Name:	Klausur-ID:
Vorname:	
Matrikelnummer:	

Lösungsvorschlag

Karlsruher Institut für Technologie
Institut für Theoretische Informatik

Prof. Jörn Müller-Quade

4. September 2017

Klausur Algorithmen I

Aufgabe 1.	Kleinaufgaben	15 Punkte
Aufgabe 2.	Maximales Produkt	8 Punkte
Aufgabe 3.	Kürzeste Wege	12 Punkte
Aufgabe 4.	(a,b)-Bäume	6 Punkte
Aufgabe 5.	Bipartite Graphen	9 Punkte
Aufgabe 6.	Dynamische Programmierung	10 Punkte

Bitte beachten Sie:

- Bringen Sie den Aufkleber mit Ihrem Namen, Ihrer Matrikelnummer und Ihrer Klausur-ID oben links auf dem Deckblatt an.
- Merken Sie sich Ihre Klausur-ID und schreiben Sie auf **alle Blätter** der Klausur und Zusatzblätter Ihre Klausur-ID und Ihren Namen.
- Die Klausur enthält 19 Blätter. Die Bearbeitungszeit beträgt 120 Minuten.
- Die durch Übungsblätter gewonnenen Bonuspunkte werden erst nach Erreichen der Bestehensgrenze hinzugezählt. Die Anzahl der Bonuspunkte entscheidet nicht über das Bestehen der Klausur.
- Als Hilfsmittel ist ein beidseitig handbeschriebenes DIN-A4 Blatt zugelassen.
- Bitte kennzeichnen Sie deutlich, welche Aufgabe gewertet werden soll. Bei mehreren angegebenen Möglichkeiten wird jeweils die schlechteste Alternative gewertet.

Aufgabe	1	2	3	4	5	6	Summe
max. Punkte	15	8	12	6	9	10	60
Punkte							

Bonuspunkte:	Summe:	**Note:**

Lösungsvorschlag

Aufgabe 1. Kleinaufgaben [15 Punkte]

Bearbeiten Sie die folgenden Aufgaben. Begründen Sie Ihre Antworten jeweils kurz. *Reine Ja/Nein-Antworten ohne Begründung geben keine Punkte.*

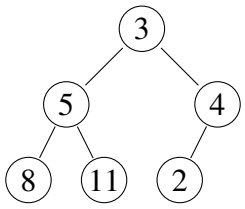
a. Gilt $2^{2n} \in O(2^n)$? Begründen Sie ihre Antwort! [1 Punkt]

Lösung

Nein, die Aussage gilt nicht, denn es gilt

$$\limsup_{n\to\infty} \frac{2^{2n}}{2^n} = \lim_{n\to\infty} \frac{2^n \cdot 2^n}{2^n} = \lim_{n\to\infty} 2^n = \infty.$$

b. Erfüllt der im Folgenden als Baum dargestellte Min-Heap die Heap-Eigenschaft aus der Vorlesung? Begründen Sie Ihre Antwort!



[1 Punkt]

Lösung

Nein, der Heap erfüllt die Heap-Eigenschaft nicht, da 2 ein Kind von 4 ist. Somit gilt nicht für alle Knoten v, dass $parent(v) \leq v$.

(weitere Teilaufgaben auf den nächsten Blättern)

Lösungsvorschlag

Fortsetzung von Aufgabe 1

c. Aus der Vorlesung ist bekannt, dass die Laufzeit von BucketSort in $O(n+K) = O(n)$ liegt, wobei n die Anzahl an Elementen und K eine konstante obere Schranke für alle zu sortierenden Zahlen ist. Es soll nun folgendermaßen ein Array A von beliebig großen Zahlen aus $\mathbb{N}$ sortiert werden, zu denen keine konstante obere Schranke bekannt ist:

1. Bestimme die größte Zahl $z_{\max}$ in A.
2. Lege ein Array mit $z_{\max}$ vielen Einträgen an.
3. Führe mithilfe des neuen Arrays und $z_{\max}$ BucketSort auf A aus.

Liegt die Laufzeit dieses Algorithmus ebenfalls in $O(n)$ (mit $n = |A|$)? Begründen Sie Ihre Antwort! [2 Punkte]

Lösung

Nein, da $z_{\max}$ nicht konstant ist, liegt die Laufzeit nicht in $O(n)$. Insbesondere kann $z_{\max}$ von $|A| = n$ abhängen. Da die Zahlen beliebig groß werden können, kann beispielsweise der Fall $z_{\max} = |A|^2 = n^2$ eintreten. Somit bräuchten wir bereits im zweiten Schritt $\Omega(n^2)$ Rechenschritte, um das Bucket-Array zu initialisieren. Das Zusammensetzen der Buckets zu einer Lösung am Schluss von BucketSort würde ebenfalls $\Omega(n^2)$ Rechenschritte benötigen.

(weitere Teilaufgaben auf den nächsten Blättern)

Lösungsvorschlag

Fortsetzung von Aufgabe 1

d. Sortieren Sie das Array $\langle 4,9,26,23,10,3,8,30\rangle$ mit dem MergeSort-Algorithmus. Verwenden Sie zur Darstellung das Schema aus der Vorlesung und geben Sie bei jedem Schritt an, ob es sich um einen split- oder merge-Schritt handelt. [2 Punkte]

