

Datenstrukturen und effiziente Algorithmen

Klausur WS 2014/15

25. Februar 2015

Beginn: 14.00 Uhr

Angaben über den/die Teilnehmer(in)

Name: Shevchenko
 Fachbereich: Informatik
 Matrikelnr.: 2706004

Vorname: Maksym
 Studiengang: B. Sc.
 Übungsgruppenleiter: Daniel

Bewertung (nicht vom Teilnehmer /der Teilnehmerin auszufüllen)

Aufgabe	1	2	3	4	5	6	7	8	9	10	Σ
mögliche Punkte	9	13	7	10	10	9	13	11	6	12	100
erreichte Punkte	5	4,5	6	0	9	0	6	-	-	2	32,5
	AD	PF	DF	CP	AD	An	CP	DF	DF	An	
											Note: <input type="text"/>

Hinweise

- Zum Bestehen der Klausur müssen mindestens 50% der Aufgaben korrekt gelöst werden.
- Kontrollieren Sie diese Klausur auf Vollständigkeit (28 Seiten) und einwandfreies Druckbild.
- Bearbeitung von Aufgaben mit Bleistift oder Rotstift führt zur Nichtbewertung dieser Aufgaben.
- Vergessen Sie nicht, die Angaben zu Ihrer Person auf dem Deckblatt zu machen und schreiben Sie auf jedes Blatt Ihre Matrikelnummer.
- Lösungen müssen entweder direkt nach der Aufgabenstellung niedergeschrieben werden oder auf die Zusatzseiten am Ende der Klausur. Sollten Sie Lösungen auf Zusatzseiten niederschreiben, so verweisen Sie bei der Aufgabe auf den Anhang und im Anhang auf die Aufgabe!
- Falls Sie mehrere Lösungsansätze zu einer Aufgabe formulieren, machen Sie bitte deutlich kennlich, welcher bewertet werden soll!
- Bei Laufzeitabschätzungen ist stets die kleinste obere Schranke anzugeben (also z. B. $\mathcal{O}(n)$ für eine lineare Laufzeit statt der schlechteren Abschätzung $\mathcal{O}(n^2)$).

Viel Erfolg!

1 Asymptotische Laufzeit und \mathcal{O} -Notation

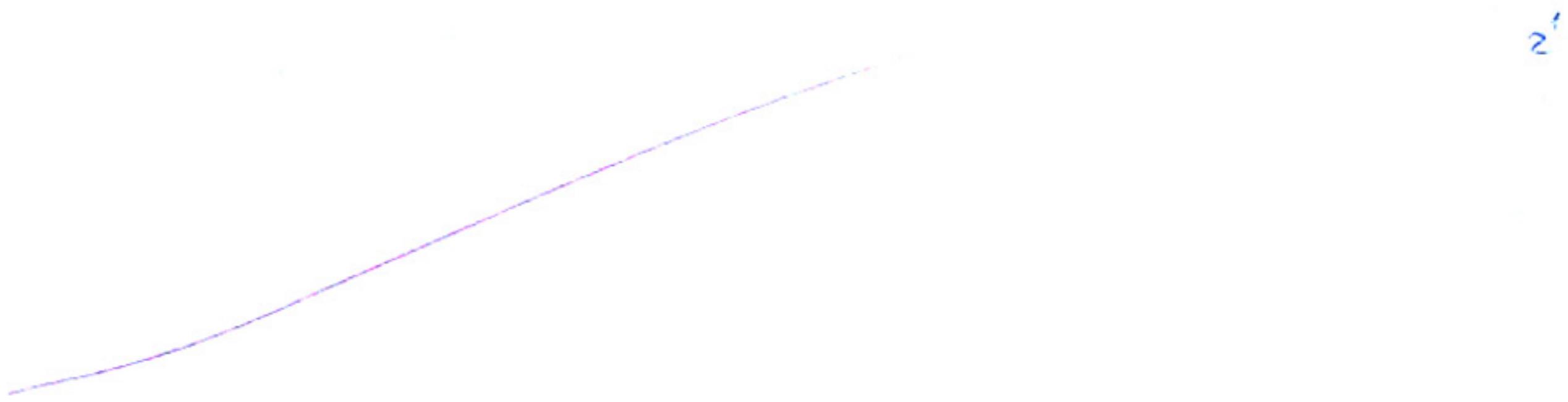
5 | (9 P)

1.A Kurze Antworten

⑤ | (4 P)

Begründen Sie kurz und präzise, ob die folgenden Aussagen richtig oder falsch sind.

- 1) Eine Zahl mit n Ziffern in der Basis 256 hat in der Binärdarstellung $\Theta(n^2)$ Ziffern. ⑤ | (1 P)



- 2) Gegeben sind drei Funktionen $f(n), g(n), h(n): \mathbb{N} \rightarrow \mathbb{N}$.
Falls $f(n) \in \mathcal{O}(g(n))$ und $f(n) \in \Omega(h(n))$ dann gilt $(g(n) + h(n)) \in \Omega(f(n))$ ⑤ | (1 P)

$$\begin{aligned}f(n) &\leq g(n) \\f(n) &\geq h(n)\end{aligned}$$

- 3) Ist $f \in \Omega(n)$ und $f(n) > 0 \forall n$, dann gilt: $\frac{n^3}{f(n)} \in \mathcal{O}(n^2)$. ⑤ | (2 P)

5 | (5 P)

1.B Rekurrenzen

Geben Sie für die folgenden Rekurrenzen jeweils eine explizite Abschätzung des Wachstumsverhaltens in \mathcal{O} -Notation an. Es sei jeweils $n \in \mathbb{N}$.

1) $T(n) = 16 T\left(\frac{n}{4}\right) + 3n^2, T(1) = 0$ (1 P)

Laut Master Theorem:

$$a = 16, b = 4, \alpha = 2$$

$$\log_4 16 = 2 \Rightarrow T(n) \in \mathcal{O}(n^2 \log n) \quad \checkmark$$

2) $T(n) = 3 T\left(\frac{n}{2}\right) + c \cdot n, T(1) = 0$ (1 P)

Laut Master Theorem:

$$a = 3, b = 2, \alpha = 1$$

$$\log_2 3 > 1 \Rightarrow T(n) \in \mathcal{O}(n^{\log_2 3}) \quad \checkmark$$

3) Gegeben die Rekursionsgleichung $f(n) = f(n-1) \cdot f(n-1), f(1) = 2, n \in \mathbb{N}$ 3 | (3 P)

Zeigen Sie mittels vollständiger Induktion, dass $f(n) \in \Omega(2^n)$.

Behauptung: $f(n) \in \Omega(2^n)$

$$\Rightarrow f(n) \geq 2^n$$

Induktion nach n .

