insert`- und eine `contains`-Operation ausgeführt (jeweils erwartet konstante Laufzeit). Insgesamt ergibt sich daher erwartet lineare Laufzeit.
Häufige Fehler: Schlechte Hashfunktionen spezifiziert; $0 \leq C \leq n$ angenommen.

Im Nachfolgenden bezeichnet K eine Schlüsselmenge, $m \in \mathbb{N}$ und $\mathscr{H} \subseteq \{0,\dots,m-1\}^K$ eine Familie von Hashfunktionen über K. Wie in der Vorlesung definiert nennen wir $\mathscr{H}$ *universell* über K, falls für alle $a,b \in K$ mit $a \neq b$ und zufälligem $h \in \mathscr{H}$ gilt: $\mathbb{P}[h(a) = h(b)] \leq 1/m$.

d. Sei $h_0\colon K \to \{0,\dots,m-1\}, k \mapsto 0$ die Hashfunktion, die alle Schlüssel auf die 0 abbildet. Beweisen oder widerlegen Sie: es gibt universelle Familien von Hashfunktionen, die h_0 enthalten. [2 Punkte]

Lösung

Die Aussage ist wahr. Intuitiv: auch „gute" Familien können einzelne „schlechte" Hashfunktionen enthalten. Konkrete Beispiele: $\mathscr{H} = \{0,\dots,m-1\}^K$ (also voll zufälliges Hashing) oder $H^\bullet$ (aus der Vorlesung) enthalten h_0.
Häufige Fehler: Nicht beachtet, dass $h(\cdot)$ in $\mathbb{P}[h(a) = h(b)] \leq 1/m$ zufällig aus $\mathscr{H}$ gezogen wird (die Hashfunktionen selber sind deterministisch); nicht beachtet, dass „universell" eine Eigenschaft von $\mathscr{H}$ und nicht einer einzelnen Hashfunktion ist.

e. Sei nun $\mathscr{H}$ eine universelle Familie von Hashfunktionen über K. Wir definieren eine neue Schlüsselmenge $K' = \{(k_1,k_2) \mid k_1,k_2 \in K\}$. Weiter definieren wir für jede Hashfunktion $h \in \mathscr{H}$ eine neue Hashfunktion h' wie folgt:

$$h'\colon K' \to \{0,\dots,m-1\}, (k_1,k_2) \mapsto h(k_1) + h(k_2) \bmod m.$$

Sei $\mathscr{H}'$ die Familie dieser neuen Hashfunktionen. Beweisen Sie, dass $\mathscr{H}'$ im Allgemeinen nicht universell über K' ist. [2 Punkte]

Lösung

Seien $a,b \in K$, $a \neq b$. Dann ist für alle $h' \in \mathscr{H}'$: $h'(a,b) = h'(b,a)$, also insbesondere für ein zufälliges $h' \in \mathscr{H}'$: $\mathbb{P}[h'(a,b) = h'(b,a)] = 1$.
Häufige Fehler: Unvollständig spezifizierte Gegenbeispiele.

Aufgabe 4. Optimierung [11 Punkte]

Dr. Meta hat die Weltherrschaft errungen und führt als Währung $\mathbb{M}$-Coins ein. Es gibt $\mathbb{M}$-Coins mit der Wertigkeit ①, ③, ④, und ⑤. Jeder Geldbetrag m kann mit einer Multimenge von $\mathbb{M}$-Coins dargestellt werden; dabei gibt es oft mehrere Möglichkeiten. Zum Beispiel kann der Betrag $m = 5$ mit den Münzkombinationen {⑤}, {④,①}, {③,①,①}, oder {①,①,①,①,①} dargestellt werden. Im Folgenden sollen Sie unter Verwendung von Dynamischer Programmierung einen Algorithmus entwerfen, welcher die minimale *Anzahl* von Münzen berechnet um einen gegebenen Betrag m darzustellen.

a. Wie lautet das Optimalitätsprinzip, welches für die Dynamische Programmierung grundlegend ist? [1 Punkt]

Lösung

Das Optimalitätsprinzip, das für die dynamische Programmierung grundlegend ist, besagt, dass die optimale Lösung eines Problems aus optimalen Lösungen seiner Teilprobleme zusammengesetzt werden kann. Gibt es mehrere optimale Lösungen für ein Problem, sind diese beim Zusammensetzen austauschbar.

b. Füllen Sie die folgende Tabelle aus, welche beschreibt, wie viele Münzen jeweils *mindestens* nötig sind um den gegebenen Betrag darzustellen.