Lösung

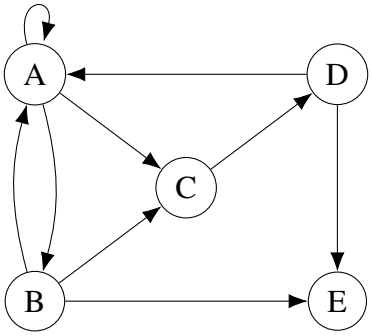
$\langle 4,9,26,23,10,3,8,30\rangle$
split
$\langle 4,9,26,23\rangle$, $\langle 10,3,8,30\rangle$
split
$\langle 4,9\rangle$, $\langle 26,23\rangle$, $\langle 10,3\rangle$, $\langle 8,30\rangle$
split
$\langle 4\rangle$, $\langle 9\rangle$, $\langle 26\rangle$, $\langle 23\rangle$, $\langle 10\rangle$, $\langle 3\rangle$, $\langle 8\rangle$, $\langle 30\rangle$
merge
$\langle 4,9\rangle$, $\langle 23,26\rangle$, $\langle 3,10\rangle$, $\langle 8,30\rangle$
merge
$\langle 4,9,23,26\rangle$, $\langle 3,8,10,30\rangle$
merge
$\langle 3,4,8,9,10,23,26,30\rangle$

(weitere Teilaufgaben auf den nächsten Blättern)

Lösungsvorschlag

Fortsetzung von Aufgabe 1

e. Geben Sie für den folgenden gerichteten Graphen eine Adjazenzmatrix und ein Adjazenzfeld an, welche den Graphen repräsentieren. Markieren Sie dabei eindeutig, welche/r Spalte/Zeile/Eintrag für welchen Knoten steht!



[2 Punkte]

Lösung

Adjazenzmatrix:

	A	*B*	*C*	*D*	*E*
A	1	1	1	0	0
B	1	0	1	0	1
C	0	0	0	1	0
D	1	0	0	0	1
E	0	0	0	0	0

Adjazenzfeld:

	A	B	C	D	E
V:	1	4	7	8	10

	1	2	3	4	5	6	7	8	9	10
E:	A	B	C	A	C	E	D	A	E	

f. Im Folgenden betrachten wir Hashing mit linearer Suche und Hashing mit verketteten Listen. Fügen Sie die Werte a, b, c, d, e, f mittels der unten angegebenen Hashwerte in der angegebenen Reihenfolge ein. Fügen Sie beim Hashing mit verketteten Listen die Elemente immer am *Anfang* der jeweiligen Liste ein! Verwenden Sie die vorgedruckten Tabellen - pro Verfahren sind zwei Tabellen angegeben (für Korrekturen etc.). Machen Sie deutlich, welche Tabelle Ihre Lösung enthält (beispielsweise indem Sie die nicht zu wertende Lösung eindeutig durchstreichen)! Bei mehreren ausgefüllten Tabellen wird die jeweils schlechtere Tabelle gewertet. Wenn Sie eine nicht vorgedruckte Tabelle benutzen, machen Sie kenntlich, für welches Verfahren Sie diese verwendet haben. Die Hashwerte der einzufügenden Elemente sind:

$$h(a) = 0, \quad h(b) = 7, \quad h(c) = 8, \quad h(d) = 7, \quad h(e) = 0, \quad h(f) = 3$$

[2 Punkte]

Lösung

Hashing mit linearer Suche:

0	1	2	3	4	5	6	7	8	9
a	e		f				b	c	d

Hashing mit verketteten Listen:

0	1	2	3	4	5	6	7	8	9
$e \to a$			f				$d \to b$	c	

(weitere Teilaufgaben auf den nächsten Blättern)

Lösungsvorschlag

Fortsetzung von Aufgabe 1

g. Geben Sie eine optimale Lösung für das Rucksackproblem mit Kapazität $M = 40$ und den Gegenständen A, B, C, D und E an, welche jeweils die folgenden Kosten und Profite haben:

	A	B	C	D	E
Kosten	10	20	15	20	35
Profit	15	10	14	12	20

[2 Punkte]

Lösung

Die optimale Lösung besteht aus den Gegenständen A und C mit den Gesamtkosten $25 \leq 40$ und dem Gesamtprofit 29.

h. Betrachten Sie für $n \in \mathbb{N}$ die folgende Rekurrenz $T(n)$:

$$T(n) = \begin{cases} 1 & \text{falls } n < 4, \\ 2 \cdot T(n-1) + T(n-2) + T(n-3) + n & \text{falls } n \geq 4. \end{cases}$$

Beweisen Sie per vollständiger Induktion, dass $T(n) \in \mathrm{O}(n!)$ gilt. [3 Punkte]

Lösung

Induktionsanfang $n \leq 4$**:** $T(1) = 1 = 1!$, $T(2) = 1 < 2 = 2!$, $T(3) = 1 < 6 = 3!$ und

$$T(4) = 2 \cdot T(3) + T(2) + T(1) + 4 = 8 \leq 24 = 4!$$

Induktionsvoraussetzung: Für ein $n \in \mathbb{N}$, $n \geq 4$, gilt $T(k) \leq k! \quad \forall k \leq n$.
Induktionsschluss $n \rightsquigarrow n+1$**:**

$$\begin{aligned} T(n+1) &= 2 \cdot T(n) + T(n-1) + T(n-2) + (n+1) \\ &\overset{IV}{\leq} 2 \cdot n! + (n-1)! + (n-2)! + (n+1) \\ &\overset{n \geq 4}{\leq} 2 \cdot n! + n! + n! + n! \\ &= 5 \cdot n! \\ &\overset{n \geq 4}{\leq} (n+1) \cdot n! \\ &= (n+1)! \end{aligned}$$

Lösungsvorschlag

Aufgabe 2. Maximales Produkt [8 Punkte]

Gegeben sei ein Array $A[1...n]$ mit $n \geq 2$ und $a[i] \in \mathbb{Z}$. Gesucht sind zwei Zahlen mit dem *maximalen* Produkt und unterschiedlichem Index in A.

