MUSTERLÖSUNG

Probeklausur

Datenstrukturen und Algorithmen

08.07.2013, Prof. Dr. L. Kobbelt

Nachname: _____

Vorname: _____

Matrikelnummer: _____

Studiengang: _____

- Schreiben Sie auf jedes Blatt **Vorname**, **Name** und **Matrikelnummer**.
- Geben Sie Ihre Antworten in lesbarer und verständlicher Form an.
- Bitte beantworten Sie die Aufgaben auf den **Aufgabenblättern** (benutzen Sie auch die Rückseiten). Antworten auf anderen Blättern können nur berücksichtigt werden, wenn **Name, Matrikelnummer und Aufgabennummer** deutlich darauf erkennbar sind.
- Was nicht bewertet werden soll, kennzeichnen Sie bitte durch **Durchstreichen**.
- Werden Täuschungsversuche beobachtet, so wird die Klausur mit **nicht bestanden** bewertet.
- Geben Sie am Ende der Bearbeitungszeit **alle Blätter zusammen mit den Aufgabenblättern ab**.
- Verwenden Sie ausschließlich dokumentenechte Stifte mit schwarzer oder blauer Farbe. Insbesondere sind keine Bleistifte und keine Stifte mit roter Farbe zugelassen.

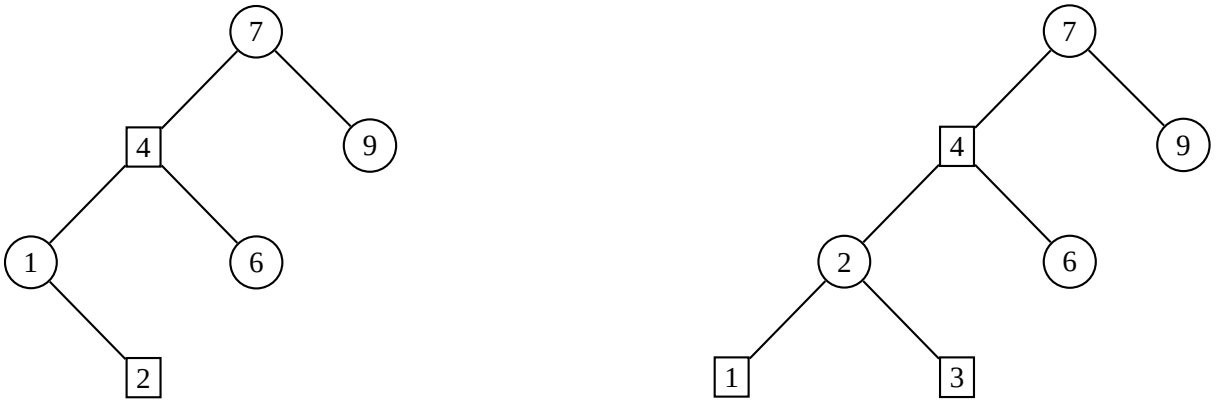
	Thema	Punkte	Erreichte Punkte
1	Bäume	14	
2	ADT	11	
3	Graphen	11	
4	Sortieren	8	
5	Multiple Choice	15	
6	Dynamisches Programmieren	8	
7	Hashing	5	
8	Laufzeitanalyse	8	
	Summe	80	

Vorname	Name	Matr.-Nr.	Seite
			1 / 27

Aufgabe 1: Bäume

1. (3 Punkte) Fügen Sie in die folgenden Rot-Schwarz-Bäume jeweils die angegebenen Schlüssel ein. Zeichnen Sie schwarze Knoten als Kreise und rote Knoten als Quadrate.

Lösung:
Insert(3):
Fall 3 (gespiegelt): Onkel ist schwarz, Knoten ist ein rechter Sohn ⇒ Rotation nach links + Umfärben.



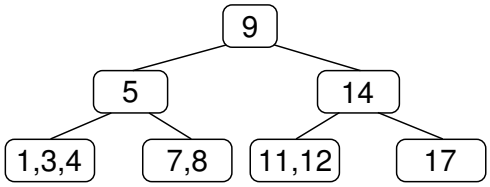
Insert(11):
1. Fall 2: Onkel ist schwarz, Knoten ist ein linker Sohn ⇒ Rotation nach rechts
2. Fall 3 (gespiegelt): Onkel ist schwarz, Knoten ist ein rechter Sohn ⇒ Rotation nach links + Umfärben.

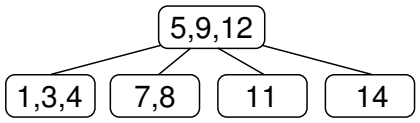
Bewertung:
1.5 Punkte pro vollständig richtigem Baum, keine Teilpunkte.

2. (4 Punkte) Löschen Sie aus den folgenden B-Bäumen der Ordnung $t = 2$ jeweils die angegebenen Schlüssel.

   Delete(17):
   Lösung:

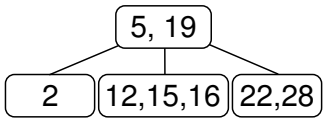
   1. Fall 3b: Merge 5, 9, 14
   2. Fall 3a: Rotate

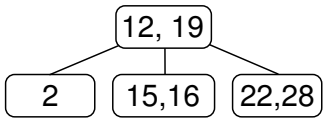




   Delete(5):
   Lösung:

   Fall 2b: Steal 12





   Bewertung:
   2 Punkte pro vollständig richtigem Baum, keine Teilpunkte.

3. (2 Punkte) Beschreiben Sie den “One-Pass-Algorithmus” zum Einfügen von Knoten in einen B-Baum der Ordnung t. Welches Problem des naiven Einfüge-Algorithmus verhindert der One-Pass-Algorithmus?

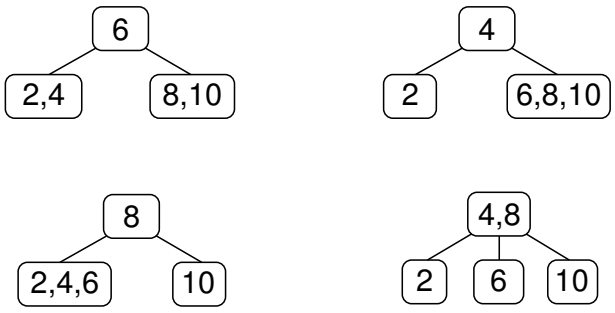
   Lösung:
   Der One-Pass-Algorithmus steigt in einem B-Baum der Ordnung t bis zu dem Knoten ab, in den ein neuer Schlüssel eingefügt werden muss. Auf dem Weg dorthin überprüft der Algorithmus den Füllgrad jedes Knotens. Alle vollen Knoten mit $2t - 1$ Schlüsseln werden während des Abstiegs gesplittet.