Ind Anf: $f(1) = 2, f(1) \geq 2^1$

Ind Schr. ~~$f(n+1) \geq 2^{n+1}$~~ ~~$f(n+1) = f(n) \cdot f(n)$~~ ~~$2^n \cdot 2^n = 2^{2n}$~~ ~~$2^{2n} \geq 2^{n+1}$~~

~~Ind Schr.~~ $\Rightarrow f(n+1) \cdot f(n+1-1) \geq 2^{n+1} \Rightarrow f(n) \cdot f(n) \geq 2^{n+1}$

~~Ind Schr.~~ $2^n \cdot 2^n \geq 2^{n+1} \Rightarrow 2^{2n} \geq 2^{n+1} \Rightarrow 2n \geq n+1 \Rightarrow 2n - n - 1 \geq 0$

$n-1 \geq 0$ ~~aus~~ als $n \in \mathbb{N}$ \checkmark

2 Suchen & Sortieren

(13 P) 4,5

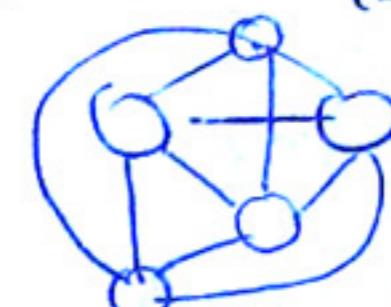
2.A Wahr oder falsch?

Begründen Sie **kurz** und **präzise**, ob die folgenden Aussagen richtig oder falsch sind. (7 P)

- 1) Die maximale Anzahl von Kanten in einem Baum $T = (V, E)$ liegt in $\Omega(|V|^2)$. (1 P)

bis maximale Anzahl von Kanten
immer $|V| - 1$.

$|V| - 1 \leq |V|^2 \Rightarrow$ es liegt nicht in $\Omega(|V|^2)$



D.h. die Aussage ist falsch.

- 2) Das k -te kleinste Element einer unsortierten Liste von n Zahlen kann mit einer erwarteten Zeitkomplexität von $T(n) \in O(n)$ bestimmt werden. (1 P)

Nein. Es gibt Algorithmen, die haben Best Case Laufzeit von $O(n)$, aber gibt keine, die haben Worst Case besser als $O(n \cdot \log n)$.

D.h. die Aussage ist falsch. Quicksort.

- 3) Wenn man für InsertionSort (sortieren durch sukzessives Einfügen an der richtigen Stelle) die Einfügepositionen in einem Array mit $n \in \mathbb{N}_0$ Elementen mittels binärer Suche bestimmt, kann man in $O(n \log n)$ sortieren. (2 P)

Die Aussage ist richtig.

Weil man ~~Indexing~~ braucht $O(n)$ für Indexing ~~von~~ von Binary Heap und $O(\log n)$ für Inserting ~~im~~ im Array nicht.
Zusammen $O(n \cdot \log n)$

WA

Begründen Sie **kurz und präzise**, ob die folgenden Aussagen richtig oder falsch sind.

- 4) In einem Min-Heap kann zu einem Element x das nächstgrößere Element y in $O(\log n)$ gefunden werden. (1 P) ✓

~~Min-heap ist sortiert, und alle Elemente~~

Min, weil ~~im~~ im worst case muss

man n -mal Extract-min operationen
heute einander durchführen.

D.h. Laufzeit $\in O(n \cdot \log n)$ ~~Naja... ja~~

b.h. die Aussage ist falsch. Ja zumindest ~~ja~~

- 5) In einer SkipList mit n Elementen ist die erwartete Suchzeit $O(\log n)$, dafür ist der erwartete Speicherbedarf dieser Datenstruktur $\Omega(n \log n)$. (1 P) O, S.

Ja, ~~Average~~ Suchzeit von SkipList ist $O(\log n)$
(Worst ist $\Theta(n)$). v.

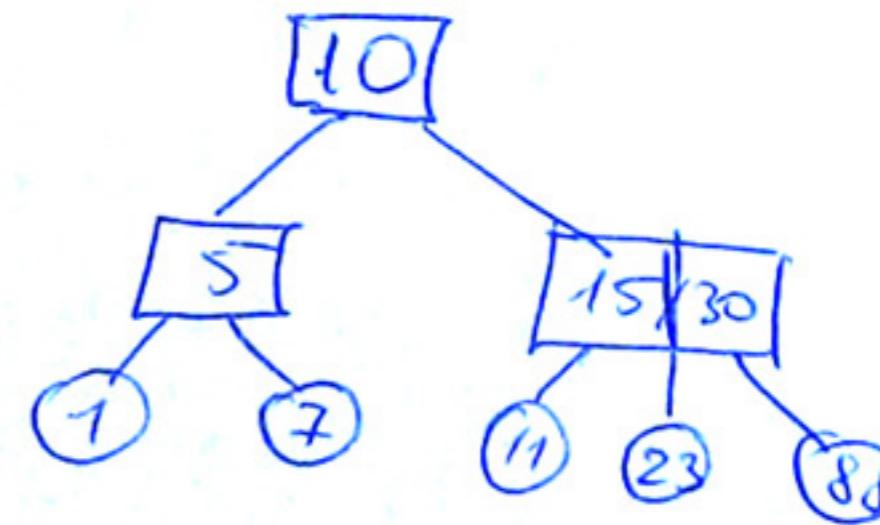
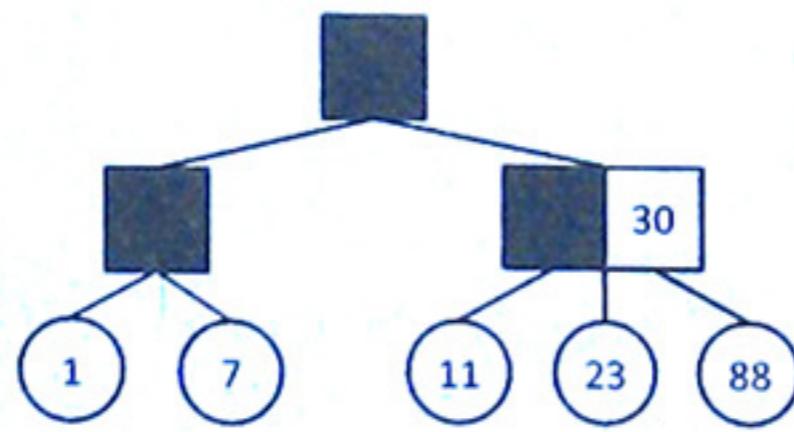
Aber erwartete Speicherbedarf ist $\Theta(n \log n)$

$\Rightarrow f(n) \leq g(n) \Rightarrow f(n) \neq g(n) \Rightarrow f(n) \notin \Omega(g(n))$

$\Rightarrow f(n) \notin \Omega(n \log n)$ v.

Aussage ist richtig (dann falsch!)

