Name:	Klausur-ID:
Vorname:	
Matrikelnummer:	

Lösungsvorschlag

Karlsruher Institut für Technologie
Institut für Theoretische Informatik

Prof. Dr. P. Sanders

28.7.2014

Klausur Algorithmen I

Aufgabe 1.	Kleinaufgaben	15 Punkte
Aufgabe 2.	Alle Wege	9 Punkte
Aufgabe 3.	Pseudocode-Analyse	5 Punkte
Aufgabe 4.	4-äre Heaps	12 Punkte
Aufgabe 5.	Shuffle	8 Punkte
Aufgabe 6.	Minimale Spannbäume	11 Punkte

Bitte beachten Sie:

- Als Hilfsmittel ist nur **ein** DIN-A4-Blatt mit Ihren **handschriftlichen** Notizen zugelassen.
- Merken Sie sich Ihre **Klausur-ID** auf dem Aufkleber für den Notenaushang.
- **Schreiben** Sie auf **alle Blätter** der Klausur und Zusatzblätter Ihre **Klausur-ID**.
- Die Klausur enthält 17 Blätter.
- Die durch Übungsblätter gewonnenen Bonuspunkte werden erst nach Erreichen der Bestehensgrenze hinzugezählt. Die Anzahl Bonuspunkte entscheidet nicht über das Bestehen.

Aufgabe		1	2	3	4	5	6	Summe
max. Punkte		15	9	5	12	8	11	60
Punkte	EK							
	ZK							

Bonuspunkte:	Summe:	**Note:**

Lösungsvorschlag

Aufgabe 1. Kleinaufgaben [15 Punkte]

a. Sie sind Software Engineer beim sozialen Netzwerk Giigle Minus. Ihr Chef möchte von Ihnen einen Algorithmus, der zu zwei gegebenen Personen p, p' des Netzwerks die gemeinsamen Freunde im Netzwerk ausgibt. Sie können davon ausgehen, dass das Netzwerk als ungerichter Graph $G = (\{p_1, p_2, p_3, \ldots, \}, E)$ in Adjazenzarray-Darstellung gegeben ist. Es gilt $\{p, p'\} \in E$ genau dann, wenn Person p mit Person p' in dem Netzwerk befreundet ist.

Skizzieren Sie einen Algorithmus, der das Problem für zwei gegebene Personen p, p' in erwartet $O(\deg(p) + \deg(p'))$ löst. Dabei bezeichnet $\deg(v)$ den Knotengrad eines Knotens v. Begründen Sie kurz die Laufzeit Ihres Algorithmus.
Hinweis: Beachten Sie, dass das Adjazenzarray nicht notwendigerweise sortiert ist und alle zusätzlichen Speicherzellen vor Verwendung initialisiert werden müssen. [3 Punkte]

Lösung

Man nehme eine Hashtabelle H mit verketteten Listen und einer zufälligen Hashfunktion aus einer universellen Familie. Dabei sei die Größe m der Hashtabelle so gewählt, dass $deg(p_1) + deg(p_2) = \Theta(m)$. Nun fügt man die Nachbarn/Freunde von Knoten p_1 in die Hashtabelle H ein. Anschließend prüft man für jeden Nachbarn des Knotens p_2, ob er in H enthalten ist. Falls dies so ist, wird der Nachbar ausgegeben. Insgesamt haben wir so $deg(p_1)$ insert() Operationen und $deg(p_2)$ contains() Abfragen. Wir erhalten damit die gewünschte Laufzeit.

Lösungsende

(weitere Teilaufgaben auf den nächsten Blättern)

Lösungsvorschlag

Fortsetzung von Aufgabe 1

b. Nennen Sie die asymptotische Anzahl von *Swaps* und *Vergleichen* im *best-case* und *worst-case* für Insertion Sort auf einem Array mit n verschiedenen Elementen, wenn die Einfügestelle mit *binärer Suche* bestimmt wird. [4 Punkte]

	best-case	worst-case
Swaps:	$\Theta(1)$	$\Theta(n^2)$
Vergleiche:	$\Theta(n \log n)$	$\Theta(n \log n)$

c. Notieren Sie die folgenden Funktionen aufsteigend sortiert nach asymptotischen Wachstum, und begründen Sie Ihre Behauptung.

$$f_1(n) = \frac{n^2}{\ln n^{2/3}} \qquad f_2(n) = n \cdot \ln n^{3/2} \qquad f_3(n) = \frac{3}{2} \cdot n^{3/2}$$

[3 Punkte]

Lösung

$f_2 = \Theta(n \log n) < f_3 = \Theta(n^{3/2}) < f_1$, denn $\limsup_{n\to\infty} \frac{f_3(n)}{f_2(n)} = \limsup_{n\to\infty} \frac{n^{1/2}}{\log n} = \infty$ und $\limsup_{n\to\infty} \frac{f_1(n)}{f_3(n)} = \limsup_{n\to\infty} \frac{n^{4/3}}{\ln n} = \infty$.