   Ohne den One-Pass-Algorithmus müssen Splits nach oben propagiert werden. Problem: auf die entsprechenden Knoten muss zweimal zugegriffen werden.

   Bewertung:
   1 Punkt für One-Pass-Algorithmus,
   1 Punkt für Erklärung des Problems ohne One-Pass.

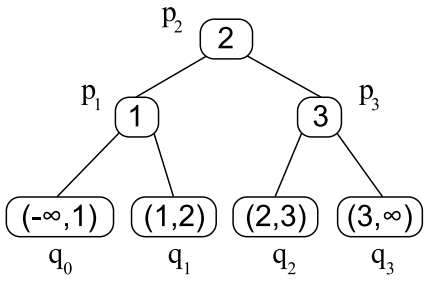
4. (2 Punkte) Geben Sie alle korrekten B-Bäume der Ordnung $t = 2$ an, die die Schlüssel $2, 4, 6, 8, 10$ enthalten. Hierbei sollen auch Bäume berücksichtigt werden, die durch das Einfügen und spätere Löschen zusätzlicher Elemente entstehen können.

   Lösung:



   Bewertung:
   0.5 Punkte pro korrektem Baum,
   -0.5 Punkte pro falschem Baum um das Erraten von Bäumen zu verhindern.

5. Wir betrachten den folgenden Suchbaum:



Gegeben seien die Wahrscheinlichkeiten der inneren Knoten $p_1 = 0.25$, $p_2 = 0.20$, $p_3 = 0.15$ und der Blätter $q_0 = 0.25$ sowie $q_1 = q_2 = q_3 = 0.05$.

(a) (1 Punkt) Berechnen Sie die erwarteten Kosten für den oben angegebenen Suchbaum.

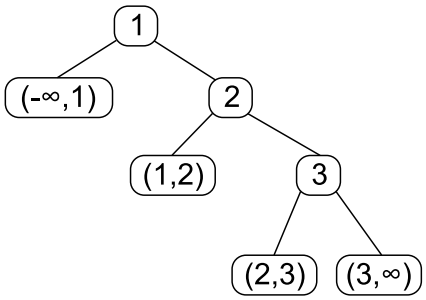
Lösung:
Die erwarteten Kosten für den gegebenen Suchbaum sind:

$$2 * 0.25 + 1 * 0.20 + 2 * 0.15 + 3 * 0.25 + 3 * 0.05 + 3 * 0.05 + 3 * 0.05 = 2.20$$

Bewertung:
1 Punkt für korrekte Kosten.

(b) (2 Punkt) Geben Sie den optimalen Suchbaum für die angegebenen Wahrscheinlichkeiten an. Wieviel Kosteneinsparung kann erwartet werden?
(Hinweis: Zusammen mit dem Baum aus Teil (a) existieren 5 mögliche Bäume. Gesucht ist der Baum mit den optimalen Kosten.)

Lösung:



Die zu erwartenden Kosten sind:

$$1 * 0.25 + 2 * 0.20 + 3 * 0.15 + 2 * 0.25 + 3 * 0.05 + 4 * 0.05 + 4 * 0.05 = 2.15$$

Die erwartete Einsparung beträgt also 0.05.

Bewertung:
1 Punkt für den optimalen Baum,
1 Punkt für die korrekte Kostenersparnis.

Aufgabe 2: Abstrakte Datentypen

In dieser Aufgabe soll ein abstrakter Datentyp PLISTE spezifiziert werden. Bei diesem Datentyp handelt es sich um eine gepackte Liste, d.h. eine Liste, deren Elemente Paare von Werten und Anzahlen sind. So wird z.B. die normale Liste mit den vier Buchstaben $[A, B, B, A]$ durch die gepackte Liste $[(A, 1), (B, 2), (A, 1)]$ dargestellt. Eine solche Kodierung nennt man "Run-Length Encoding" (kurz: RLE).

1. (2 Punkte) Zuerst benötigen wir einen Hilfstypen LISTE für normale (ungepackte) Listen von Werten.

   Definieren Sie den ADT LISTE für eine LIFO-Liste mit den folgenden Operationen durch ihre Signatur und geeignete Axiome:

$Create$	Erzeugt eine leere Liste.
$Push$	Fügt ein Element am Anfang der Liste ein.
Top	Liefert das erste Element der Liste zurück.
Pop	Entfernt das erste Element der Liste.
$IsEmpty$	Liefert $True$, falls die Liste leer ist.

   Sie können davon ausgehen, dass der Elementtyp WERT bereits definiert ist. Ebenso dürfen Sie davon ausgehen, dass der Typ BOOLEAN für Wahrheitswerte mit $True, False: \rightarrow$ BOOLEAN gegeben ist.

   Lösung:

$$
\begin{array}{rrcl}
Create: & & \rightarrow & \text{LISTE} \\
Push: & \text{WERT} \times \text{LISTE} & \rightarrow & \text{LISTE} \\
Top: & \text{LISTE} & \rightarrow & \text{WERT} \\
Pop: & \text{LISTE} & \rightarrow & \text{LISTE} \\
IsEmpty: & \text{LISTE} & \rightarrow & \text{BOOLEAN}
\end{array}
$$

$$
\begin{aligned}
Top(Push(t, l)) &= t \\
Pop(Push(t, l)) &= l \\
IsEmpty(Create()) &= True() \\
IsEmpty(Push(t, l)) &= False()
\end{aligned}
$$

   Bewertung:
   je 0.25 Punkte für Signaturen außer $Create$,
   je 0.25 Punkte für Axiome.

2. (2 Punkte) Nun definieren wir den ADT PLISTE für gepackte LIFO-Listen.