- 6) Der folgende Baum ist ein gültiger (2,5)-Baum. (1 P) ✓



~~Min, weil verletzt Eigenschaften von (2,5)-Bäumen~~

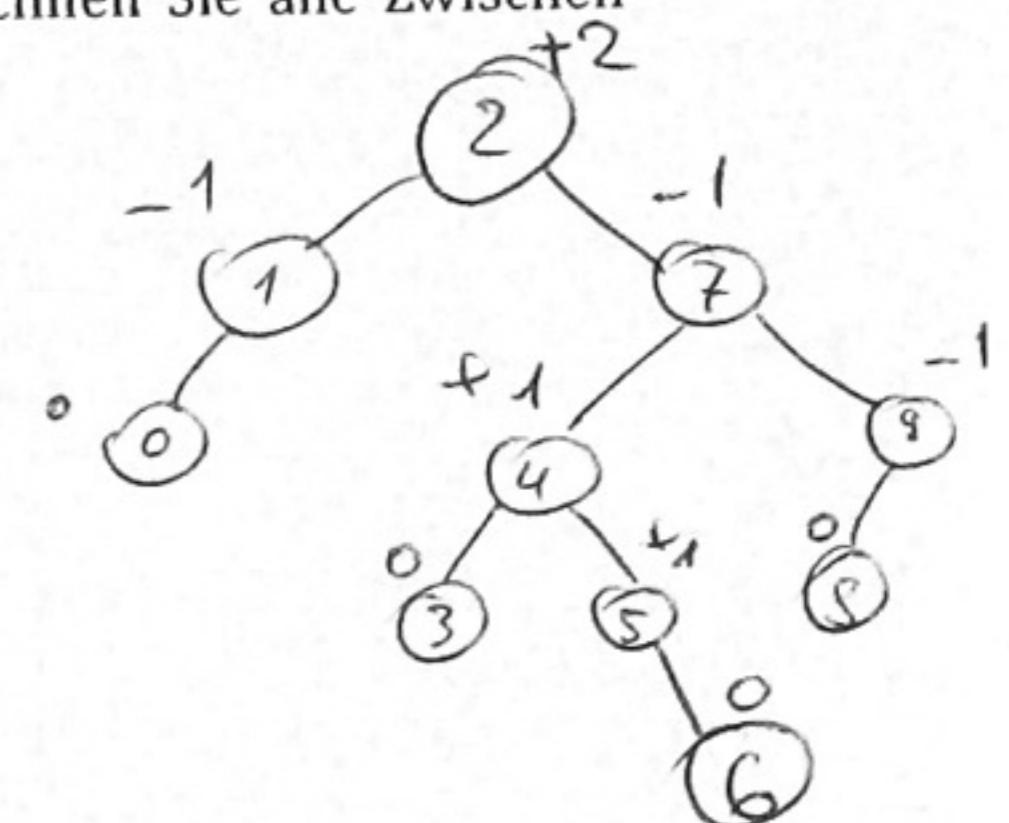
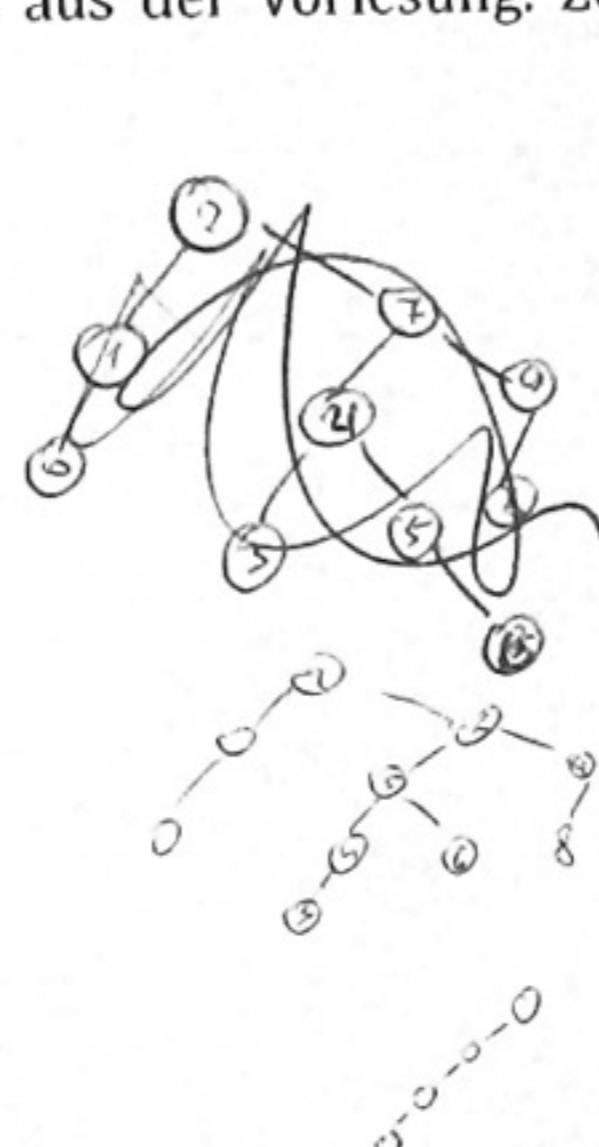
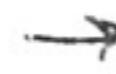
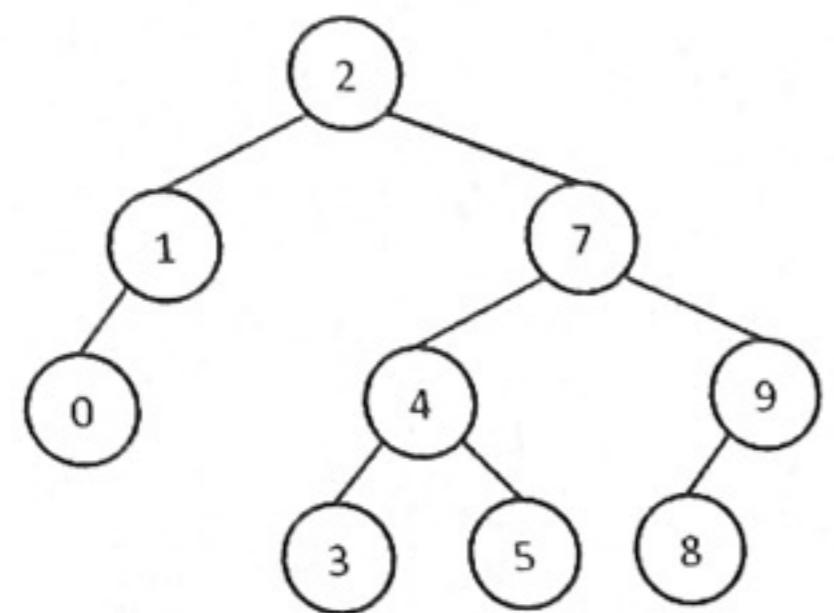
~~Z.B. # Blättern müssen in (2,5)-Bäumen so oft~~
 ~~$2^t \leq n \leq 5^t$ sein. $t=2$ $2^3 \leq n \leq 5^2$~~
 ~~$n=8$ ~~4~~~~ v.