Betrag ($\mathbb{M}$)	0	1	2	3	4	5	6	7	8	9	10
Anzahl Münzen	0	1	2	1	1	1					

[1 Punkt]

Lösung

Betrag ($\mathbb{M}$)	0	1	2	3	4	5	6	7	8	9	10
Anzahl Münzen	0	1	2	1	1	1	2	2	2	2	2

c. Sei $\mathrm{n}_{\mathbb{M}}(m)$ die minimale Anzahl Münzen, welche nötig ist, um den Betrag m darzustellen. Stellen Sie die Rekursionsgleichung für $\mathrm{n}_{\mathbb{M}}(m)$ auf!

$$\mathrm{n}_{\mathbb{M}}(m) = \begin{cases} \infty & \text{falls } m < 0, \\ \dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots & \text{falls } m = 0, \\ \dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots & \\ \dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots & \\ \dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots & \\ \dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots & \text{sonst} \end{cases}$$

[3 Punkte]

Lösung

$$\mathrm{n}_{\mathbb{M}}(m) = \begin{cases} \infty & \text{falls } m < 0, \\ 0 & \text{falls } m = 0, \\ 1 + \min \begin{cases} \mathrm{n}_{\mathbb{M}}(m-1), \\ \mathrm{n}_{\mathbb{M}}(m-3), \\ \mathrm{n}_{\mathbb{M}}(m-4), \\ \mathrm{n}_{\mathbb{M}}(m-5) \end{cases} & \text{sonst} \end{cases}$$

Zur Feier seiner errungenen Weltherrschaft will Dr. Meta möglichst viele seiner Feinde bekochen. Ihm stehen dabei zwei Rezepte zur Verfügung: Lasagne und Pizza.

- Dr. Meta hat t Tomaten und k Stücke Käse zur Verfügung. Alle weiteren Zutaten sind zur Genüge vorhanden.
- Von jeder Lasagne werden 3 Feinde satt, von jeder Pizza nur 2.
- Für jede Pizza benötigt Dr. Meta 2 Tomaten und 4 Stücke Käse.
- Für jede Lasagne benötigt Dr. Meta 3 Tomaten und 3 Käse.
- Dr. Meta kann nur ganze Pizzen und ganze Lasagnen zubereiten.
- Dr. Meta muss nicht alle vorhandenen Zutaten verbrauchen.

d. Seien $t = 8$ Tomaten und $k = 11$ Stücke Käse vorhanden. Wie viele Feinde kann Dr. Meta mit diesen Zutaten *maximal* satt bekommen? Begründen Sie Ihre Lösung kurz! [2 Punkte]

Lösung

Die Zubereitung einer Pizza und zweier Lasagnen sättigt acht Personen. Hierbei handelt es sich um das *globale* Optimum, da

(a) jede Tomate maximal eine Person sättigt und diese Lösung alle Tomaten aufbraucht, bzw.

(b) wir den Suchraum erschöpfend abgesucht und keine bessere Lösung gefunden haben.

Hinweis: Es ist wichtig, die *globale* Optimalität der Lösung zu zeigen, nicht nur die *lokale*. Das Problem ist nicht greedy lösbar. Argumente wie "Lasagnen sind effizienter", oder "eine Lasagne durch Pizza ersetzen sättigt weniger Personen" funktionieren daher nicht.

e. Modellieren Sie das Maximierungsproblem als ILP (Integer Linear Program). Sei dabei x_P die Anzahl zubereiteter Pizzen und x_L die Anzahl zubereiteter Lasagnen.

maximiere ....

so dass $x_P, x_L \in \mathbb{Z}$

....

....

....

....