Lösungsende

(weitere Teilaufgaben auf den nächsten Blättern)

Lösungsvorschlag

Fortsetzung von Aufgabe 1

d. Beschreiben Sie kurz den Begriff *erwartete* Laufzeit. [1 Punkte]

Lösung

Als *erwartete* Laufzeit bezeichnet man den Erwartungswert der Laufzeit eines Algorithmus als Zufallsvariable über alle möglichen Eingaben.

Lösungsende

e. Tragen Sie die Namen der folgenden Algorithmen und Datenstruktur-Operationen aus der Vorlesung in die entsprechenden Felder nach ihrer asymptotischer *worst-case Zeitkomplexität*. [4 Punkte]

- *quickSelect()* auf einem Array der Länge n.
- *LSDRadixSort()* auf einem Array der Länge n, das 32-bit Ganzzahlen enthält.
- *splice()* auf einer doppelt verketteten Liste der Länge n.
- *buildAddressableHeap()* auf einem Array der Länge n.
- *isDAG()* auf einem gerichteten Graphen mit n Knoten und $4n$ Kanten.
- *bellmanFord()* auf einem gerichteten Graphen mit n Knoten und $2n$ Kanten.

$\Theta(1)$	splice()
$\Theta(\log n)$	
$\Theta(n)$	buildAddressableHeap(), isDAG(), LSDRadixSort()
$\Theta(n \log n)$	
$\Theta(n^2)$	quickSelect(), bellmanFord()

Lösungsvorschlag

Aufgabe 2. Alle Wege [9 Punkte]

a. Gegeben sei ein gerichteter Graph $G = (V,E)$ mit ungewichteten Kanten. Entwerfen Sie einen Algorithmus in Pseudocode, der in $O(|V|+|E|)$ für zwei gegebene Knoten $s,t \in V$ die *Anzahl der kürzesten Wege* von s nach t bestimmt. Sie können hierfür das folgende Breitensuchschema verwenden und an den gekennzeichneten Stellen Pseudocode hinzufügen, oder auf extra Blättern einen eigenen Algorithmus entwickeln. [4 Punkte]

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

```
d=⟨∞,...,∞⟩ : Array of ℕ0 ∪ {∞};   d[s] := 0
parent=⟨⊥,...,⊥⟩ : Array of NodeId;   parent[s] := s
Q = ⟨s⟩, Q' = ⟨⟩ : Set of NodeId
k=⟨0,...,0⟩ : Array of ℕ0;   k[s] := 1
for (ℓ := 0; Q ≠ ⟨⟩; ℓ++ )
        foreach u ∈ Q do
                foreach (u,v) ∈ E do
                        if parent[v] = ⊥ then
                                Q' := Q' ∪ {v};   d[v] := ℓ+1;   parent[v] := u
                                k[v] := k[u]
                        else if d[v] = ℓ+1 then
                                k[v] := k[u] + k[v]
        (Q,Q') := (Q',⟨⟩)
return [ k[t] ]
```

(weitere Teilaufgaben auf den nächsten Blättern)

Lösungsvorschlag

Fortsetzung von Aufgabe 2

b. Beweisen Sie die *Korrektheit* und *Laufzeit* Ihres Algorithmus aus Teilaufgabe b). [5 Punkte]