   Die folgenden Operationen sollen auf einer PLISTE durchführbar sein:

$PCreate()$	Erzeugt eine leere gepackte Liste.
$PPush()$	Fügt einen Wert und die zugehörige Anzahl an den Anfang der gepackten Liste ein.
$Value$	Liefert den im ersten Element der Liste gespeicherten Wert zurück.
$Count$	Liefert die im ersten Element der Liste gespeicherte Anzahl zurück.
$PPop$	Entfernt das erste Element (also das erste Paar) der gepackten Liste.
$PIsEmpty$	Liefert $True$, falls die gepackte Liste leer ist.

   Sie können davon ausgehen, dass der Elementtyp WERT bereits definiert ist. Außerdem dürfen Sie wieder BOOLEAN benutzen und können davon ausgehen, dass der Typ NAT für natürliche Zahlen wie folgt gegeben ist:

$$\begin{array}{rrcl} Zero: & & \to & \text{NAT} \\ Succ, Pred: & \text{NAT} & \to & \text{NAT} \\ IsZero: & \text{NAT} & \to & \text{BOOLEAN} \end{array}$$

$$\begin{array}{rcl} Pred(Succ(n)) & = & n \\ IsZero(Zero()) & = & True() \\ IsZero(Succ(n)) & = & False() \end{array}$$

   Geben Sie die Signatur von PLISTE sowie geeignete Axiome an.

Lösung:

$$\begin{array}{rrcl} PCreate: & & \to & \text{PLISTE} \\ PPush: & \text{WERT} \times \text{NAT} \times \text{PLISTE} & \to & \text{PLISTE} \\ Value: & \text{PLISTE} & \to & \text{WERT} \\ Count: & \text{PLISTE} & \to & \text{NAT} \\ PPop: & \text{PLISTE} & \to & \text{PLISTE} \\ PIsEmpty: & \text{PLISTE} & \to & \text{BOOLEAN} \end{array}$$

$$\begin{array}{rcl} Value(PPush(w,n,l)) & = & w \\ Count(PPush(w,n,l)) & = & n \\ PPop(PPush(w,n,l)) & = & l \\ PIsEmpty(PCreate()) & = & True() \\ PIsEmpty(PPush(w,n,l)) & = & False() \end{array}$$

Bewertung:
je 0.25 Punkte pro Signatur außer $PCreate$ und $PIsEmpty$,
je 0.25 Punkte pro Axiom (wobei die beiden von $PIsEmpty$ zusammen zählen)

3. Es soll eine Operation $UnRLE$ definiert werden, die eine gepackte Liste in eine normale (ungepackte) Liste überführt. Dabei soll der Wert jedes Elements entsprechend der angegebenen Anzahl vervielfältigt werden. So soll z.B.

$$UnRLE(PPush(A, 1, PPush(B, 2, PPush(A, 1, PCreate()))))$$

zu

$$Push(A, Push(B, Push(B, Push(A, Create()))))$$

ausgewertet werden können.

(a) (2 Punkte) Geben Sie Signatur und Axiome für $UnRLE$ sowie evtl. benötige Hilfsoperationen an. Die Datentypen WERT, LISTE, BOOLEAN und NAT sowie ihre Axiome dürfen vorausgesetzt werden.

Lösung:

$$UnRLE: \quad \text{PLISTE} \quad \rightarrow \quad \text{LISTE}$$

$$\begin{aligned} UnRLE(PCreate()) &= Create() \\ UnRLE(PPush(w, Zero(), l) &= UnRLE(l) \\ UnRLE(PPush(w, Succ(n), l) &= Push(w, UnRLE(PPush(w, n, l))) \end{aligned}$$

Bewertung:
je 0.25 Punkte für Signatur und 1. Axiom,
0.5 Punkte für 2. Axiom,
1 Punkt für 3. Axiom.

(b) (2 Punkte) Geben Sie Pseudocode/“Stripped Java” für die Operation $UnRLE$ an. Benutzen Sie hierfür höchstens die Operationen $Create$, $Push$, $PPush$, $PPop$, $Value$, $Count$, $PIsEmpty$, $IsZero$ und $Pred$.

Lösung:

```
Liste UnRLE(PListe l) {
  if (PIsEmpty(l)) {
    return Create();
  } else {
    Nat n = Count(l);
    PListe rest = PPop(l);
    if (IsZero(n)) {
      return UnRLE(rest);
    } else {
      Wert w = Value(l);
      return Push(w,UnRLE(PPush(w,Pred(n),rest)));
    }
  }
}
```

Bewertung:
2 Punkte für korrekte Implemetierung.

4. (3 Punkte) Es soll eine Operation RLE definiert werden, die eine normale (ungepackte) Liste bekommt und eine gepackte Liste liefert, so dass die Originalliste durch entsprechendes Vervielfältigen wieder hergestellt werden kann. So soll z.B.

$$RLE(Push(A, Push(B, Push(B, Push(A, Create())))))$$

zu

$$PPush(A, 1, PPush(B, 2, PPush(A, 1, PCreate())))$$

ausgewertet werden können.

Die Signaturen von RLE und einer benötigten Hilfsfunktion $RLE2$ sind

$$\begin{array}{rrcl} RLE: & \text{LISTE} & \rightarrow & \text{PLISTE} \\ RLE2: & \text{LISTE} \times \text{WERT} \times \text{NAT} & \rightarrow & \text{PLISTE} \end{array}$$

Geben Sie Axiome für RLE und $RLE2$ an. Die Datentypen WERT, LISTE, BOOLEAN und NAT sowie ihre Axiome dürfen vorausgesetzt werden.

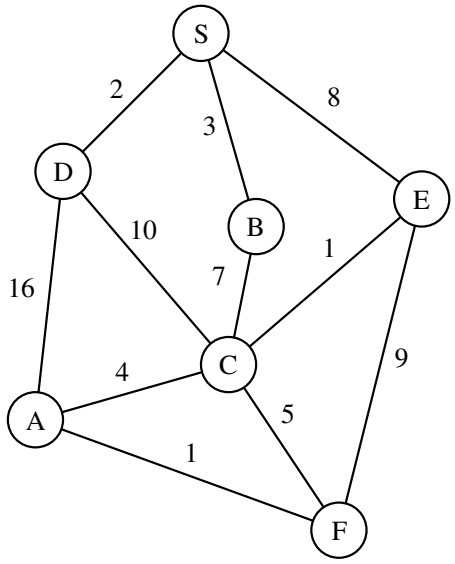
Lösung:

$$\begin{array}{rcl} RLE(Create()) & = & PCreate() \\ RLE(Push(w,l)) & = & RLE2(l, w, Succ(Zero())) \\ RLE2(Create(), w, n) & = & PPush(w, n, PCreate()) \\ RLE2(Push(w,l), w, n) & = & RLE2(l, w, Succ(n)) \\ RLE2(Push(w,l), v, n) & \overset{w \neq v}{=} & PPush(v, n, RLE2(l, w, Succ(Zero()))) \end{array}$$

Bewertung:
1 Punkt für Basisfall leere Liste (Axiome 1 und 2),
1 Punkt für Axiome 3 und 4,
1 Punkt für Axiom 5.