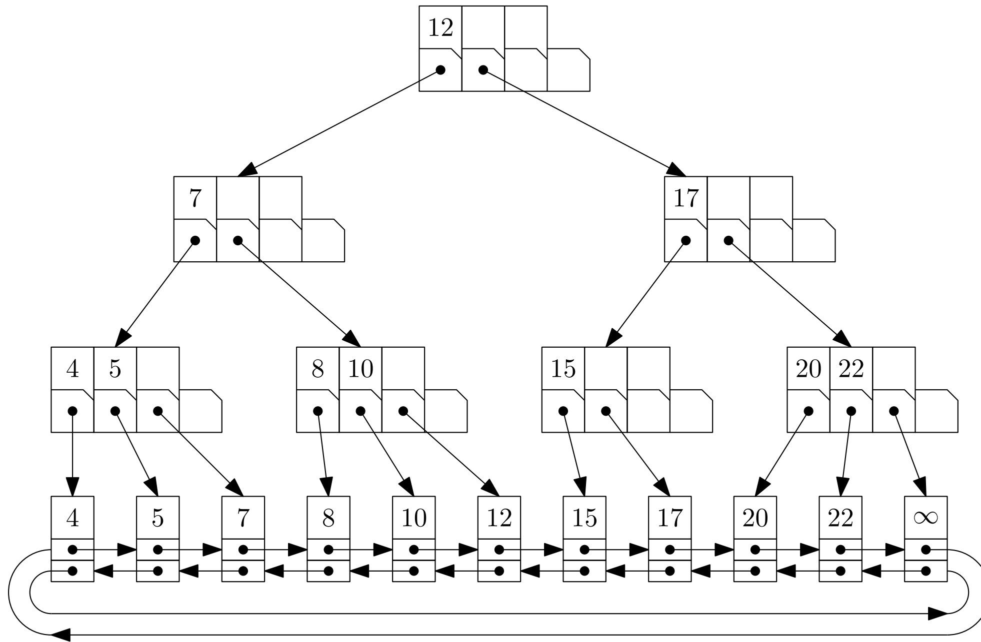
[4 Punkte]

Lösung

$$
\begin{aligned}
\text{maximiere} \quad & 2x_P + 3x_L \\
\text{so dass} \quad & x_P, x_L \in \mathbb{Z} \\
& x_L, x_P \geq 0 \\
& 2x_P + 3x_L \leq t \\
& 4x_P + 3x_L \leq k
\end{aligned}
$$

Aufgabe 5. Suchbäume und Heaps [11 Punkte]

a. Gegeben sei der unten abgebildete $(2,4)$ Baum. Zeichnen Sie den neuen Zustand des Baums nachdem die Operation `remove(15)` ausgeführt wurde. [2 Punkte]



Lösung

Da $\underbrace{u'.d}_{=3} + \underbrace{a-1}_{=1} \leq \underbrace{b}_{=4}$ rufen wir `fuse` auf und propagieren nach oben

b. Wieso gibt es keinen (a,b)-Baum mit $a = 6$ und $b = 7$? [1 Punkt]

Lösung

Laut Vorlesung muss $b \geq 2a - 1$ gelten. Diese Bedingung wäre bei $a = 6$ und $b = 7$ verletzt.

c. Gegeben sei ein $(2,4)$-Suchbaum, der die Schlüssel nur im Suchbaum selber, nicht aber als zusätzlich explizit sortierte Liste speichert. Geben Sie einen Algorithmus mit linearer Laufzeit an, welcher den Baum traversiert und dabei alle Schlüssel in sortierter Reihenfolge ausgibt. Verwende Sie folgende Schnittstelle zur Traversierung:

- Die Baumwurzel ist Knoten W.

- Die Funktion `ntesKind(`K`, `N`)` gibt den N-ten Kindsknoten von Knoten K zurück ($N \in \{0,1,2,3\}$), oder `NIL`, falls kein N-ter Kindsknoten existiert.
- Die Funktion `nterWert(`K`, `N`)` gibt den N-ten in K gespeicherten Schlüsselwert zurück ($N \in \{0,1,2\}$), oder ∞, falls kein N-ter Wert existiert.

Die Schlüsselwerte eines Knotens K sind nach N aufsteigend gespeichert, also

$$\texttt{nterWert(K, 0)} \leq \texttt{nterWert(K, 1)} \leq \texttt{nterWert(K, 2)}.$$

[3 Punkte]

Lösung

Wir führen eine modifizierte Tiefensuche auf dem Suchbaum aus, beginnend bei W: Für jeden Knoten betrachten wir `ntesKind(`K`, `N`)` in Indexreihenfolge, und steigen rekursiv in die Kindknoten ab.
Wenn kein K-tes Kind existiert, wird `nterWert(`K`, `N`)` aufgerufen. Ist der zurückgegebene Wert $\neq \infty$, wird er ausgegeben. Da der Baum ein Suchbaum ist und die Kinder in Schlüsselreihenfolge betrachtet werden, ist die Ausgabereihenfolge bereits korrekt.
Haben wir alle Kinder eines Knoten traversiert, kehrt der rekursive Aufruf zurück.
Die Tiefensuche benötigt Laufzeit $\mathcal{O}(n+m)$, da $m = n-1$ ist die Laufzeit also linear in der Suchbaumgröße.

d. Was ist die minimale und maximale Anzahl an Elementen in einem binären Heap der Höhe h?
Hinweis: Ein Heap mit nur einem Element habe hier die Höhe 0; die Höhe ist also die Länge des längsten Pfades von einem Blatt zur Wurzel. [1 Punkt]