Ja. Weil kein Eigenschaft von (2,5)-~~Baum~~ Baum verletzt
 $2^{tief} \leq 8 \leq 5^{tief}$ ✓ $2^3 \leq 8 \leq 5^3$ ✓; $1 \leq 5 \leq 7$; $11 \leq 15 \leq 23 \leq 30 \leq 88$; $5 \leq 10 \leq 30$

2.B AVL-Baum: Insert

(3 P)

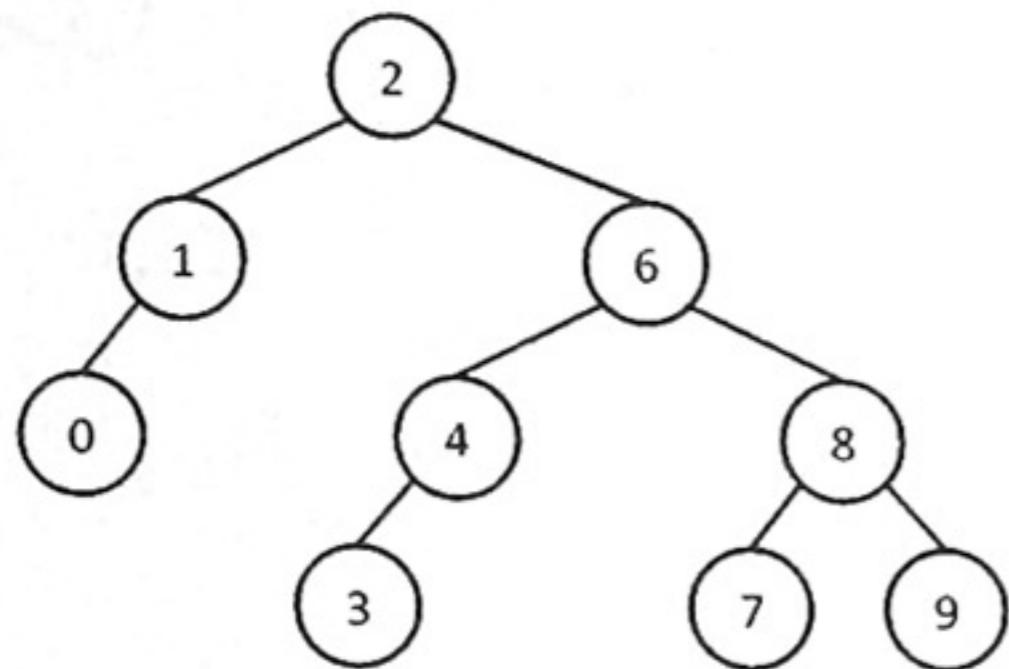
Gegeben ist folgender AVL-Baum. Führen Sie die Operation `insert(6)` durch. Rebalancieren Sie den Baum gegebenenfalls mit dem Verfahren aus der Vorlesung. Zeichnen Sie alle Zwischen-schritte auf!



2.C AVL-Baum: Delete

(3 P)

Gegeben ist folgender AVL-Baum. Führen Sie die Operation `delete(1)` durch. Rebalancieren Sie den Baum gegebenenfalls mit dem Verfahren aus der Vorlesung. Zeichnen Sie alle Zwischen-schritte auf!



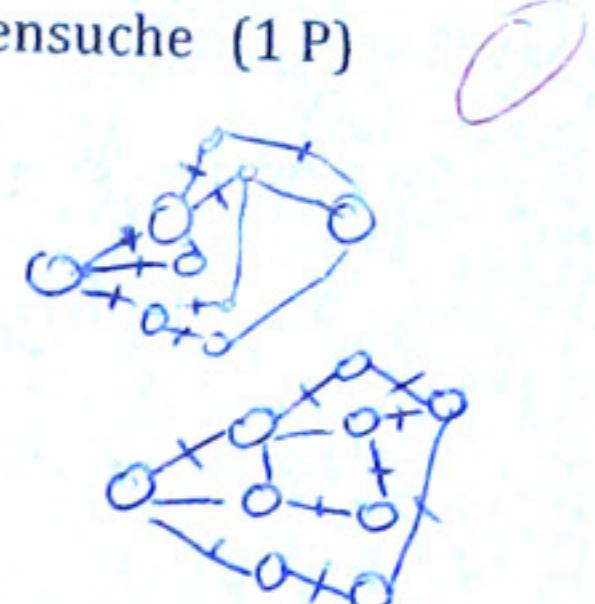
3 Graphen

(7 P)

Begründen Sie **kurz und präzise**, ob die folgenden Aussagen richtig oder falsch sind.

- 1) In einem ungewichteten Graphen können die kürzesten Wege sowohl mit Breitensuche als auch mit Tiefensuche in $\mathcal{O}(|V| + |E|)$ gefunden werden. (1 P)

Ja. Weil die Beide haben Laufzeit von $\mathcal{O}(|V| + |E|)$

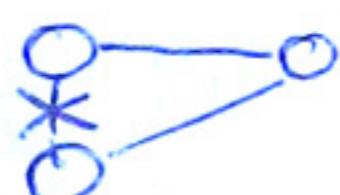


D.h. Aussage ist richtig

DFS findet nicht immer
kürzeste Wege

- 2) In einem bipartiten Graphen haben alle Zyklen eine gerade Anzahl an Kanten. (1 P)

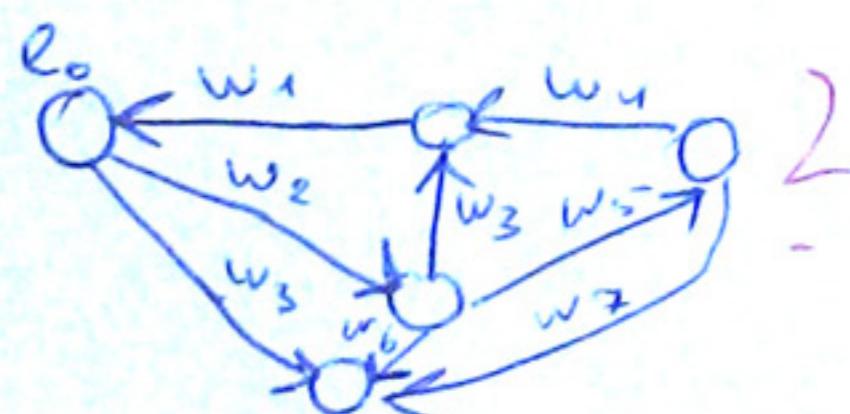
Ja. Weil zwei Knoten von einer Seite ~~nur mit einander~~ nur zwischen Knoten von anderen Seiten verbunden sein können.