Aufgabe 3: Graphen

1. (2 Punkte) Wenden Sie Dijkstras Algorithmus auf den folgenden Graphen G mit Startknoten S an. Geben Sie die Werte aller oberen Schranken nach jedem Durchlauf der Hauptschleife von Dijkstras Algorithmus in Form einer Tabelle an. Die Tabelle enthält also eine Spalte für jeden Knoten $v \in G$. Pro Iteration soll eine Zeile hinzugefügt werden, die die aktuellen oberen Schranken $\delta(S, v)$ enthält. Markieren Sie dabei in jeder Zeile den Knoten, den Sie auswählen.

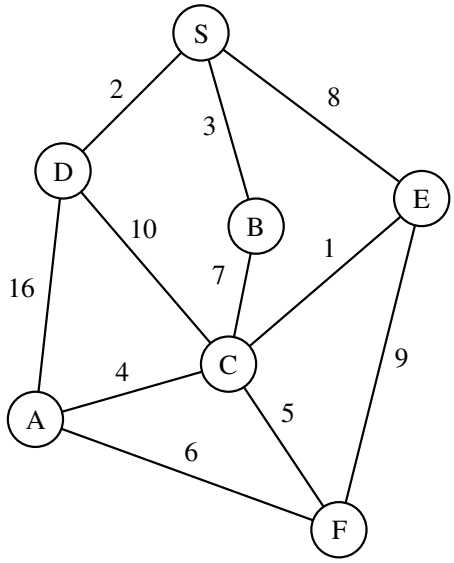


Lösung:

	S	A	B	C	D	E	F	Knoten
0	0	∞	3	∞	2	8	∞	S
1	0	18	3	12	2	8	∞	D
2	0	18	3	10	2	8	∞	B
3	0	18	3	9	2	8	17	E
4	0	13	3	9	2	8	14	C
5	0	13	3	9	2	8	14	A
6	0	13	3	9	2	8	14	F

Bewertung:
2 Punkte bei vollständig korrekter Tabelle,
1 Punkt bei kleinen Rechenfehlern aber richtiger Knoten-Reihenfolge,
0 Punkte sonst.

2. (1 Punkt) Wenden Sie den Algorithmus von Prim an, um einen minimal spannenden Baum des folgenden Graphen zu berechnen. Geben Sie dazu die Kanten in der Reihenfolge an, in der sie zur Lösungsmenge hinzugefügt werden. Da alle Kantengewichte verschieden sind, genügt es, jede Kante durch ihr Gewicht zu identifizieren.



Lösung:
$2, 3, 7, 1, 4, 5$

Bewertung:
1 Punkt für vollständig richtige Kantenfolge. Keine Teilpunkte.

3. Union-Find Strukturen

(a) (1 Punkt) Führen Sie die folgenden Operationen mit Höhenbalancierung auf einer anfangs leeren Union-Find Struktur aus. Beachten Sie: Wenn die Höhe der Bäume gleich ist, wird der Repräsentant mit dem kleineren Wert der neue Repräsentant. Zeichnen Sie den entstandenen Wald nach den Operationen

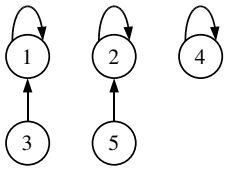
- ```
  for i = 1 to 5 do
      MakeSet(i)
  ```

Lösung:

Bewertung:
0.25 Punkte für alle MakeSet Operationen

- ```
  Union(1, 3)
  Union(2, 5)
  ```

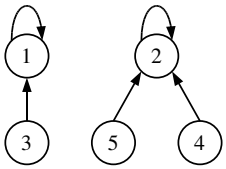
Lösung:



Bewertung:
0.25 Punkte für korrekte Operationen.

- ```
  Union(4, 5)
  ```

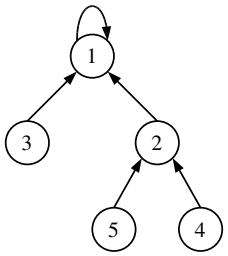
Lösung:



Bewertung:
0.25 Punkte für korrekte Operation.

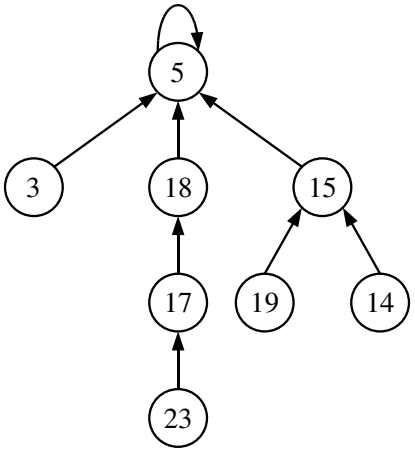
- ```
  Union(5, 3)
  ```

Lösung:



Bewertung:
0.25 Punkte für korrekte Operation.

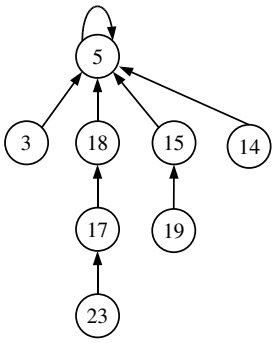
(b) (1 Punkt) Führen Sie auf der Union-Find Struktur



nacheinander die folgenden Operationen mit Pfadkompression aus. Zeichnen Sie nach jeder Operation den entstandenen Baum.

- `Find(14)`

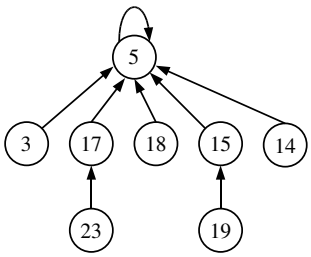
  Lösung:

  Bewertung:
  0.5 Punkte für korrekten Baum.