a. Geben Sie für das Eingabearray das maximale Produkt an und markieren Sie die beiden Zahlen: [1 Punkt]

8	-8	10	1	-12	-2	6	-6	4	-9

Antwort: 108

b. Betrachten Sie folgenden Algorithmus, der ein Array $A[1...n]$ als Eingabe erhält und die Zahlen mit maximalem Produkt berechnet:

```
Function maxProduct(A : Array [1..n] of ℤ) : (ℤ, ℤ)
    p_max := −∞
    i_max := 1, j_max := 2
    i := 1, j := 2
    while i ≤ n − 1 do
        j := i + 1
        while j ≤ n do
            invariant: (i_max, j_max) = arg max_{k,l} {A[k] · A[l] | k < l ∧ (k < i ∧ l ≤ n) ∨ (k = i ∧ l < j)}
            if p_max < A[i] · A[j] then
                p_max := A[i] · A[j]
                i_max := i, j_max := j
            j := j + 1
        i := i + 1
    return (A[i_max], A[j_max])
```

Beweisen Sie die Korrektheit des Algorithmus, in dem Sie an der mit $\otimes$ gekennzeichneten Stelle eine Schleifeninvariante angeben und diese beweisen. Geben Sie zusätzlich die Laufzeit des Algorithmus im O-Kalkül an. [3 Punkte]

Lösung

Der Algorithmus benötigt $O(n^2)$ Zeit. Wir beweisen die Korrektheit mit Hilfe der Invariante

$$(i_{\max}, j_{\max}) = \arg\max_{k,l} \{A[k] \cdot A[l] \mid k < l \wedge (k < i \wedge l \leq n) \vee (k = i \wedge l < j)\}. \quad (1)$$

Die Invariante gilt vor dem ersten Schleifendurchlauf, da laut Aufgabenstellung $n \geq 2$. In jedem Schleifendurchlauf müssen dann zwei Fälle unterschieden werden:

i. Wenn $p_{\max} < A[i] \cdot A[j]$, dann werden sowohl $p_{\max}$ als auch $i_{\max}$ and $j_{\max}$ aktualisiert, sodass die Invariante wieder erfüllt ist.

ii. Im Falle $p_{\max} \geq A[i] \cdot A[j]$ muss kein Update gemacht werden und (1) bleibt erfüllt.

Nach Beenden der beiden Schleifen gilt: $i = n$, $j = n + 1$ und (1). Folglich sind die beiden gesuchten Zahlen: $A[i_{\max}], A[j_{\max}]$.

Lösungsvorschlag

Fortsetzung von Aufgabe 2

c. Entwerfen Sie einen Algorithmus, der zur Lösung des Problems maximal $O(n)$ Zeit benötigt. Geben Sie den Algorithmus in Pseudocode an! Sie müssen die Laufzeit des Algorithmus nicht beweisen. [4 Punkte]

Lösung

Das Problem lässt sich in $O(n)$ Zeit wie folgt lösen:

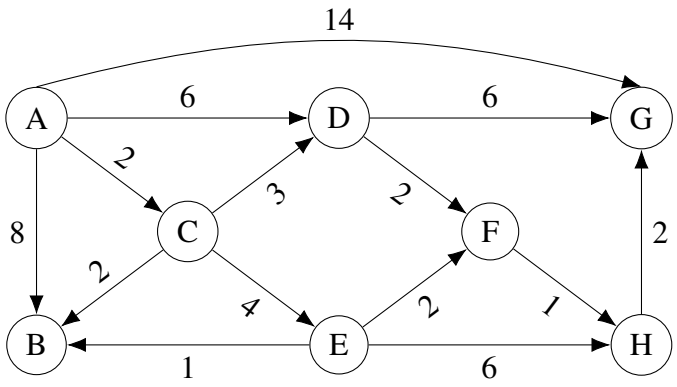
```
Function maxProductScan(A : Array [1..n] of ℤ) : (ℤ, ℤ)
    max1 := A[1], max2 := −∞
    min1 := A[1], min2 := ∞
    for i := 2 to n do
        if A[i] > max1 then
            max2 := max1
            max1 := A[i]
        else if A[i] > max2 then
            max2 := A[i]
        if A[i] < min1 then
            min2 := min1
            min1 := A[i]
        else if A[i] < min2 then
            min2 := A[i]
    if max1 * max2 > min1 * min2 then
        return (max1, max2)
    else
        return (min1, min2)
```

(weitere Teilaufgaben auf den nächsten Blättern)

Lösungsvorschlag

Aufgabe 3. Kürzeste Wege [12 Punkte]

a. Führen Sie Dijkstras Algorithmus auf dem folgenden Graphen ausgehend vom Knoten A aus. Geben Sie die Reihenfolge der Knoten an, in der sie als *scanned* markiert werden. Geben Sie außerdem für jeden Knoten $v \in V$ ***alle*** Zwischenwerte $d[v]$ und $parent[v]$ an, die während der Ausführung des Algorithmus angenommen werden. [5 Punkte]



Lösung

scanned-Reihenfolge: A, C, B, D, E, F, H, G

Knoten	$d[\cdot]$	$parent[\cdot]$
A	0	A
B	$\infty, 8, 4$	$\perp, A, C$
C	$\infty, 2$	$\perp, A$
D	$\infty, 6, 5$	$\perp, A, C$
E	$\infty, 6$	$\perp, C$
F	$\infty, 7$	$\perp, D$
G	$\infty, 14, 11, 10$	$\perp, A, D, H$
H	$\infty, 12, 8$	$\perp, E, F$

(weitere Teilaufgaben auf den nächsten Blättern)