D.h. Aussage ist richtig

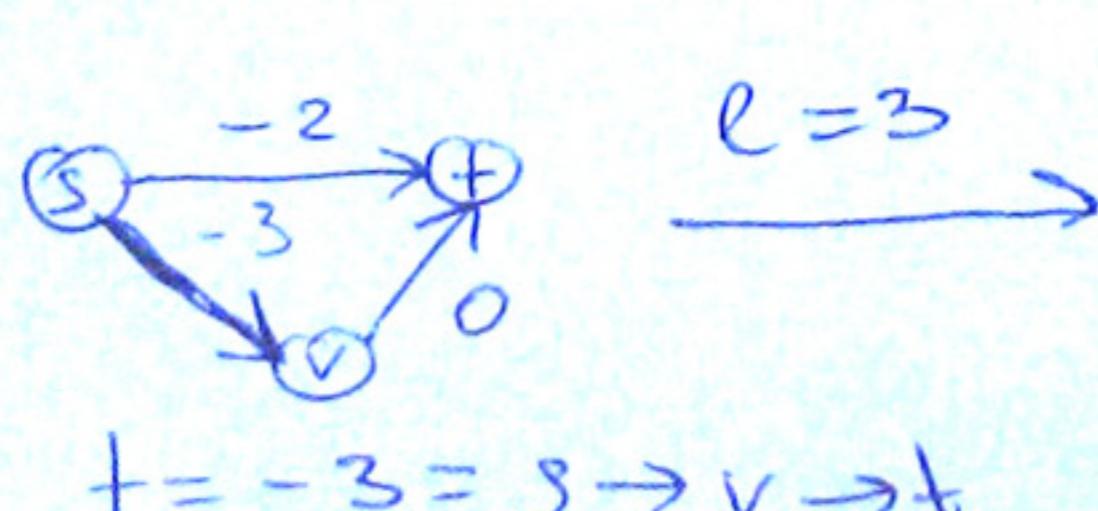
- 3) Gegeben sei ein gerichteter Graph $G = (V, E)$ mit Kantengewichten $w: E \rightarrow \mathbb{R}$ sowie ein Knoten $v \in V$. Sei $G' = G$ mit Kantengewichten $w': E \rightarrow \mathbb{R}$ und sei $\ell > 0$. Für alle $e \in E$ gelte $w'(e) = w(e) + \ell$. Dann sind die kürzesten Wege von v zu allen anderen Knoten in G' und G gleich. (2 P)

2

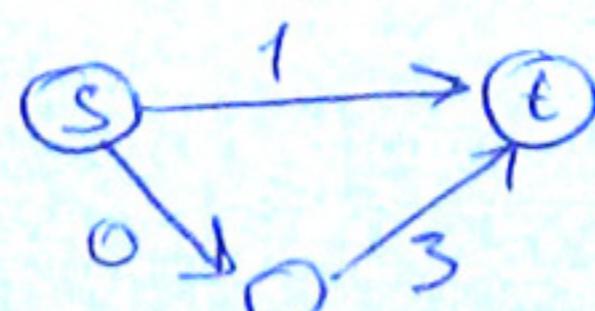


Nein, die Aussage ist Falsch.

Gebe Beispiel:



$$t = -3 = s \rightarrow v \rightarrow t$$



$$t = 1 = s \rightarrow t$$



Begründen Sie **kurz und präzise**, ob die folgenden Aussagen richtig oder falsch sind.

- 4) In einem bipartiten Graphen $G = (S \cup T, E)$ mit $|S| < |T|$ gibt es ein maximales Matching M . Wenn zu S ein Knoten v hinzugefügt wird, der mit allen Knoten aus T verbunden ist, dann gibt es Matching M' mit $|M'| > |M|$. (3 P) 3

~~Richtig.~~

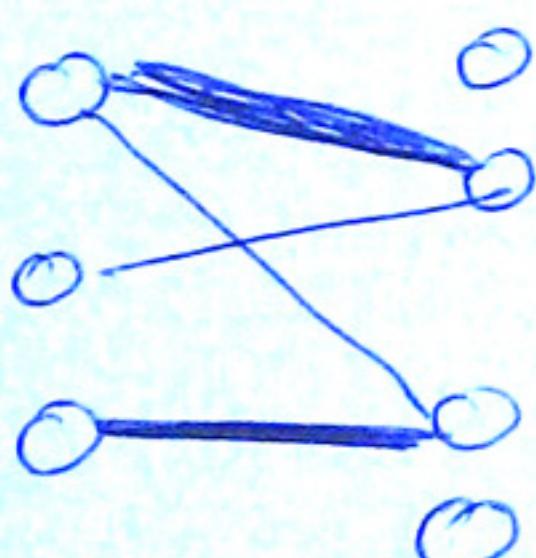
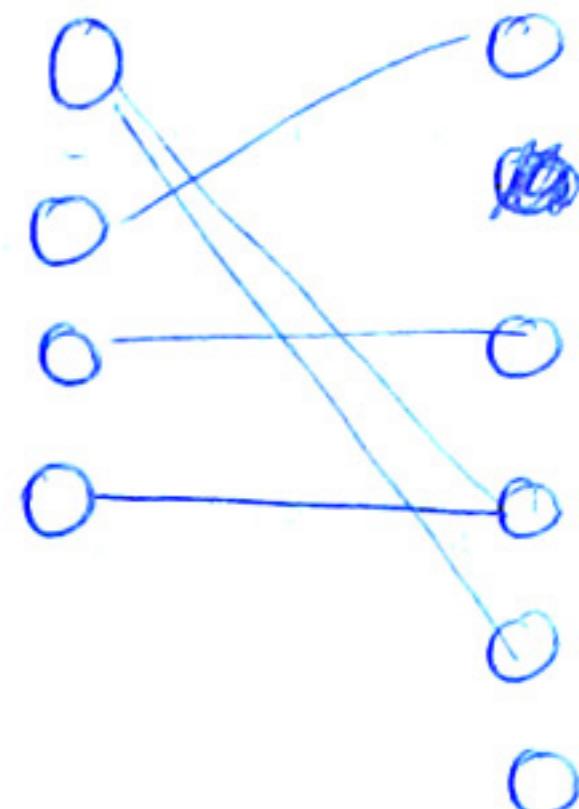
Weil ~~es~~ gibt immer
eine oder mehrere ~~Knoten~~
Knoten von T die