- `Find(17)`

  Lösung:

  Bewertung:
  0.5 Punkte für korrekten Baum.

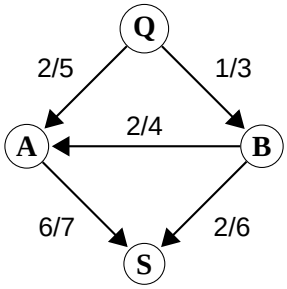
(c) (2 Punkte) Geben Sie den Kruskal-Algorithmus in Pseudo-Code/“Stripped Java” mit Hilfe von Union-Find Strukturen an. Verwenden Sie dazu die Funktionen `Emptyset`, `MakeSet`, `Find` und `Union`.

Lösung:

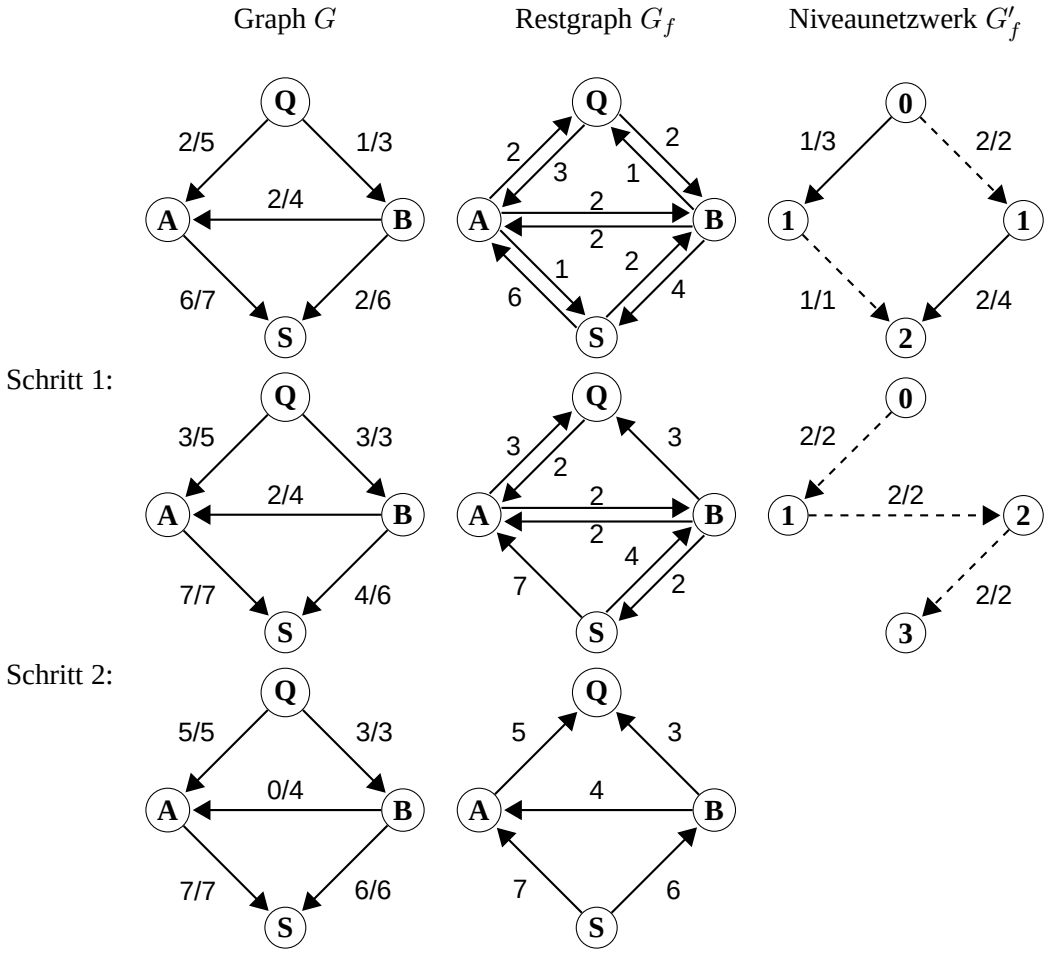
```
Kruskal(G,w)
    A = Emptyset
    for all v in V do
        MakeSet(v)
    sort E into non-decreasing order
    for each (u,v) in E do
        if Find(u) != Find(v) then
            A = A + {(u,v)}
            Union(u,v)
    return A
```

Bewertung:
0.5 Punkte für den `MakeSet`-Block,
0.5 Punkte für die Sortierung der Kanten,
1 Punkt für korrekte Schleife über die Kanten.

4. (4 Punkte) Wenden Sie den Algorithmus von *Dinic* auf den folgenden Graphen G an. Geben Sie zu jeder Iteration jeweils den Fluss in G, den Restgraph und das Niveaunetzwerk an.



Lösung:



Bewertung:
Je 0.75 Punkte für G_f und G'_f in der ersten Zeile.
Je 0.5 Punkte für jeden weiteren Graphen.

Aufgabe 4: Sortieren

1. (2 Punkte) Sortieren Sie die folgende Zahlenfolge aufsteigend mit Hilfe des Algorithmus Merge-Sort aus der Vorlesung. Verwenden Sie hierbei die iterative Variante. Geben Sie den Inhalt des Arrays nach dem Zusammenführen von allen Teilfolgen der gleichen Länge an.

   Lösung:

13	56	77	42	14	21	83	64	22	98	59	11	85	51	19	73
13	56	42	77	14	21	64	83	22	98	11	59	51	85	19	73
13	42	56	77	14	21	64	83	11	22	59	98	19	51	73	85
13	14	21	42	56	64	77	83	11	19	22	51	59	73	85	98
11	13	14	19	21	22	42	51	56	59	64	73	77	83	85	98

   Bewertung:
   2 Punkte für korrekte Tabelle.

2. (3 Punkte) Für die vergleichsbasierte Sortierung von n Werten wurde in der Vorlesung gezeigt, dass man mindestens Zeitaufwand $\Omega(n \log n)$ benötigt. Wieso kann ein Algorithmus wie Radix-Sort dann in $O(n)$ laufen?