Lösungsvorschlag

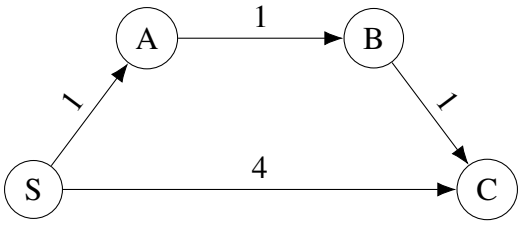
Fortsetzung von Aufgabe 3

Im Folgenden betrachten wir ausschließlich zusammenhängende und gerichtete Graphen mit positiven und ganzzahligen Kantengewichten größer 0. Wir erweitern das Problem der kürzesten Wege. In den folgenden Aufgaben b), c) und d) seinen die ***Kosten*** eines Weges im Graphen definiert als die Summe der Kantengewichte plus die Anzahl der Kanten des Weges. Gesucht sind nun ausgehend von einem Startknoten S die Wege zu allen anderen Knoten mit jeweils *minimalen* Kosten.
Hinweis: Verwechseln Sie nicht die *Kosten* eines Weges mit seiner Länge!

b. Beweisen Sie, dass Dijkstras Algorithmus nicht zwingend die Wege mit minimalen Kosten berechnet. Geben Sie dazu einen Graphen mit maximal 6 Knoten als Gegenbeispiel an. Erklären Sie kurz, weshalb es sich bei Ihrem Graphen um ein Gegenbeispiel handelt. [2 Punkte]

Lösung

Man betrachte den folgenden Graphen:



Dijkstras Algorithmus würde ausgehend von S den Weg $\langle S, A, B, C\rangle$ nach C berechnen. Dieser ist für Dijkstras Algorithmus der kürzeste Weg, da die Summe der Kantengewichte gleich 3 und damit kleiner als das Gewicht 4 der Kante von S nach C ist. Die Kosten dieses Weges sind aber $3+3=6$. Die Kante von S nach C hätte dagegen die Kosten 5.

(weitere Teilaufgaben auf den nächsten Blättern)

Lösungsvorschlag

Fortsetzung von Aufgabe 3

c. Entwerfen Sie einen Algorithmus, der bei Eingabe eines zusammenhängenden gerichteten Graphens $G = (V, E)$, einer Gewichtsfunktion $c : E \to \mathbb{N}$ mit $\forall e \in E : c(e) > 0$ und eines Knotens $S \in V$ die Wege mit minimalen Kosten in G ausgehend von S berechnet. Sie können dazu davon ausgehen, dass Ihnen eine Funktion $dijkstra(G, S, c)$ zur Verfügung steht, die auf dem Graphen G Dijkstras Algorithmus ausgehend von S berechnet. Die Laufzeit ihres Algorithmus soll im O-Kalkül nicht langsamer als die Laufzeit von Dijkstras Algorithmus sein. Für diese Aufgabe ist kein Pseudocode nötig, es genügt wenn Sie Ihren Algorithmus eindeutig beschreiben. Sie müssen die Laufzeit Ihres Algorithmus nicht beweisen. [3 Punkte]

Lösung

Der Algorithmus erhält als Eingabe einen Graphen $G = (V, E)$, eine Gewichtsfunktion c und einen Knoten S. Er erzeugt eine neue Gewichtsfunktion $c'(e) := c(e) + 1$. Dies bildet die Kosten der Kanten ab und sorgt dafür, dass die Anzahl Kanten eines Weges beim Berechnen des kürzesten Weges berücksichtigt wird. Der Algorithmus führt nun $dijkstra(G, S, c')$ aus und gibt die Ausgabe davon als seine eigene Ausgabe aus.

Das Erhöhen der Kantengewichte kann in $O(|E|)$ erfolgen (oder schneller – je nach Implementierung der Kostenfunktion), somit ist die Gesamtlaufzeit des Algorithmus asymptotisch nicht schlechter als Dijkstras Algorithmus, da dieser in $\Omega(|E|)$ liegt. (Die Laufzeitanalyse ist hier nur zur Vollständigkeit angegeben und war zur Lösung der Aufgabe nicht notwendig)

d. Beweisen Sie die Korrektheit Ihres Algorithmus zu Berechnung von Wegen mit minimalen Kosten aus Teilaufgabe c). [2 Punkte]

Lösung

Aus der Vorlesung ist bekannt, dass Dijkstras Algorithmus für Graphen mit positiven Kantengewichten die kürzesten Wege berechnet. Der obige Algorithmus berechnet also für alle Knoten $X \neq S$ den Weg $W_{SX} = \arg\min_W \{\sum_{e \in W} c'(e) \mid W \text{ ist ein Weg von } S \text{ nach } X\}$. Für diesen gilt

$$\begin{aligned} W_{SX} &= \arg\min_W \{\sum_{e \in W} c'(e) \mid W \text{ ist ein Weg von } S \text{ nach } X\} \\ &= \arg\min_W \{\sum_{e \in W} (c(e) + 1) \mid W \text{ ist ein Weg von } S \text{ nach } X\} \\ &= \arg\min_W \{(\sum_{e \in W} c(e)) + |W| \mid W \text{ ist ein Weg von } S \text{ nach } X\}. \end{aligned}$$

Somit ist W_{SX} also der Weg mit den minimalen Kosten von S nach X in G und der Algorithmus berechnet das Gewünschte.