Lösung

Wir berechnen in $k[v]$ die Anzahl der kürzesten Wege von s nach $v \in V$ und betrachten für den Beweis den BFS-Baum, der durch die in s beginnende Breitensuche bestimmten wird. Dann stellen wir fest, dass alle "tree"-Kanten des Baum auf einem kürzesten s-v-Pfad liegen. Da dies ein Baum ist, gibt es von s zu jedem v genau einen kürzesten Weg im Baum. Das sind aber nicht alle kürzeste Pfade, denn weitere können durch andere (nicht-"tree") Kanten entstehen. Da $d[v]$ die BFS-Ebene des Knotens v ist, dann liegen keine Kanten $v \to w$ mit $d[v] \geq d[w]$ auf einem kürzesten s-v-Pfad, da bereits ein kürzerer Pfad zu w gefunden wurde. Es bleiben also nur noch Kanten $v \to w$ mit $d[v] + 1 = d[w]$, denn $d[v] + 1 > d[w]$ ist unmöglich. Bei solchen Kanten mit $d[v] + 1 = d[w]$ gibt es neben der tree-Kante $x \to w$ eine weitere Kante $v \to w$, über die w mit der gleichen Distanz erreichbar ist. Die Anzahl kürzester Wege nach w ist also $\sum_{v \in \text{pred}(w)} k[v]$, wobei $\text{pred}(w)$ alle Knoten mit $d[v] + 1 = d[w]$ sind.
Mit dieser rekursiven Formel bestimmen wir in a) ausgehend von $k[s] = 1$ alle $k[v]$ mit der Breitensuche. Initial wird $k[v] = 0$ für $v \in V \setminus \{s\}$ gesetzt und für alle betrachteten Kanten mit $d[v] + 1 = d[w]$ der Wert $k[w]$ um $k[v]$ erhöht, wobei die Werte $d[]$ in der laufenden Breitensuche bestimmt werden. Der Wert von $k[t]$ kann am Ende dann ausgegeben werden, für ein beliebiges $t \in V$.
Die Laufzeit ist die der Breitensuche, also $O(|V| + |E|)$.

Lösungsende

Lösungsvorschlag

Aufgabe 3. Pseudocode-Analyse [5 Punkte]

Gegeben sei der nachfolgende Pseudocode.

```
Function calculate(A : Array [1..n] of BigNumber) : BigNumber
    calculateRec(A, 1, n)

Function calculateRec(A : Array [1..n] of BigNumber, l : ℤ, r : ℤ) : BigNumber
    s := 0
    if l ≤ r then
        i := l
        while i ≤ r do
            if A[i] ≥ 0 then
                s := s ⊕ A[i]
            else
                s := s ⊖ A[i] ⊕ 2014728
            i := i + 1
        s := s ⊕ calculateRec(A, l + 1, r)
    return s
```

In den folgenden Teilaufgaben analysieren wir die *exakte Anzahl* der arithmetischen Operationen $\oplus$ (Addition) und $\ominus$ (Subtraktion) von `BigNumbers`, die ein Aufruf von calculate(A) für ein Array A von `BigNumber` mit n Elementen benötigt.

a. Charakterisieren Sie sowohl die *best-* als auch die *worst-case* Eingaben der Größe n für die Funktion calculate(). [1 Punkt]

Lösung

best-case-Eingaben enthalten nur positive Zahlen (≥ 0), da im Falle $A[i] \geq 0$ in der `while`-Schleife jeweils nur eine Operation ($\oplus$) durchgeführt wird. *worst-case*-Eingaben hingegen enthalten nur negative Zahlen, da in diesem Fall in jedem Durchlauf der `while`-Schleife *zwei* Operationen durchgeführt werden ($\oplus$ und $\ominus$). Alle anderen Operationen sind unabhängig von der Eingabe.

Lösungsende

(weitere Teilaufgaben auf den nächsten Blättern)

Lösungsvorschlag

Fortsetzung von Aufgabe 3

b. Geben Sie für *best-case* Eingaben eine Rekurrenz $T_{bc}(n)$ und für *worst-case*-Eingaben eine Rekurrenz $T_{wc}(n)$ an. *Hinweis:* Denken Sie auch an die Anfangswerte. [2 Punkte]

Lösung

$T_{bc}(n) = T_{bc}(n-1) + n + 1,\ T_{bc}(0) = 0$
$T_{wc}(n) = T_{wc}(n-1) + 2n + 1,\ T_{wc}(0) = 0$

Lösungsende

c. Bestimmen und beweisen Sie für $T_{wc}(n)$ eine geschlossene Form. [2 Punkte]

Lösung

Geschlossene Form für $T_{wc}(n)$:
$T_{wc}(n) = T_{wc}(n-1) + 2n + 1 = \sum_{k=1}^{n}(2k+1) = 2\frac{n(n+1)}{2} + n = n^2 + 2n$

Lösungsende

Lösungsvorschlag

Aufgabe 4. 4-äre Heaps [12 Punkte]