   Lösung:

   Betrachten wir Radix-Sort auf binären Wörtern. Alle n Werte stammen aus einem beschränkten und diskreten Wertebereich $\{0, 1, \ldots, 2^k\}$. Um die n Werte zu sortieren, sortieren wir für jede Stelle (also k-mal) eine Liste von 0-en und 1-en. Dies funktioniert offensichtlich in Linearzeit (also $O(n)$). Damit erhalten wir für ein von n unabhängiges k direkt $O(k \cdot n) = O(n)$.

   Bewertung:
   1.5 Punkte für Idee, dass Radix-Sort auf beschränktem, diskretem Wertebereich arbeitet,
   1.5 Punkte für Begründung der linearen Laufzeit.

3. (3 Punkte) Gegeben seien n Punkte $(x_i, y_i) \in \mathbb{R}^2$ im Quadrat mit den Ecken $(1, 0)$, $(0, 1)$, $(-1, 0)$ und $(0, -1)$, d.h., $|x_i| + |y_i| \leq 1$. Diese Punkte seien gleichverteilt, d.h. die Wahrscheinlichkeit, dass ein Punkt in einem Gebiet G innerhalb des Quadrates zu liegen kommt ist proportional zum Flächeninhalt von G.

   Beschreiben Sie eine Variante des Bucketsort-Algorithmus, der die Punkte nach ihrer Betragssumme $|X| + |Y|$ sortiert und dessen Laufzeit im erwarteten Fall $O(n)$, d.h. linear ist. Geben Sie eine Definition für die verwendeten Buckets sowie eine Begründung für die lineare Laufzeit an.

   Lösung:
   Der Flächeninhalt des Quadrates ist 2. Wir benötigen n Buckets, von denen jeder eine Fläche von $\frac{2}{n}$ umschließt und den Wertebereich der Betragssumme in n Teile zerteilt.

   Nimmt man die ersten i Teilquadrate zusammen, so müssen diese einen Flächeninhalt von $A_i = i \cdot \frac{2}{n}$ besitzen. Bezeichnet r_i die halbe Diagonale des i-ten Teilquadrats, so gilt für den Flächeninhalt ebenso $A_i = 2 \cdot r_i^2$.

   Durch Gleichsetzen erhalten wir $i \cdot \frac{2}{n} = 2 \cdot r_i^2$ und damit, unter Beachtung der Randbedingung, dass r_i positiv ist, $r_i = \sqrt{\frac{i}{n}}$.

   Durch Anwendung von Bucket-Sort erhalten wir auf Grund der Gleichverteilung einen Erwartungswert von $O(n)$ für die Laufzeit unseres Algorithmus.

   Bewertung:
   1 Punkt für die Begründung der linearen Laufzeit,
   2 Punkte für die korrekten Buckets.

Aufgabe 5: Multiple Choice

Bei einigen der folgenden Fragen sind mehrere der vorgegebenen Antworten richtig. Kreuzen Sie alle richtigen Antworten an. Pro Frage erhalten Sie für eine vollständig richtige Antwort 1 Punkt. Beantworten Sie ein Frage nicht, so erhalten Sie 0 Punkte. Beantworten Sie eine Frage unvollständig oder falsch, so erhalten Sie -0.5 Punkte. Insgesamt können Sie jedoch in dieser Aufgabe keine negative Punktzahl erhalten.

1. Gegeben sei folgende Rekursionsgleichung

$$T(1) = 7$$
$$T(n) = 7 \cdot T(\frac{n}{7}) + 7 \cdot n \quad \text{für } n > 1$$

Wie läßt sich T dann klassifizieren?

⊗ $T(n) \in \Omega(1)$
⊗ $T(n) \in O(n \log n)$
⊗ $T(n) \in O(n^3)$
○ Keine der obigen Klassifizierungen trifft zu.

2. Gegeben sei folgende Rekursionsgleichung

$$T(1) = 42$$
$$T(n) = 4 \cdot T(\frac{n}{2}) + 8 \cdot n^2 + 16 \cdot n \quad \text{für } n > 1$$

Wie läßt sich T dann klassifizieren?

○ $T(n) \in O(n \log n)$
⊗ $T(n) \in O(n^2 \log n)$
⊗ $T(n) \in \Omega(n^2)$
○ Keine der obigen Klassifizierungen trifft zu.

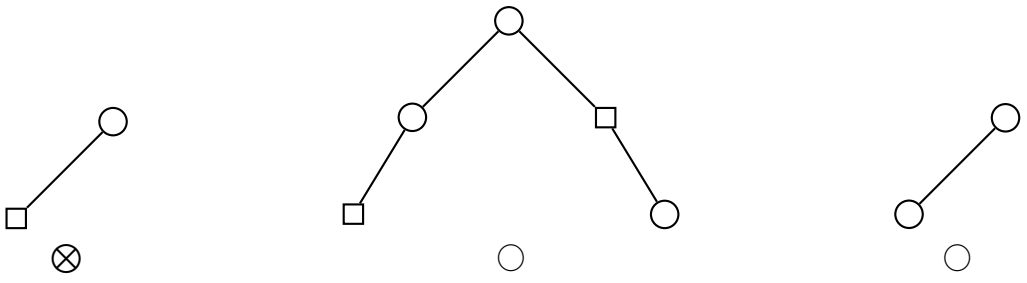
3. In zusammenhängenden Graphen $G = (V, E)$ gilt $|E| = \Omega(|V|)$.

⊗ Richtig ○ Falsch

4. Zwei verschiedene spannende Bäume eines Graphen enthalten mindestens eine gemeinsame Kante.

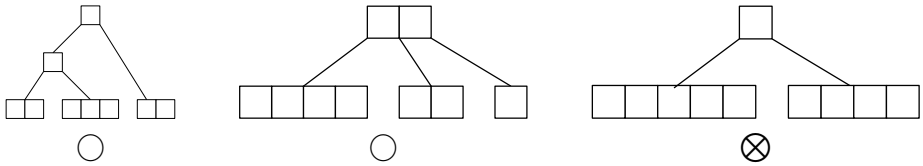
○ Richtig ⊗ Falsch

5. Welche der folgenden Bäume sind Rot-Schwarz-Bäume? Kreise stellen schwarze Knoten dar, Quadrate stellen rote Knoten dar.