mit Knoten von S in

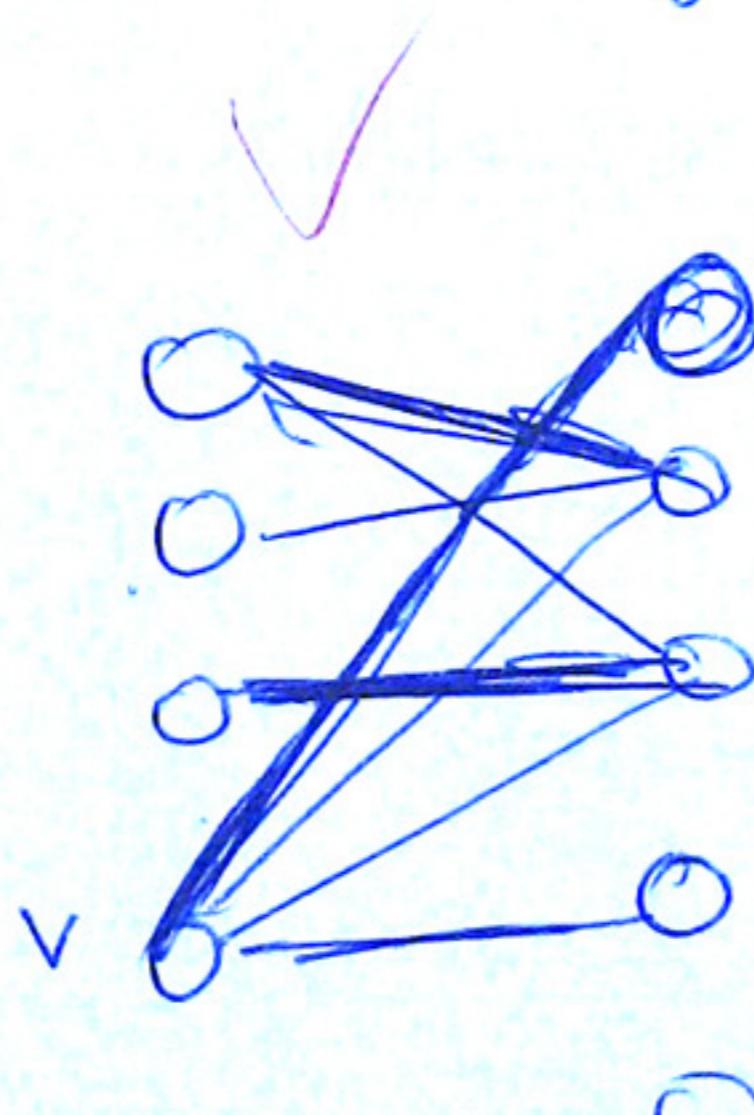
maximale Matching haben keine Matching.

Und wenn man nun Knoten v hinzufügt,
der mit allen Knoten ~~aus~~ T verbunden sind,

dann ~~Knoten~~ kann man neu Match
zwischen v und nicht gemacht Knoten aus T
machen.



$$|M| = 2$$



$$|M'| = 3$$

4 Paare von Zahlen verschwindender Summe

0/ (10 P)

Sei $L := (l_0, \dots, l_{n-1})$ eine unsortierte Liste von reellen Zahlen der Länge n . Es sollen alle verschiedenen 2-Tupel (l_i, l_j) bestimmt werden, für die gilt $l_i + l_j = 0$ für $i, j \in \{0, \dots, n-1\}$ und $i < j$.