In der Vorlesung wurden binäre Heaps als Implementierung von Prioritätswarteschlangen behandelt. Im Folgenden betrachten wir 4-äre Heaps. Hier hat jeder Knoten maximal vier anstatt maximal zwei Kinder. Wie binäre Heaps werden auch 4-äre Heaps implizit repräsentiert, indem die Einträge ebenenweise in ein Array geschrieben werden. Sei ein solches Array $h[1..n]$ gegeben. Dann gibt die Funktion parent(i) den Elternknoten des Knotens i zurück, und child(i, j), $(1 \leq j \leq 4)$ gibt das j-te Kind des Knotens i. Daraus ergibt sich folgende
Invariante: $\forall i > 1 : h[\text{parent}(i)] \leq h[i]$

a. Für folgenden 4-ären Heap ist die Invariante verletzt. Markieren Sie die betroffene(n) Position(en) i, indem Sie ein **X** unter den Eintrag schreiben. [1 Punkte]

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
2	11	34	17	47	15	16	23	31	32	70	81	76	62	57	83

Lösung

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
2	11	34	17	47	15	16	23	31	32	70	81	76	62	57	83
									X						

Lösungsende

b. In einen anfangs leeren 4-ären Heap werden nacheinander folgende Elemente eingefügt:
92, 98, 48, 70, 22, 95, 76, 36, 17, 55, 94.
Anschließend wird eine deleteMin-Operation durchgeführt. Geben Sie den Zustand des Feldes h an, nach Ausführen

1. der ersten fünf insert-Operationen,
2. aller insert-Operationen sowie
3. der anschließenden deleteMin-Operation.

Geben Sie auch den Rückgabewert der deleteMin-Operation an. Nutzen Sie den freien Platz bzw. Konzeptpapier zum Berechnen und die drei vorbereiteten Felder zum Angeben Ihrer Lösungen. Beachten Sie die *Reihenfolge* dieser insert-Operationen. [3 Punkte]

1	2	3	4	5	6	7	8	9	10	11

1	2	3	4	5	6	7	8	9	10	11

1	2	3	4	5	6	7	8	9	10	11

Lösung

1	2	3	4	5	6	7	8	9	10	11
22	98	92	70	48						

1	2	3	4	5	6	7	8	9	10	11
17	22	55	70	48	98	95	76	36	92	94

Rückgabewert: 17

1	2	3	4	5	6	7	8	9	10	11
22	36	55	70	48	98	95	76	94	92	

Lösungsende

(weitere Teilaufgaben auf den nächsten Blättern)

Lösungsvorschlag

Fortsetzung von Aufgabe 4

c. Die Wurzel des Heaps sei in Ebene 0. Wieviele Elemente e_k sind in der k-ten vollständig gefüllten Ebene? Wieviele Elemente d_k sind in den Ebenen davor?
Hinweis: $\sum_{i=0}^{n} a^i = \frac{a^{n+1}-1}{a-1}$ für alle $a \in \mathbb{R}, n \in \mathbb{N}$. [1 Punkt]

Lösung

Die Anzahl der Elemente vervierfacht sich in jeder Ebene, daher sind in der k-ten Ebene 4^k Elemente. In den Ebenen davor sind $\sum_{j=0}^{k-1} 4^j = \frac{4^k-1}{3}$.

Lösungsende

d. Implementieren Sie die Funktionen parent(i) und child(i,j), indem Sie geschlossene Formeln angeben und herleiten.
Hinweis: Stellen Sie zunächst eine Formel auf, welche die Ebene eines Indizes zurückgibt. [5 Punkte]

Lösung

In der k-ten Ebene sind die Indexe i mit $\sum_{j=0}^{k-1} 4^j < i \leq \sum_{j=0}^{k} 4^j$, oder auch $(4^k-1)/3 < i \leq (4^{k+1}-1)/3$. Es gilt also $k = \lceil \log_4(3i+1) \rceil - 1 =: e(i)$. Die Position des Indexes i innerhalb einer Ebene ist somit $i - (4^{e(i)}-1)/3$.