⊗ ○ ○

6. Welche der folgenden Bäume können B-Bäume sein?

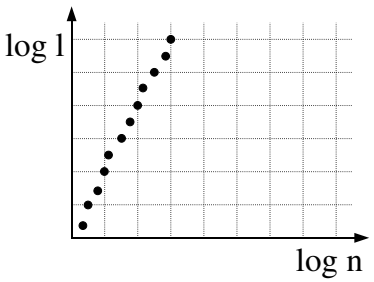


○ ○ ⊗

7. Es gibt einen planaren, ungerichteten Graphen mit 5 Knoten und 10 (verschiedenen) Kanten.

○ Richtig ⊗ Falsch

8. Die Laufzeiten l_i eines Algorithmus A wurden für eine Menge von Datensätzen der Größen n_i gemessen. Es ergab sich folgender Zusammenhang, wobei sowohl die n- als auch die l-Achse eine logarithmische Skala haben.



Welche asymptotische Komplexität hat A vermutlich?

○ $\Theta(n)$ ○ $\Theta(n \log n)$ ⊗ $\Theta(n^2)$

9. Bei geschlossenem Hashing beträgt der erwartete Aufwand zum Einfügen eines Elements in eine Hashtabelle mit Belegungsfaktor α

○ $O(\alpha^2)$ ○ $O(1+\alpha)$ ⊗ $O(\frac{1}{1-\alpha})$

10. Die minimale Zahl an Kopieroperationen um eine Folge der Länge n mit Selection-Sort zu sortieren ist

⊗ $\Theta(n)$ ○ $\Theta(n \log n)$ ○ $\Theta(n^2)$

11. Jeder ungerichtete Graph $G = (V, E)$ mit mehr als $|V|$ Kanten ist zusammenhängend.

◯ Richtig ⊗ Falsch

12. Das *Versickern* eines Elements bei Heap-Sort benötigt $\Omega(n \log n)$ Zeitaufwand.

◯ Richtig ⊗ Falsch

13. Dijkstras Algorithmus hat auf einem Baum mit n Knoten einen Aufwand von

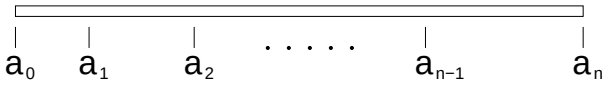
⊗ $\Theta(n)$ ◯ $\Theta(n \log n)$ ◯ $\Theta(n^2)$

14. Die Komplexität des Algorithmus von Dinic auf einem Graphen $G = (V, E)$ ist

⊗ $O(|V|^2 \cdot |E|)$ ◯ $O(|V|^3)$ ⊗ $O(|V|^4)$

Aufgabe 6: Dynamisches Programmieren

Gegeben sei ein Stab der Länge $l \in \mathbb{N}_{>0}$. Dieser Stab soll an $n-1$ Stellen $a_i \in \mathbb{N}, 0 \leq a_i \leq l$, $i = 1, \ldots, n-1$ in n Segmente zerteilt werden. Der Einfachheit halber setzen wir außerdem $a_0 = 0$ und $a_n = l$ und es gelte $a_i < a_{i+1}\ \forall i$.



Die Kosten für einen Schnitt durch ein Segment der Länge m betragen m. Die Kosten, um den Stab an den vorgegebenen Stellen zu zerschneiden, hängen von der Reihenfolge der Schnitte ab. Beispiel: Ein Stab der Länge 10 soll an den Stellen $a_1 = 5$, $a_2 = 8$ und $a_3 = 9$ zerschnitten werden. Schneidet man in der Reihenfolge a_1, a_2, a_3 so betragen die Kosten $10 + 5 + 2 = 17$. Schneidet man in der Reihenfolge a_3, a_2, a_1 so betragen die Kosten hingegen $10 + 9 + 8 = 27$.
In dieser Aufgabe soll ein auf Dynamischem Programmieren basierender Algorithmus hergeleitet werden, der die minimalen Kosten berechnet, die anfallen, wenn der Stab zerschnitten werden soll. Hierzu bezeichne $c(i, j), 0 \leq i < j \leq n$ die minimalen Kosten um alle Schnitte zwischen a_i und a_j durchzuführen. Gesucht ist also $c(0, n)$.

1. Geben Sie eine Rekursionsformel für $c(i, j)$ an.

   Basisfall: (1 Punkt) Für alle i mit $0 \leq i < n$ ist

$$c(i, i+1) = 0$$

   Rekursion: (3 Punkte) Für alle i, j mit $0 \leq i$ und $i + 1 < j$ und $j \leq n$ ist

$$c(i, j) = a_j - a_i + \min_{k=i+1,\ldots,j-1} c(i, k) + c(k, j)$$

   Bewertung:
   1 Punkt für Basisfall,
   3 Punkte für Rekursion

2. (2 Punkte) Geben Sie in Pseudocode/“Stripped Java” einen auf Dynamischem Programmieren basierenden Algorithmus an, der $c(0, n)$ berechnet.

Lösung:

```
for i ← 0 to n − 1 do
    c[i, i + 1] ← 0
for h ← 2 to n do
    for i ← 0 to n − h do
        j ← i + h
        min ← c[i, i + 1] + c[i + 1, j]
        for k ← i + 2 to j − 1 do
            if c[i, k] + c[k, j] < min then
                min ← c[i, k] + c[k, j]
        c[i, j] ← a[j] − a[i] + min
return c[0, n]
```

Bewertung:
1 Punkt für die Reihenfolge der Schleifen, äußere Schleife über Anzahl der Schnitte,
1 Punkt für korrekten Inhalt der äußeren beiden Schleifen.

3. (2 Punkte) Geben Sie eine kleinste obere Schranke für die asymptotische Laufzeit Ihres Algorithmus an (mit kurzer Begründung).

Lösung:
Die Tabelle enthält $n \times n$ Einträge. Jeder Eintrag kann in $O(n)$ Schritten berechnet werden. Der Gesamtaufwand ist daher $O(n^3)$.

Bewertung:
1 Punkt für korrekte Antwort,
1 Punkt für Begründung.