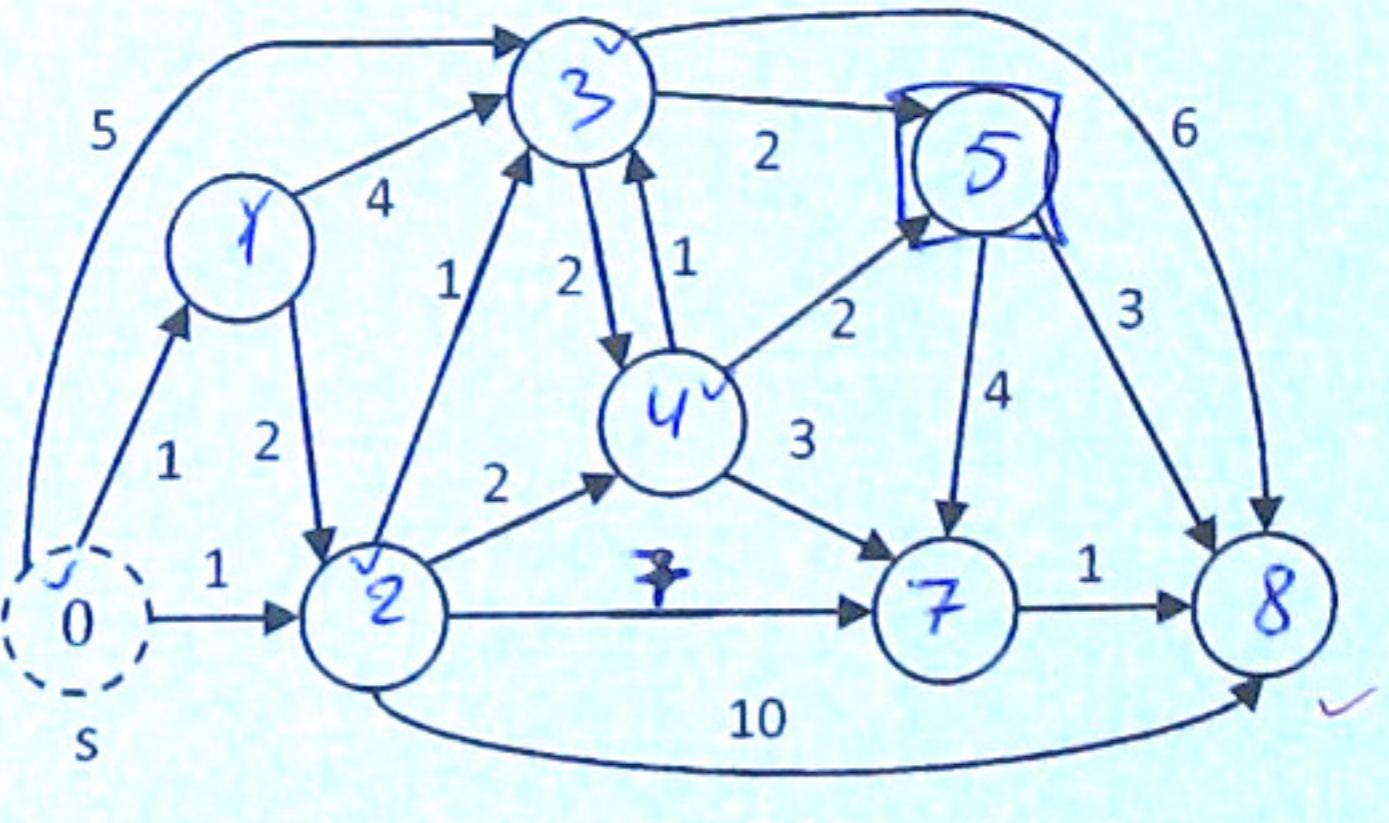
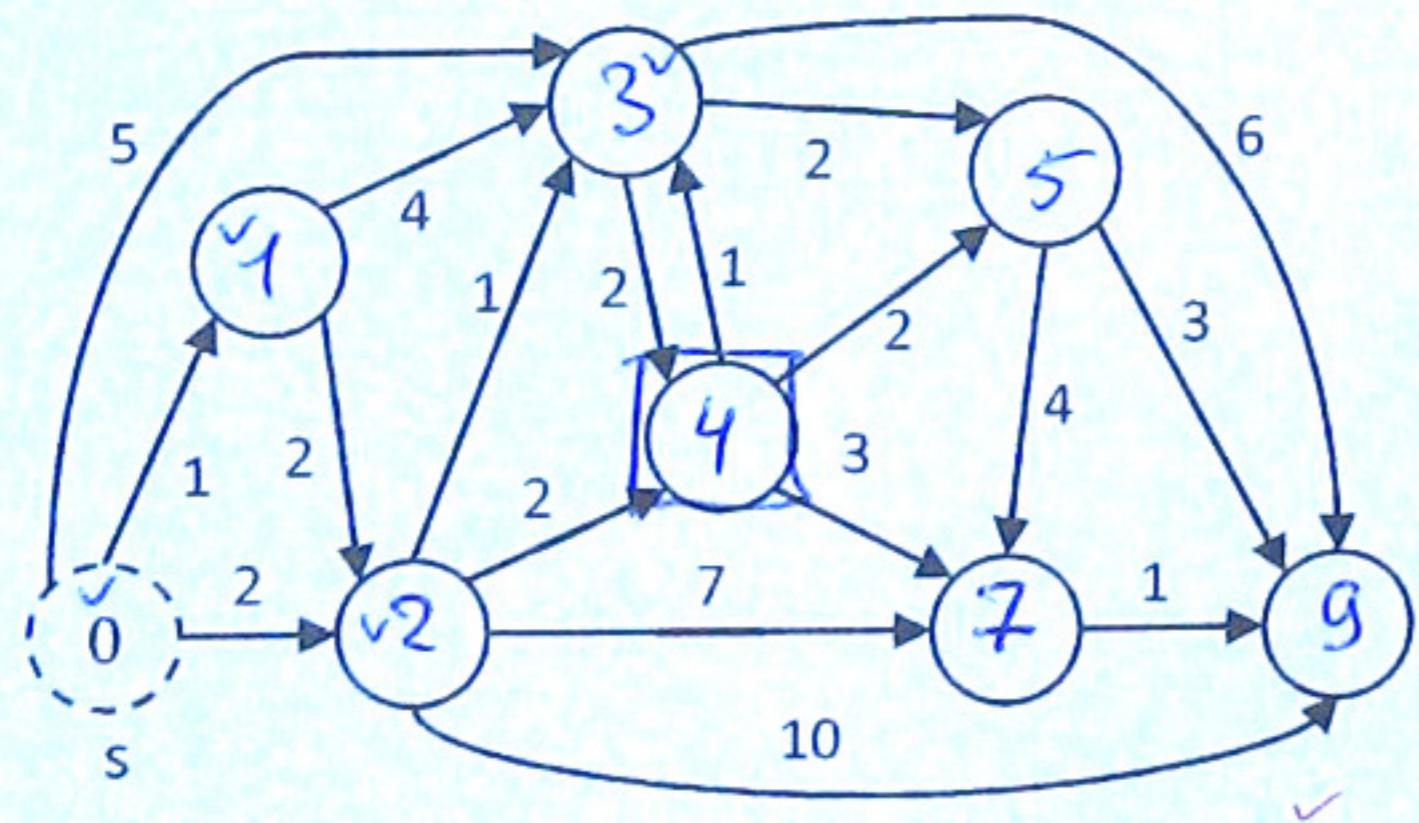