Damit erhalten wir child$(i,j) = \overbrace{(4^{e(i)+1}-1)/3}^{\text{Anfang nächste Ebene}} + \overbrace{4(i-(4^{e(i)}-1)/3-1)}^{\text{4·Index von } i \text{ in jetzige Ebene}} + j = 4i-3+j$.
Für parent(i) bestimmen wir zunächst wieder die Position von i in der Ebene, teilen durch 4 und addieren dies auf die darunterliegende Ebene:
parent$(i) = \lfloor \frac{i-(4^{e(i)}-1)/3-1}{4} \rfloor + (4^{e(i)-1}-1)/3+1 = \lfloor \frac{i+2}{4} \rfloor$

Lösungsende

e. Implementieren Sie insert(e). Nutzen Sie Funktionen parent(i) und child(i,j) statt expliziter Indexberechnung. [2 Punkte]

Lösung

```
Procedure insert(e : Element)
    n++ ; h[n]:= e
    siftUp(n)

Procedure siftUp(i : ℕ)
    if i = 1 ∨ h[parent(i)] ≤ h[i] then return
    swap(h[i], h[parent(i)])
    siftUp(parent(i))
```

Lösungsende

Lösungsvorschlag

Aufgabe 5. Shuffle [8 Punkte]

Gegeben seien drei Wörter A, B, und C der Länge $|A| = n$, $|B| = m$ und $|C| = n+m$.
Das Wort C heißt *Shuffle* der Wörter A und B genau dann, wenn C durch eine Verzahnung der Buchstaben von A und B gebildet werden kann, wobei die Reihenfolge der Buchstaben der beiden Wörter erhalten bleiben muss.

a. Gegeben seien die Wörter $A =$*fußball* und $B =$*weltmeister*. Zeigen Sie oder widerlegen Sie, dass die folgenden beiden Wörter C_1 und C_2 jeweils ein Shuffle der Wörter A und B sind:
$C_1 =$*fuwelßtmebailstler*
$C_2 =$*fuwßeltbamelitsler* [2 Punkte]

Lösung

Damit C ein Shuffle der Wörter A und B sein kann, muss der letzte Buchstabe von C entweder der letzte Buchstabe des Wortes A oder der letzte Buchstabe des Wortes B sein und die verbleibenden Buchstaben von C müssen ein Shuffle aus den verbleibenden Buchstaben von A und B sein. Wir starten also jeweils vom Ende der Wörter C_1 und C_2 und überprüfen diese Bedingung.

Sei $X[i..j]$ das Teilwort von X, das den i-ten bis j-ten Buchstaben von X enthält.

Das Wort $C_1 =$*fuwelßtmebailstler* ist ein Shuffle der Wörter $A =$*fußball* und $B =$*weltmeister*:
$C_1 = A[1..2]B[1..3]A[3..3]B[4..6]A[4..5]B[7..7]A[6..6]B[8..9]A[7..7]B[10..11]$.

Das Wort $C_2 =$*fuwßeltbamelitsler* ist *kein* Shuffle der Wörter A und B, da die Zeichenfolge $C_2[14..15] = ts$ nicht erzeugt werden kann, ohne die Reihenfolge der Buchstaben in Wort B zu verändern.

Lösungsende

b. Entwerfen Sie einen Algorithmus, der für ein Wort C in Zeit $O(nm)$ bestimmt, ob es ein Shuffle der Wörter A und B ist. Wie in Teilaufgabe a) sei $|A| = n$, $|B| = m$ und $|C| = n+m$. [6 Punkte]

Lösung

Wir verwenden dynamische Programmierung und speichern die Zwischenergebnisse in einem $n \times m$ boolean Array S.
$S[i][j]$ enthalte genau dann *true*, wenn die ersten $i+j$ Buchstaben von C ein Shuffle der ersten i Buchstaben von A, sowie der ersten j Buchstaben von B bilden.
Sei $X[i..j]$ das Teilwort von $X \in \{A, B, C\}$, das den i-ten bis j-ten Buchstaben von X enthält.
Es gelten die folgenden beiden Randbedingungen, welche uns die Initialisierung des Arrays S liefern:

- Ist $|A| = 0$, so ist C genau dann ein Shuffle, wenn $C[1..m] = B[1..m]$.
- Ist $|B| = 0$, so ist C genau dann ein Shuffle, wenn $C[1..n] = A[1..n]$.

Andernfalls müssen 2 Fälle unterschieden werden, um festzustellen, ob $C[1..i+j]$ ein Shuffle von $A[1..i]$ und $B[1..j]$ ist:

- Der aktuelle Buchstabe $A[i]$ ist gleich $C[i+j]$ und das Teilwort $C[1..i+j-1]$ ist ein Shuffle von $A[1..i-1]$ und $B[1..j]$.
- Der aktuelle Buchstabe $B[j]$ ist gleich $C[i+j]$ und das Teilwort $C[1..i+j-1]$ ist ein Shuffle von $A[1..i]$ und $B[1..j-1]$.