Lösungsvorschlag

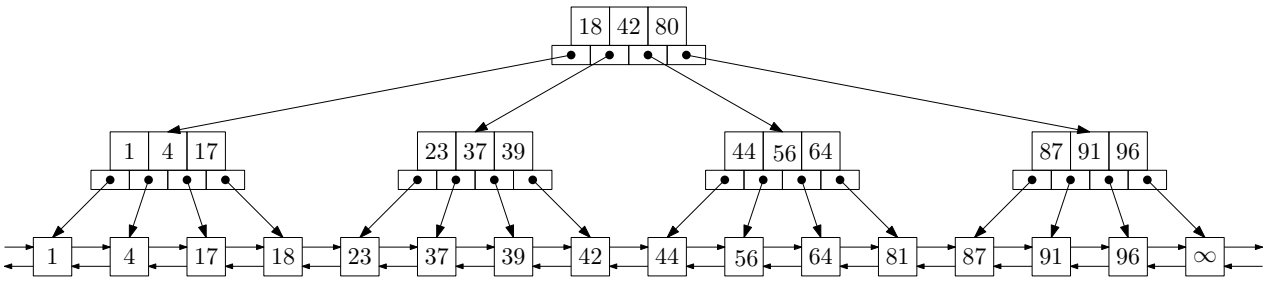
Aufgabe 4. (a,b)-Bäume [6 Punkte]

Für die im Folgenden auszuführenden (a,b)-Baum Operationen gelten folgende Regeln: Wenn ein Baumknoten gespalten werden muss bekommt der linke Knoten $\lfloor (b+1)/2 \rfloor$ Elemente und der rechte Knoten $\lceil (b+1)/2 \rceil$ Elemente. Für fuse/balance Operationen wird immer der linke Nachbarknoten verwendet.

a. Gegeben sei folgender (2,4)-Baum für die sortierte Folge

$$(1,4,17,18,23,37,39,42,44,56,64,81,87,91,96).$$

Führen Sie die nachfolgenden **locate**-Operationen durch. Markieren Sie hierzu den entsprechenden Suchpfad durch den Baum, indem Sie die jeweils betrachteten Kanten eindeutig markieren (z.B. Umkringeln der Kanten, deutliches Nachfahren der Kanten...). Tragen Sie die Rückgabewerte in die entsprechenden Felder ein. Liefern beide Aufrufe ein korrektes Ergebnis? Falls nein, begründen Sie, warum der Baum fehlerhaft ist.

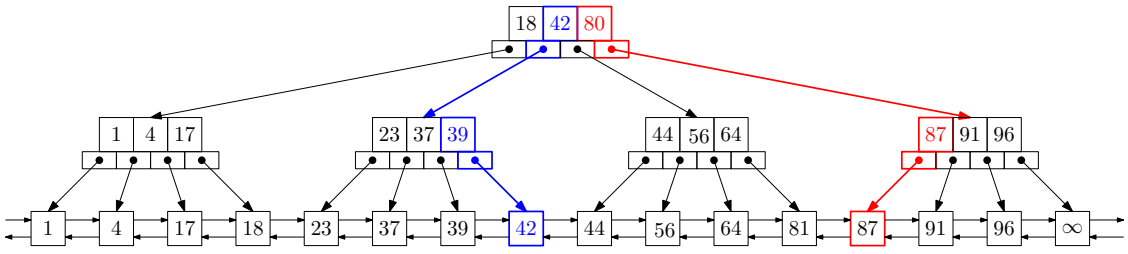


[2 Punkte]

locate(42): 42 **locate(81)**: 87

Begründung:

Lösung



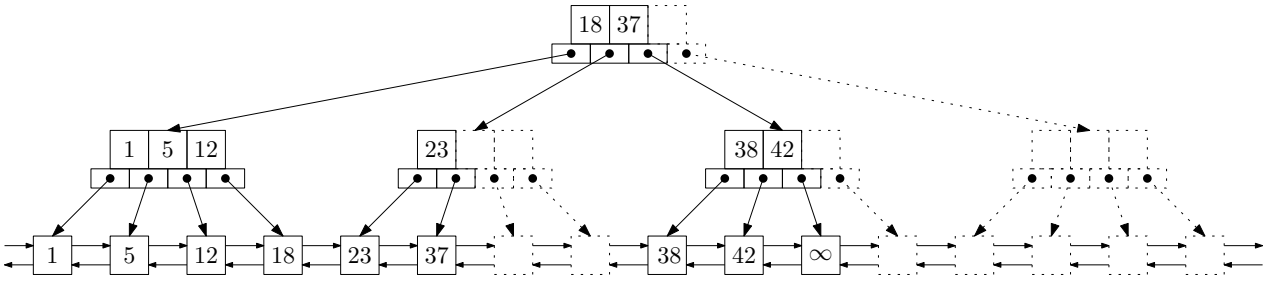
Der Aufruf von **locate(81)** liefert ein falsches Ergebnis zurück, da der letzte Splitter im Wurzelknoten des (2,4)-Baumes falsch ist. Statt 80 müsste dort 81 stehen.

(weitere Teilaufgaben auf den nächsten Blättern)

Lösungsvorschlag

Fortsetzung von Aufgabe 4

b. Gegeben sei folgender (2,4)-Baum für die sortierte Folge $(1, 5, 12, 18, 23, 37, 38, 42)$. Nicht verwendete Baumelemente sind gestrichelt dargestellt.