Aufgabe 7: Hashing

1. Erstellen Sie durch sukzessives Einfügen von Datensätzen mit den Schlüsselwerten

$$7, 19, 24, 63, 50, 73, 37, 54, 43, 49, 44$$

eine Hashtabelle der Länge $m = 11$. Verwenden Sie dazu die Hashfunktion

$$h(x) = x \mod 11$$

und

(a) (1 Punkt) geschlossenes Hashing mit linearem Sondieren

Lösung:

0	1	2	3	4	5	6	7	8	9	10
54	43	24	44	37	49	50	7	19	63	73

7 : 7
19: 8
24: 2
63: $8 \to 9$
50: 6
73: $7 \to 8 \to 9 \to 10$

37: 4
54: $10 \to 0$
43: $10 \to 0 \to 1$
49: 5
44: $0 \to 1 \to 2 \to 3$

Bewertung:
1 Punkt für eine korrekt ausgefüllte Tabelle, 0 Punkte sonst

(b) (1 Punkt) geschlossenes Hashing mit quadratischem Sondieren. Verwenden Sie die Variante aus der Vorlesung, bei der die Quadrate addiert werden.

Lösung:

0	1	2	3	4	5	6	7	8	9	10
73	44	24	43	37	49	50	7	19	63	54

7 : 7
19: 8
24: 2
63: $8 \to 9$
50: 6
73: $7 \to 8 \to 0$

37: 4
54: 10
43: $10 \to 0 \to 3$
49: 5
44: $0 \to 1$

Bewertung:
1 Punkt für eine korrekt ausgefüllte Tabelle, 0 Punkte sonst

2. Betrachten Sie nun offenes Hashing, bei dem Kollisionen durch Verkettung von Elementen behandelt werden.

   (a) (1 Punkt) Geben Sie die worst-case Laufzeiten (in O-Notation) für die erfolgreiche Suche, die erfolglose Suche, das Einfügen und Löschen in einer offenen Hashtabelle mit regulären (unsortierten) Listen an.

   Lösung:

   - Erfolgreiche Suche: $O(n)$, gesamte Liste muss durchlaufen werden
   - Erfolglose Suche: $O(n)$, gesamte Liste muss durchlaufen werden
   - Einfügen: $O(1)$, Einfügen am Anfang der Liste
   - Löschen: $O(n)$, identisch zu Suche

   Bewertung:
   0.25 Punkte pro korrekter Laufzeit.

   (b) (2 Punkte) Die Listen seien nun sortiert. Geben Sie wieder die worst-case Laufzeiten für die Operationen erfolgreiche Suche, erfolglose Suche, Einfügen und Löschen an. Welche Variante der Listen ist effizienter im worst-case?

   Lösung:
   Alle Operationen sind im worst-case in $O(n)$, insbesondere auch das Einfügen, weil jetzt die korrekte Position für das neue Element gefunden werden muss. Im worst-case ist das das Ende der Liste.
   Im worst-case ist also die Variante mit unsortierten Listen effizienter.

   Bewertung:
   1 Punkt für die korrekten Laufzeiten (0.25 Punkte pro Operation),
   1 Punkt für die begründete Antwort, welche Variante besser ist.

Aufgabe 8: Laufzeitanalyse

1. (2 Punkte) Zeigen oder widerlegen Sie:
   $f(n) \in O\big(h(n)\big) \wedge g(n) \in O\big(h(n)\big) \Rightarrow f(n) \cdot g(n) \in O\big(h(n) \cdot h(n)\big)$

   Lösung:
   Die Aussage ist wahr. Beweis:

$$f(n) \in O(h(n)) \Rightarrow \exists c_1 > 0\ \exists n_{0,1} > 0\ \forall n > n_{0,1} : f(n) \leq c_1 \cdot h(n)$$
$$g(n) \in O(h(n)) \Rightarrow \exists c_2 > 0\ \exists n_{0,2} > 0\ \forall n > n_{0,2} : g(n) \leq c_2 \cdot h(n)$$

$$\begin{aligned} & f(n) \leq c_1 h(n) \\ \Rightarrow & f(n) \cdot g(n) \leq c_1 h(n) \cdot g(n) \\ \Rightarrow & f(n) \cdot g(n) \leq c_1 h(n) \cdot c_2 h(n) \\ \Rightarrow & f(n) \cdot g(n) \in O(h(n) \cdot h(n)) \quad \text{mit } c := c_1 c_2, n_0 := \max(n_{0,1}, n_{0,2}) \end{aligned}$$

   Bewertung:
   2 Punkte für richtige Lösung mit Begründung,
   0 Punkte falls Antwort nicht begründet.

2. (2 Punkte) Zeigen oder widerlegen Sie: $a^{2n} \in \Theta(a^n)$ für eine Konstante $a > 1$.

   Lösung:
   Die Aussage ist falsch. Beweis:

$$\lim_{n \to \infty} \frac{a^{2n}}{a^n} = \lim_{n \to \infty} a^n = \infty$$

   Die Funktionen wachsen also nicht gleich schnell und es gilt $a^{2n} \notin \Theta(a^n)$.

   Bewertung:
   2 Punkte für richtige Lösung mit Begründung,
   0 Punkte falls Antwort nicht begründet.

3. (2 Punkte) Zeigen oder widerlegen Sie: $\ln(n^2) \in O(\ln(n))$.

   Lösung:
   Die Aussage ist wahr. Beweis:

   $$\ln(n^2) = 2\ln(n)$$

   $\ln(n^2)$ ist also in $O(\ln(n))$ mit $c := 2$ und $n_0 := 1$.

   Bewertung:
   2 Punkte für richtige Lösung mit Begründung,
   0 Punkte falls Antwort nicht begründet.

4. (2 Punkte) Es seien $f(n) := \ln(\ln(n))$ und $g(n) := (\ln(n))^2$.
   Gilt $f \in O(g)$? Gilt $g \in O(f)$? Beweisen Sie Ihre Behauptung.

   Lösung:
   Beweis durch Grenzwertbildung:

   $$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{\ln(\ln(n))}{(\ln(n))^2}$$

   l'Hopital:

   $$\lim_{n\to\infty} \frac{f'(n)}{g'(n)} = \lim_{n\to\infty} \frac{\frac{1}{n\ln(n)}}{2\frac{\ln(n)}{n}} = \lim_{n\to\infty} \frac{1}{2(\ln(n))^2} = 0$$

   $f(n)$ wächst also langsamer als $g(n)$ und es gilt: $f \in O(g)$ und $g \notin O(f)$.

   Bewertung:
   1 Punkt für jede korrekt begründete Antwort.

Vorname	Name	Matr.-Nr.	Seite
			26 / 27

Zusatzblatt 1

Vorname	Name	Matr.-Nr.	Seite
			27 / 27

Zusatzblatt 2