Hieraus ergibt sich folgendes dynamisches Programm:

```
Function isShuffle(A,B,C : string) : bool
    n := |A|
    m := |B|
    S[n+1][m+1] : Array of Array of bool
    S[0][0] := true
    for i = 1,...,n do
        S[i][0] := C[i] = A[i] ∧ S[i-1][0]
    for j = 1,...,m do
        S[0][j] := C[j] = B[j] ∧ S[0][j-1]
    for i = 1,...,n do
        for j = 1,...,m do
            S[i][j] := (C[i+j] = A[i] ∧ S[i-1][j]) ∨ (C[i+j] = B[j] ∧ S[i][j-1])
    return S[n][m]
```

Lösungsende

Lösungsvorschlag

Aufgabe 6. Minimale Spannbäume [11 Punkte]

a. Nennen und erläutern Sie die Eigenschaft von minimalen Spannbäumen (MSTs) auf der der Algorithmus von Jarník-Prim basiert. [1 Punkt]

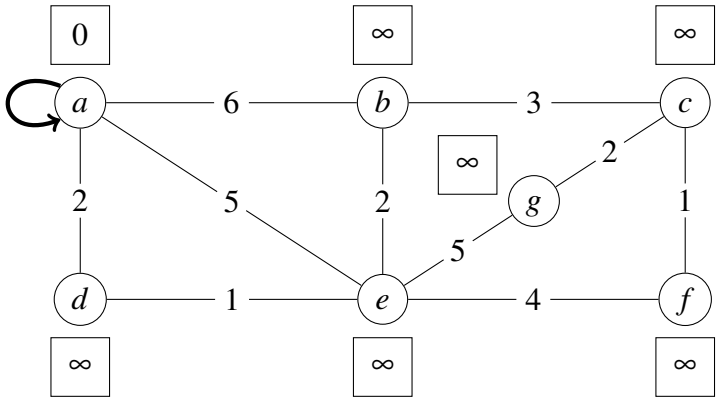
Lösung

Der Algorithmus von Jarník-Prim basiert auf der Schnitteigenschaft. Die leichteste Kante in einem Schnitt kann in einem MST verwendet werden.

Lösungsende

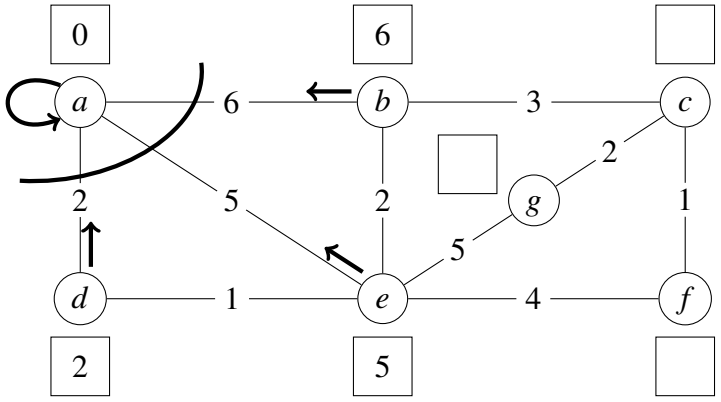
b. Führen Sie auf dem folgenden ungerichteten Graphen mit Kantengewichten den Algorithmus von Jarník-Prim mit Startknoten a durch. Verwenden Sie für jeden Schleifendurchlauf eine Kopie des Graphen, wobei die ersten zwei Iterationen bereits vorgegeben sind. Markieren Sie nach jeder Iteration den *aktuellen Schnitt* und tragen Sie für jeden Knoten v den Vorgänger $pred[v]$ als kurzen Pfeil und die aktuelle Distanz $d[v]$ in den Kasten ein, wobei Sie nur geänderte Werte eintragen müssen. Notieren Sie zwischen den Iterationen den Inhalt der Queue, und markieren sie nach Beenden den berechneten minimalen Spannbaum. Beschriften Sie deutlich die Graphen die gewertet werden sollen mit den Schleifendurchlaufsnummern! [5 Punkte]