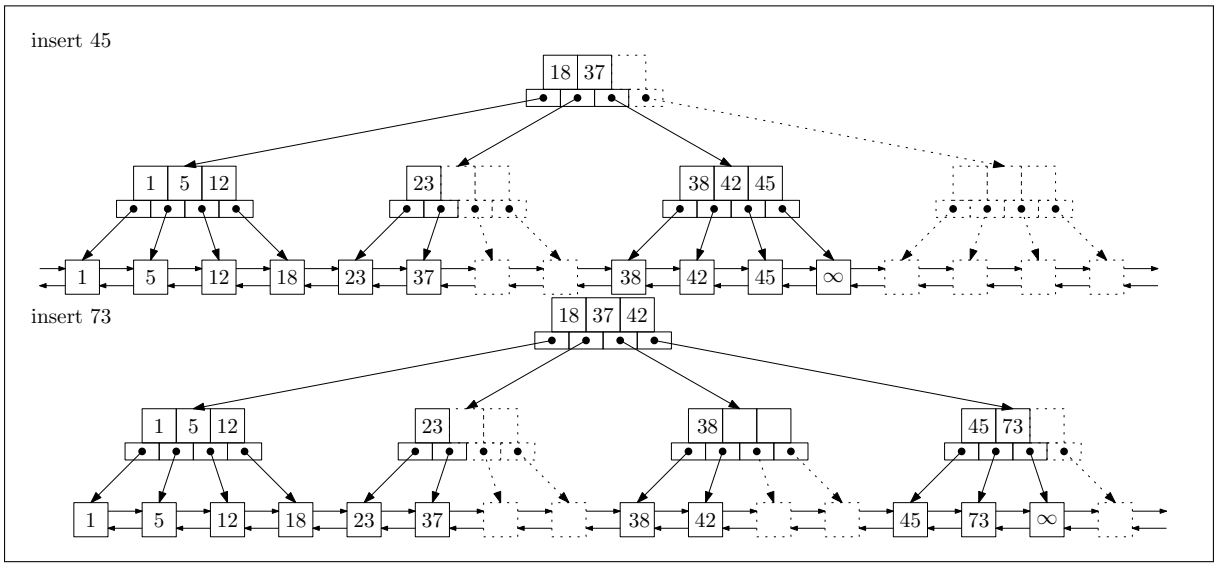


Führen Sie die geforderten Operationen in der angegebenen Reihenfolge durch:

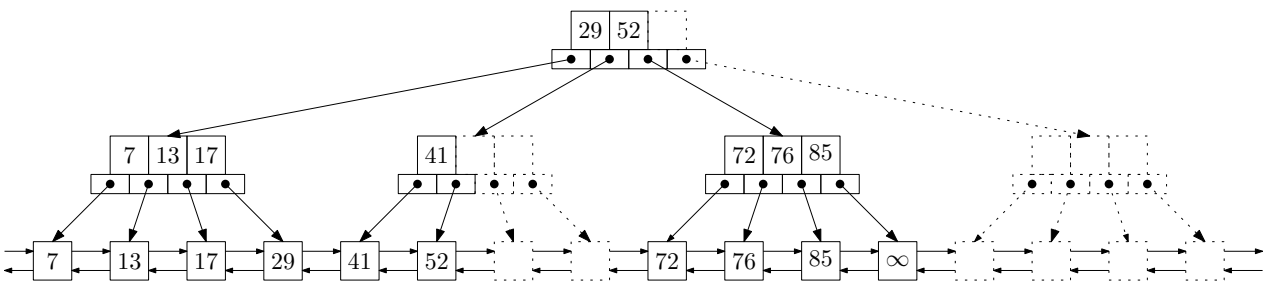
Einfügen von 45, danach Einfügen von 73

Zeichnen Sie den Endzustand des Baumes nach jeder der angegebenen Operationen in das vorgegebene Baumschema ein. Es sind weitere Baumschemata angegeben, die Sie zur Korrektur verwenden können - markieren Sie bitte deutlich, welche Operation dargestellt wird und streichen Sie eindeutig die Bäume durch, die nicht gewertet werden sollen (ansonsten wird der schlechtere Baum gewertet). [2 Punkte]

Lösung



c. Gegeben sei folgender (2,4)-Baum für die sortierte Folge $(7, 13, 17, 29, 41, 52, 72, 76, 85)$. Nicht verwendete Baumelemente sind gestrichelt dargestellt.

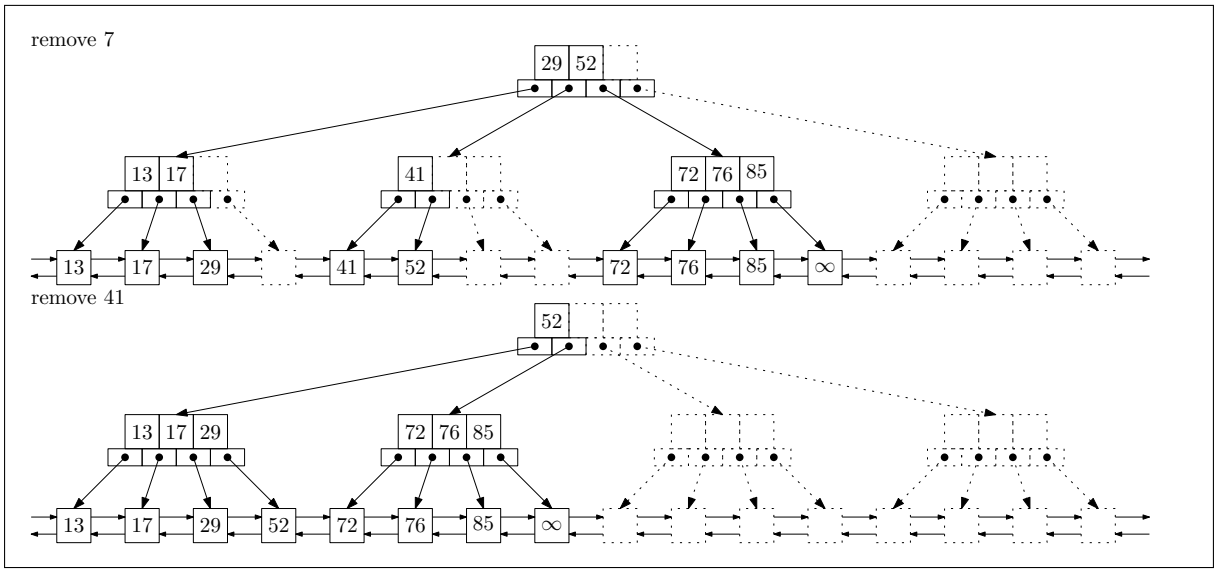


Führen Sie die geforderten Operationen in der angegebenen Reihenfolge durch:

Löschen von 7, danach Löschen von 41

Zeichnen Sie den Endzustand des Baumes nach jeder der angegebenen Operationen in das vorgegebene Baumschema ein. Es sind weitere Baumschemata angegeben, die Sie zur Korrektur verwenden können - markieren Sie bitte deutlich, welche Operation dargestellt wird und streichen Sie eindeutig die Bäume durch, die nicht gewertet werden sollen (ansonsten wird der schlechtere Baum gewertet). [2 Punkte]

Lösung



Lösungsvorschlag

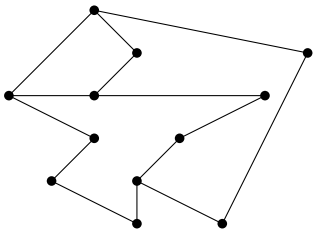
Aufgabe 5. Bipartite Graphen [9 Punkte]

Gegeben sei ein ungerichteter und zusammenhängender Graph $G = (V, E)$ mit ungewichteten Kanten. Ein Graph heißt *bipartit*, falls sich seine Knoten in zwei disjunkte Teilmengen A und B aufteilen lassen, sodass zwischen den Knoten innerhalb beider Teilmengen keine Kanten verlaufen. Das heißt für jede Kante $\{u, v\} \in E$ gilt entweder $u \in A$ und $v \in B$ oder $u \in B$ und $v \in A$.
Im Folgenden betrachten wir zwei Möglichkeiten zu prüfen, ob ein gegebener Graph bipartit ist:

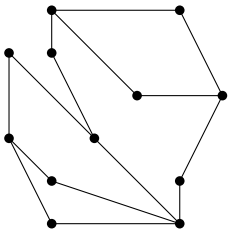
1. Ein Graph ist bipartit, genau dann wenn er *zweifärbbar* ist, das heißt, wenn es eine Abbildung $f : V \to \{0, 1\}$ gibt, so dass $\forall \{u, v\} \in E : f(u) \neq f(v)$. Es dürfen also keine zwei benachbarten Knoten die gleiche „Farbe“ 0 bzw. 1 haben.

2. Ein Graph ist bipartit, genau dann wenn er keinen Kreis *ungerader* Länge, d.h. mit einer ungeraden Anzahl an Kanten, enthält.

a. Prüfen Sie, ob folgende Graphen bipartit sind. Geben Sie für bipartite Graphen eine Zweifärbung an, indem sie die Knoten entsprechend markieren (beispielsweise durch Umkreisen oder Durchkreuzen). Kennzeichnen Sie in nicht bipartiten Graphen einen ungeraden Kreis. [1 Punkte]

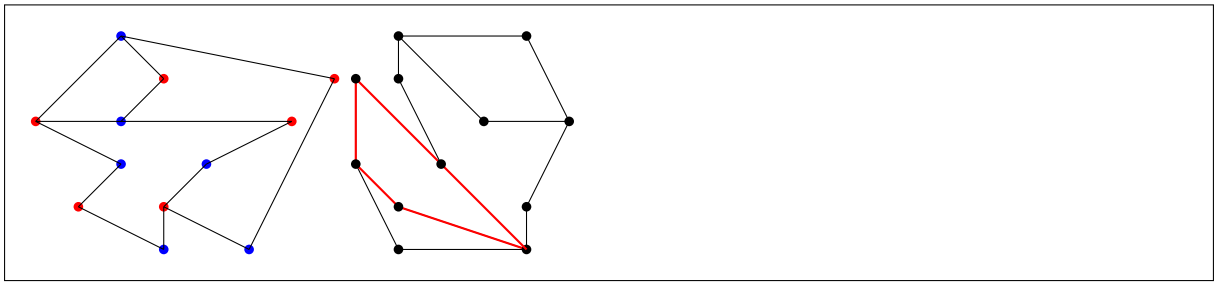


(a)



(b)

Lösung



(weitere Teilaufgaben auf den nächsten Blättern)

Lösungsvorschlag

Fortsetzung von Aufgabe 5

b. Entwerfen Sie einen Algorithmus in Pseudocode, der mittels Tiefensuche in $O(|V|+|E|)$ Zeit prüft, ob G *zweifärbbar* ist. Verwenden Sie hierfür das folgende Tiefensuchschema, indem Sie an den gekennzeichneten Stellen Pseudocode hinzufügen. Sie müssen nicht an allen gekennzeichneten Stellen Code einfügen. Innerhalb dieser Aufgabe dürfen Sie keine weiteren Algorithmen, Subroutinen, Methoden o.ä. aufrufen - geben Sie alle Ihre Schritte explizit in Pseudocode an! [4 Punkte]

Tiefensuchschema für $G=(V,E)$ mit Startknoten $s \in V$

$\forall v \in V$: marked$[v]$:= `false`
$\forall v \in V$: color$[v] := \perp$
color$[s] := 0$
bipartite := `true`
marked$[s]$:= `true`
DFS$(\perp, s)$
`return` bipartite