Lösung

Ein Heap der Höhe h enthält mindestens $(\sum_{i=0}^{h-1} 2^i) + 1 = 2^h$ und höchstens $\sum_{i=0}^{h} 2^i = 2^{h+1} - 1$ Elemente.

e. Interpretieren Sie ein aufsteigend sortiertes Array als implizite Repräsentation eines Binärbaums. Erfüllt ein solcher Baum immer die Heapeigenschaft? Begründen Sie Ihre Antwort. [1 Punkt]

Lösung

Ja. Der Wert eines Kindes ist immer größer als der Wert des Elternteils, daher ist die Heapeigenschaft an jedem Knoten erfüllt.

f. Das Array ~~[4, 9, 12, 10, 6, 13]~~ [4, 9, 12, 10, 16, 13] sei ein binärer Heap. Zeichnen Sie diesen wie in der Vorlesung als Binärbaum. Führen Sie dann in gegebener Reihenfolge folgende Operationen aus. Geben Sie das *Endergebnis* als Array an.

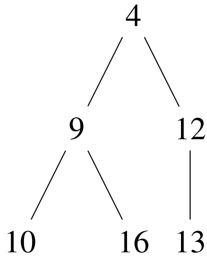
i) `deleteMin()`

ii) `insert(1)`

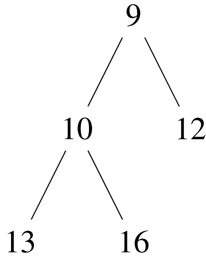
[3 Punkte]

Lösung

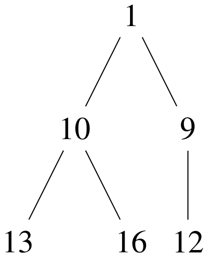
Initialer Zustand



nach `deleteMin()`[1]



nach `insert(1)`[1]



[1] Nicht gefordert

Endergebnis als Array:

1	10	9	13	16	12

Lösungsvorschlag

Punkte

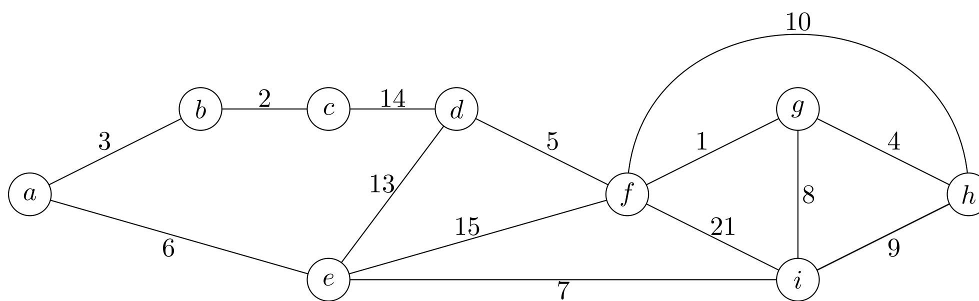
Aufgabe 6. Minimale Spannbäume [11 Punkte]

a. Betrachten Sie den angegeben Graphen mit den annotierten Kantengewichten. Geben Sie jeweils an, in welcher Reihenfolge die Kanten des Graphens zum minimalen Spannbaum hinzugefügt werden, wenn dieser

i) mit Jarnik-Prim's Algorithmus (ausgehend von Startknoten a) und

ii) mit Kruskal's Algorithmus

berechnet wird. [3 Punkte]



Lösung

i) Jarnik-Prim: $\{a,b\},\{b,c\},\{a,e\},\{e,i\},\{g,i\},\{f,g\},\{g,h\},\{d,f\}$.

ii) Kruskal: $\{f,g\},\{b,c\},\{a,b\},\{g,h\},\{d,f\},\{a,e\},\{e,i\},\{g,i\}$.

Ab jetzt bezeichne $G = (V, E, c)$ immer einen ungerichteten, zusammenhängenden Graphen mit Kantengewichten $c\colon E \to \mathbb{R}_+$.

b. Beweisen Sie, dass ein ungerichteter, zusammenhängender Graph mit $|E| = |V| - 1$ Kanten ein Baum ist.
Ergänzung: Ein Baum sei definiert als ein ungerichteter, zusammenhängender und azyklischer Graph. [4 Punkte]

Lösung

Möglichkeit 1: Beweis durch Induktion
I.A.: $n = |V| = 1$: ein Graph mit $|V| = 1$ und $|E| = 0$ ist ein einzelner Knoten, also ein Baum.
I.V.: Alle ungerichteten, zusammenhängenden Graphen mit $|V| = n$ und $|E| = |V| - 1$ seien Bäume.
I.S.: Sei $|V'| = n + 1$ und $|E'| = |V'| - 1$. Dann gibt es einen Knoten u von Grad 1, denn

- es gibt keine isolierten Knoten, da der Graph zusammenhängend ist, und
- hätten alle Knoten mindestens Grad 2, wäre $|E'| \geq |V'|$.