0



$Q = \{$ *a* $\}$

1

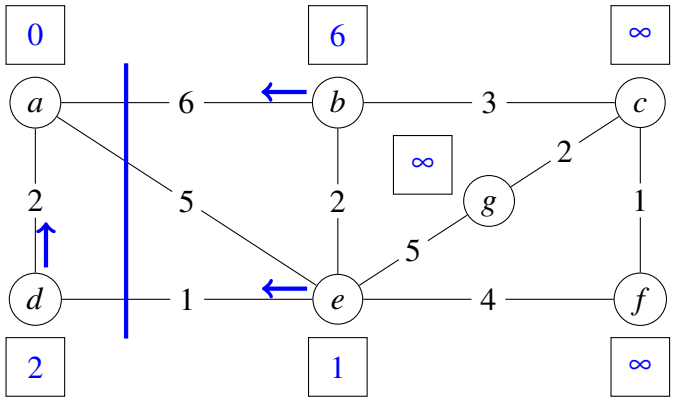


$Q = \{$ *b*, *d*, *e* $\}$

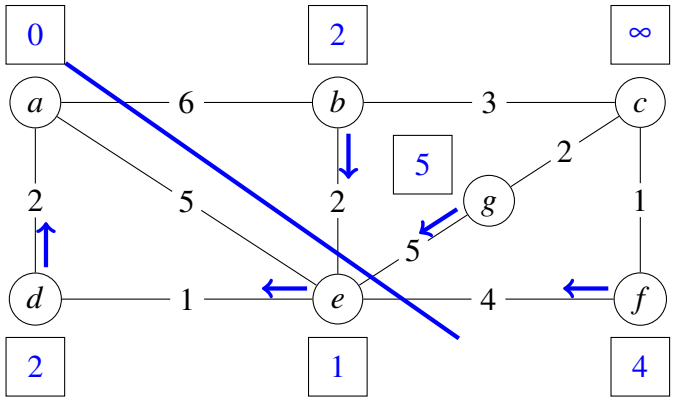
(weitere Graphen auf dem nächsten Blatt)