`Procedure` DFS$(u, v$: NodeId$)$
  `foreach` $(v,w) \in E$ **`do`**
    `if` not marked$[w]$ **`then`**
      color$[w] :=$ (color$[v]+1$) mod 2
      marked$[w]$:= `true`
      DFS(v,w)
    `else if` color$[w]=$ color$[v]$ **`then`**
      bipartite := `false`

c. Entwerfen Sie einen Algorithmus in Pseudocode, der mittels Breitensuche in $O(|V|+|E|)$ Zeit prüft, ob G einen Kreis ungerader Länge enthält. Verwenden Sie hierfür das folgende Breitensuchschema, indem Sie an den gekennzeichneten Stellen Pseudocode hinzufügen. Sie müssen nicht an allen gekennzeichneten Stellen Code einfügen. Innerhalb dieser Aufgabe dürfen Sie keine weiteren Algorithmen, Subroutinen, Methoden o.ä. aufrufen - geben Sie alle Ihre Schritte explizit in Pseudocode an! [4 Punkte]

Breitensuchschema für $G=(V,E)$ mit Startknoten $s \in V$

$d=\langle\infty,\ldots,\infty\rangle$: **`Array of`** $\mathbb{N}_0 \cup \{\infty\}$; $d[s] := 0$
parent$=\langle\perp,\ldots,\perp\rangle$: **`Array of`** NodeId; parent$[s] := s$
$Q = \langle s\rangle, Q' = \langle\rangle$: **`Set of`** NodeId
bipartite := `true`
`for` $(\ell := 0;\ Q \neq \langle\rangle;\ \ell{+}{+})$
  `foreach` $u \in Q$ **`do`**
    `foreach` $(u,v) \in E$ **`do`**
      `if` parent$[v] = \perp$ **`then`**
        $Q' := Q' \cup \{v\}$; $d[v] := \ell+1$; parent$[v] := u$
      `else if` $d[v] = d[u]$ **`then`**
        bipartite := `false`
  $(Q, Q') := (Q', \langle\rangle)$
`return` bipartite

Lösungsvorschlag

Aufgabe 6. Dynamische Programmierung [10 Punkte]

Die Levenshtein-Distanz ist eine Metrik zur Bestimmung der Ähnlichkeit zweier Zeichenketten. Dabei gibt $d(v,w)$ die minimale Anzahl an Edit-Operationen an, um v in w umzuwandeln (oder umgekehrt). Als Operationen sind dabei nur das Einfügen, Entfernen oder Ersetzen einzelner Zeichen zugelassen.
Beispielsweise ist $d(Aufgabe, Aufnahme) = 3$, da 3 Schritte notwendig sind, um ein Wort in das andere umzuwandeln:

Aufgabe	Ersetzen von g durch n
Aufnabe	Ersetzen von b durch h
Aufnahe	Einfügen von m
Aufnahme	

a. Bestimmen Sie die Levenshtein-Distanz der Wörter *Dynamisch* und *Stammtisch* und geben Sie eine mögliche minimale Sequenz von Operationen an, um ein Wort in das andere umzuwandeln. [2 Punkte]

Lösung

Die Levenshtein-Distanz beträgt 5:

Dynamisch	Ersetzen von D durch S
Synamisch	Ersetzen von y durch t
Stnamisch	Entfernen von n
Stamisch	Einfügen von m
Stammisch	Einfügen von t
Stammtisch	

b. Entwerfen Sie einen Algorithmus nach dem Entwurfsprinzip der dynamischen Programmierung, welcher die Levenshtein-Distanz $d(v,w)$ zweier Wörter v und w berechnet. Die Laufzeit des Algorithmus soll dabei in $O(|v| \cdot |w|)$ liegen. Für diese Aufgabe ist kein Pseudocode notwendig, es genügt, wenn Sie Ihren Algorithmus eindeutig beschreiben.
Hinweis: Verwenden Sie ein 2-dimensionales Array D der Größe $(|v|+1) \times (|w|+1)$ und speichern Sie in Eintrag $D_{i,j}$ die Lösung des Teilproblems $d(v[0..i], w[0..j])$. [5 Punkte]

Lösung

Wir definieren ein 2-dimensionales Array D der Größe $(|v|+1) \times (|w|+1)$, wobei $D_{i,j}$ der Levenshtein-Distanz der Prefixe $v[0..i]$ und $w[0..j]$ entspricht. Wir initialisieren $D_{0,j} := j$ für $0 \leq j \leq |w|$ und $D_{i,0} := i$ für $0 \leq i \leq |v|$.

$$D_{0,j} := j \qquad 0 \leq j \leq |w|$$
$$D_{i,0} := i \qquad 0 \leq i \leq |v|.$$

Anschließend setzen wir die restlichen Einträge des Arrays mittels

$$D_{i,j} := \min\left(D_{i-1,j-1} + (v_i \neq w_j), D_{i,j-1} + 1, D_{i-1,j} + 1\right).$$

Dabei entspricht der erste Term dem Ersetzen, falls die Zeichen an diesen Positionen unterschiedlich sind (+1), und keiner Operation, falls die Zeichen identisch sind (+0); der zweite Term entspricht dem Einfügen eines Zeichens; und der dritte Term entspricht dem Entfernen eines Zeichens. Das Ergebnis steht dann im Eintrag $D_{|v|,|w|} = d(v,w)$.

c. Analysieren Sie die Laufzeit und den Speicherverbrauch Ihres Algorithmus im O-Kalkül. [3 Punkte]

Lösung

Erstellen des Arrays benötigt Zeit $O(|v| \cdot |w|)$, Initialisierung der ersten Zeile und Spalte $O(|v| + |w|)$, Berechnung der restlichen Einträge jeweils $O(1)$; insgesamt ergibt sich also eine Laufzeit von $O(|v| \cdot |w|)$. Der Speicherbedarf liegt ebenfalls in $O(|v| \cdot |w|)$ für das Array.