Entfernen von u liefert einen Graphen G, für den die Induktionsvoraussetzung greift, welcher also ein Baum ist. Das Hinzufügen des Grad-1-Knoten u zum Baum G ergibt wieder einen Baum G'. Also ist auch jeder beliebige ungerichtete, zusammenhängende Graph $G' = (V', E')$ mit $|V'| = n + 1$ und $|E'| = |V'| - 1$ ein Baum.

Möglichkeit 2: Beweis durch Widerlegen der Gegenannahme
Sei G ein ungerichteter, zusammenhängender Graph mit $|E| = |V| - 1$ Kanten.

Ann.: G enthalte einen Kreis C_k, welcher $k \leq |V|$ Knoten enthält. Nach Handshake-Lemma enthält der Kreis auch mindestens k Kanten. Der Fall $k = |V|$ ist damit also für G ausgeschlossen.

Möglichkeit 2.1: Zähle Kanten außerhalb des Kreises
Für jeden Knoten $v \in V \setminus C_k$ gibt es einen Pfad $p = (v, w, \dots, c)$ für ein beliebiges $c \in C_k$, da G zusammenhängend ist. Die Kante $\{v, w\}$ ist dabei nicht in C_k enthalten. Es gibt also pro Knoten, welcher nicht in C_k enthalten ist, wenigstens eine Kante, welche nicht in C_k enthalten ist. Insgesamt gibt es damit mindestens $|V| - k$ Kanten, welche nicht in C_k enthalten sind. Also $|E| \geq k + |V| - k = |V| > |V| - 1$. Mit diesem Widerspruch kann kein Kreis in G existieren.

Möglichkeit 2.2: Kontrahiere den Kreis
Kontrahiere den Kreis C_k zu einem Knoten v_C, indem die k Kanten des Kreises entfernt werden und jegliche andere Kanten, welche inzident zu einem Knoten in C_k waren, zu v_C umgebogen werden.
Der entstehende Graph $G' = (V', E')$ enthält nun $|V'| = |V| - k + 1$ Knoten und $|E'| = |V| - k - 1$ Kanten. Damit kann G' nicht zusammenhängend sein, denn $|E'| < |V'| - 1$.
Betrachte zwei Knoten $u, v \in V'$, für welche kein Pfad von u nach v existiert. Nehme an, es gäbe einen Pfad p von u nach v in G. Enthält p keinen Knoten in C_k, so müsste p auch in G' existieren, was ein Widerspruch ist. Enthält p einen Teilpfad p', welcher in C_k enthalten ist, so kann man diesen Teilpfad durch v_C ersetzen, um einen Pfad von u nach v in G' zu erhalten, was ebenfalls ein Widerspruch ist. Also kann es in G keinen Pfad von u nach v geben, und G ist nicht zusammenhängend. Mit diesem Widerspruch kann kein Kreis in G existieren.

c. Für ein festes $k \in \mathbb{N}$ bezeichnen wir einen zusammenhängenden, ungerichteten Graphen G als *k-fast-Baum*, falls $|E| < |V| + k$. Geben Sie einen Algorithmus mit Laufzeit $O(k \cdot |E|)$ an, der minimale Spannbäume auf k-fast-Bäumen berechnet. Eine textuelle Beschreibung Ihres Algorithmus ist ausreichend. Sie müssen keinen Pseudocode formulieren. Begründen Sie die Laufzeit und Korrektheit Ihres Algorithmus. [4 Punkte]

Lösung

Wiederhole bis zu k mal: suche Kreis K in G (BFS oder DFS). Falls K existiert, entferne die schwerste Kante auf K aus G. Brich ab, sobald G kreisfrei ist. Die verbleibenden Kanten bilden einen MST.
Laufzeit: jede Iteration läuft in $O(|E|)$ und entfernt eine Kante, nach k Iterationen ist der Graph also kreisfrei.
Korrektheit: folgt direkt aus der Cycle Property.

Anm.: Es können auch im Zuge *einer* BFS oder DFS alle Kreise gefunden und eliminiert werden. Jedoch müssen dann die Parent-Pointer der Suche nach Entfernen einer Kante angepasst werden, um im weiteren Verlauf der Suche Kreise korrekt rekonstruieren zu können. Das ist auch in $O(|E|)$ pro entfernter Kante möglich.