Lösungsvorschlag

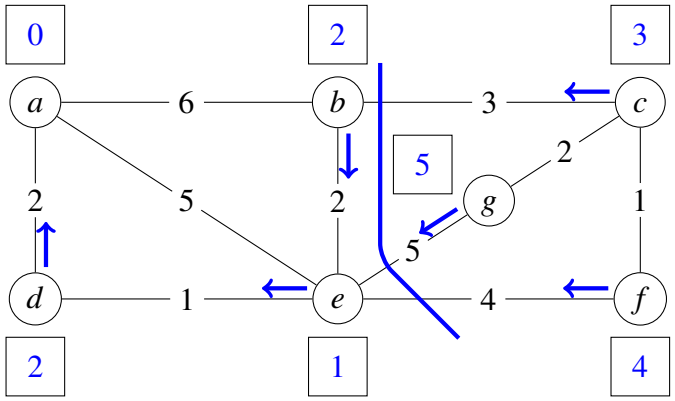
Fortsetzung von Aufgabe 6



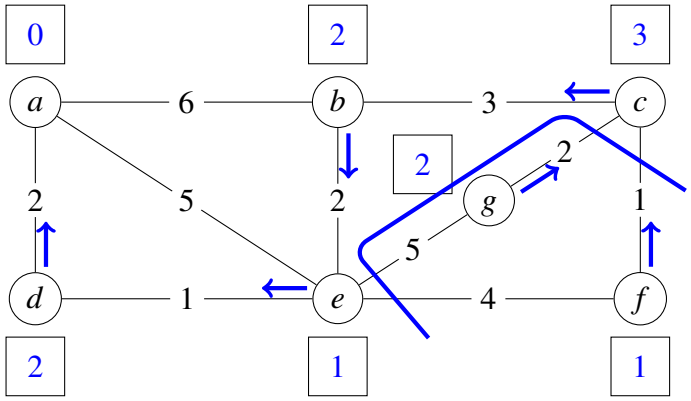
$Q = \{$ b, e $\}$



$Q = \{$ b, f, g $\}$



$Q = \{$ c, f, g $\}$

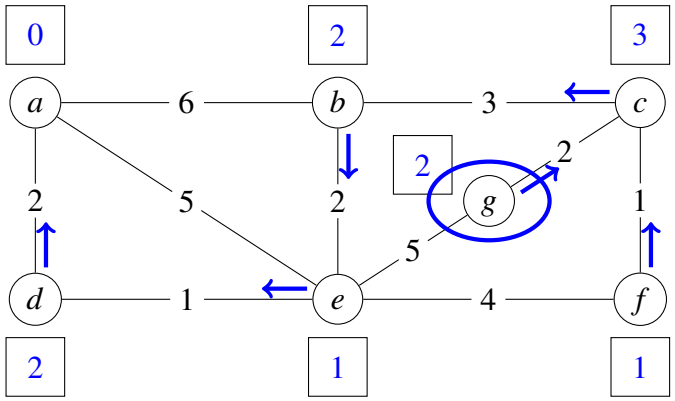


$Q = \{$ f, g $\}$

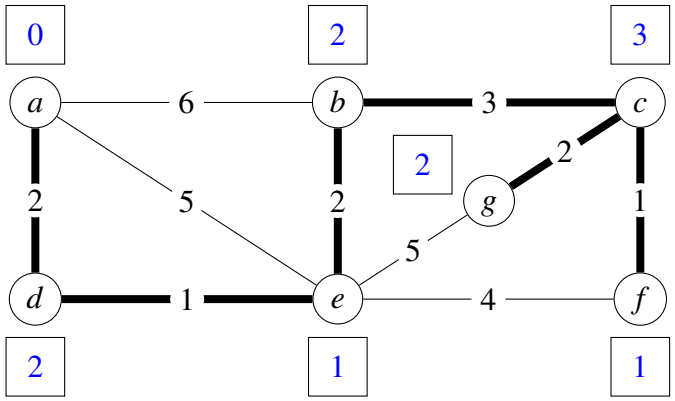
(weitere Graphen auf dem nächsten Blatt)

Lösungsvorschlag

Fortsetzung von Aufgabe 6



$Q = \{$ *g* $\}$



$Q = \{$ $\}$

(Teilaufgabe **c.** auf dem nächsten Blatt.
Sie erhalten weitere Blätter mit Graphen bei Bedarf von der Aufsicht.)

Lösungsvorschlag

Fortsetzung von Aufgabe 6

c. Professor Euler hat bereits viele Brücken untersucht und schlägt folgenden Algorithmus zur Berechnung eines minimalen Spannbaums vor:
In einem ungerichteten, gewichteten, zusammenhängenden Graphen untersucht man die Kanten nach Gewicht in nicht-steigender Reihenfolge, und entfernt jede Kante, die zu diesem Zeitpunkt keine Brücke ist. Eine Kante ist eine Brücke, wenn der Graph beim Entfernen der Kante in zwei Komponente verfällt.
Berechnet der Algorithmus von Professor Euler in jedem Graphen einen minimalen Spannbaum? Beweisen Sie dies, oder geben Sie ein Gegenbeispiel an. [5 Punkte]

Lösung

Der Algorithmus ist richtig und berechnet immer einen minimalen Spannbaum. Er verwendet das Kreiskriterium auf eine andere Art als der Algorithmus von Kruskal.
Zuerst zeigen wir, dass ein Baum entsteht: da keine Brücken entfernt werden, ist der verbleibende Graph zu jedem Zeitpunkt zusammenhängend. Nach Ausführen des Algorithmus ist der Graph kreislos, da der Algorithmus so lange Kanten löscht wie Kreise vorhanden sind weil alle Kanten eines Kreises keine Brücken sind. Damit ist der verbleibende Graph ein Baum.
Nun müssen wir zeigen, dass er kleinstes Gewicht hat, also ein MST ist. Hierzu verwenden wir Induktion und die Kreiseigenschaft:
Induktioninvariante: Es gibt einen MST des Graphen dessen Kanten alle Teilmenge der noch nicht gelöschten Kanten sind.
Induktionsanfang: Natürlich sind die Kanten jedes MST Teilmenge der Kanten des Graphen.
Induktionsschritt: Wird eine Kante e betrachtet, die keine Brücke ist, dann liegt diese auf einem Kreis. Auf diesem Kreis ist e auch garantiert eine schwerste Kante, da eine schwerere Kante bereits vorher betrachtet worden wäre und da diese auch keine Brücke ist, wäre sie entfernt worden. Daher ist e eine schwerste Kante eines Kreises, und laut Kreiseigenschaft gibt es einen MST für den diese Kante nicht benötigt wird. Daher kann sie entfernt werden. Damit sind auch nach der Löschung von e alle Kanten eines MST Teilmenge der noch nicht gelöschten Kanten.
Da am Ende des Algorithmus genau so viele Kanten übrig bleiben wie für einen MST benötigt werden, und alle Kanten eines MST in dieser Kantenmenge enthalten sind, bleiben genau die Kanten eines MST übrig.

Lösungsende