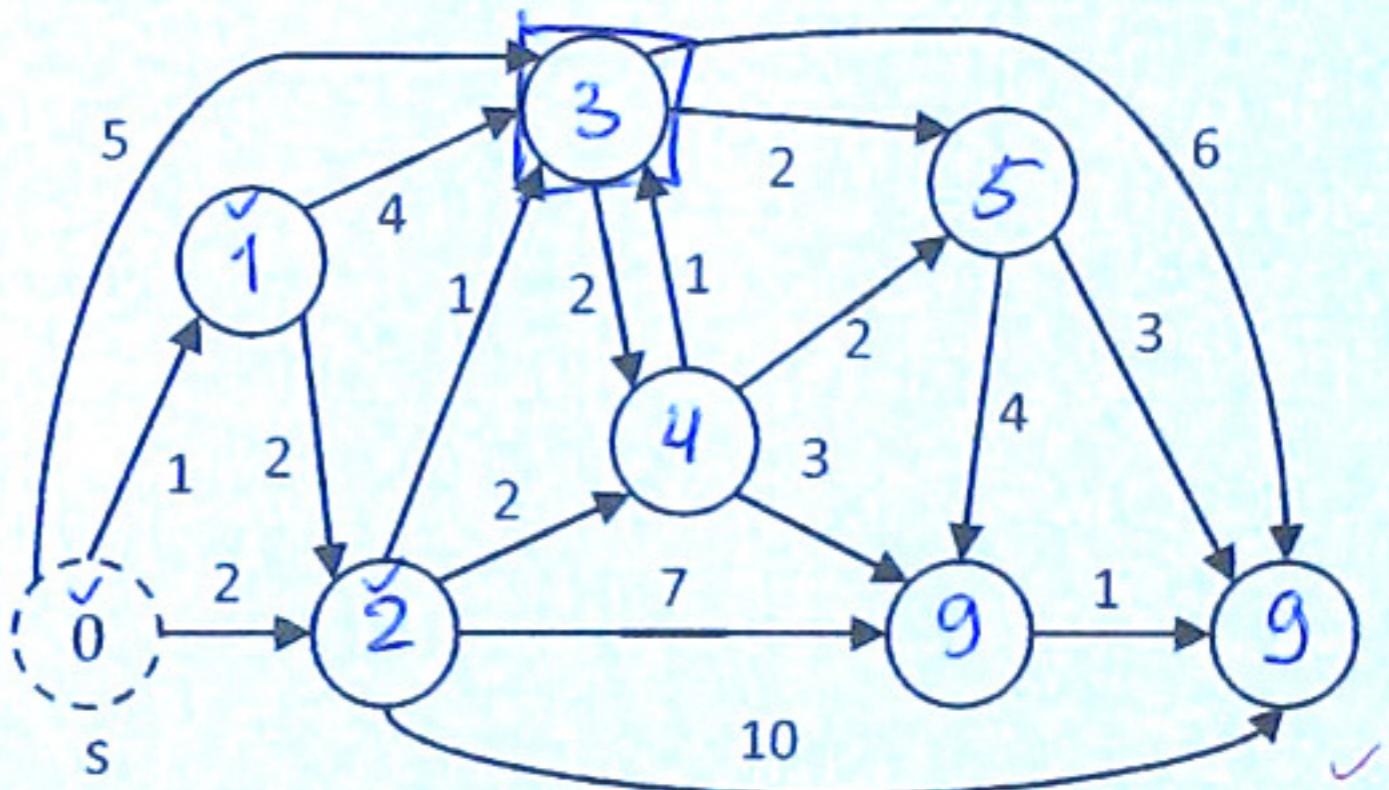
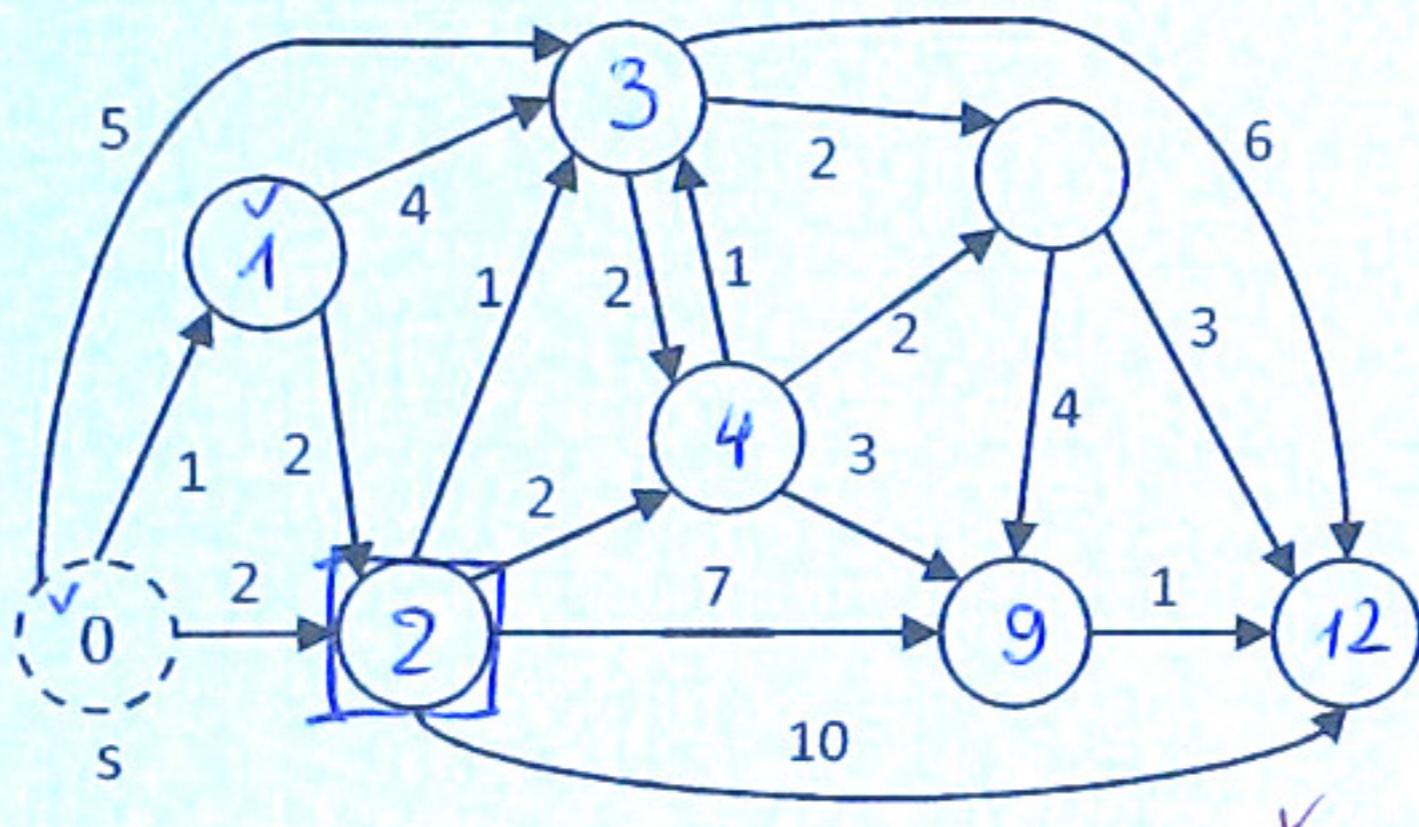
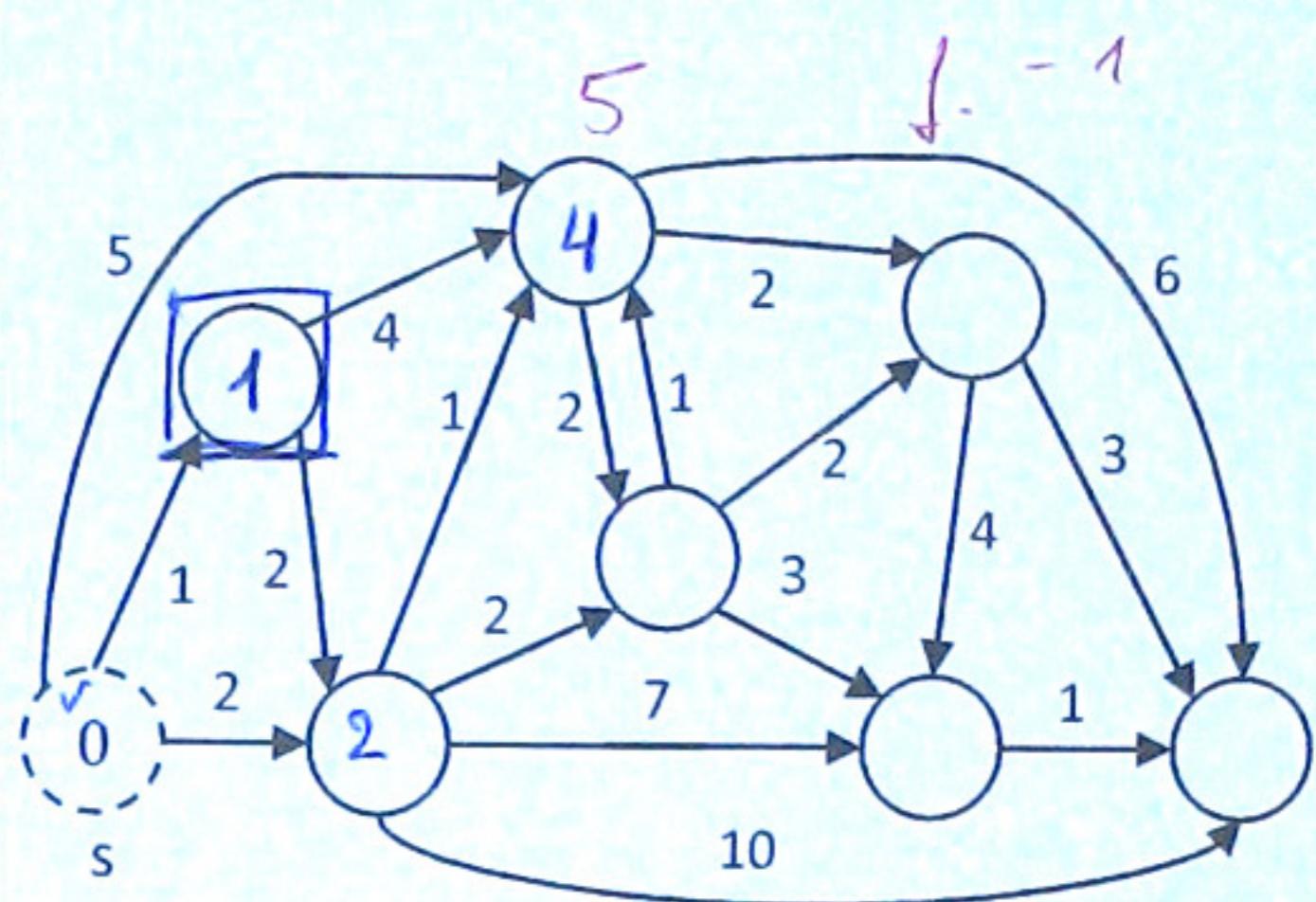
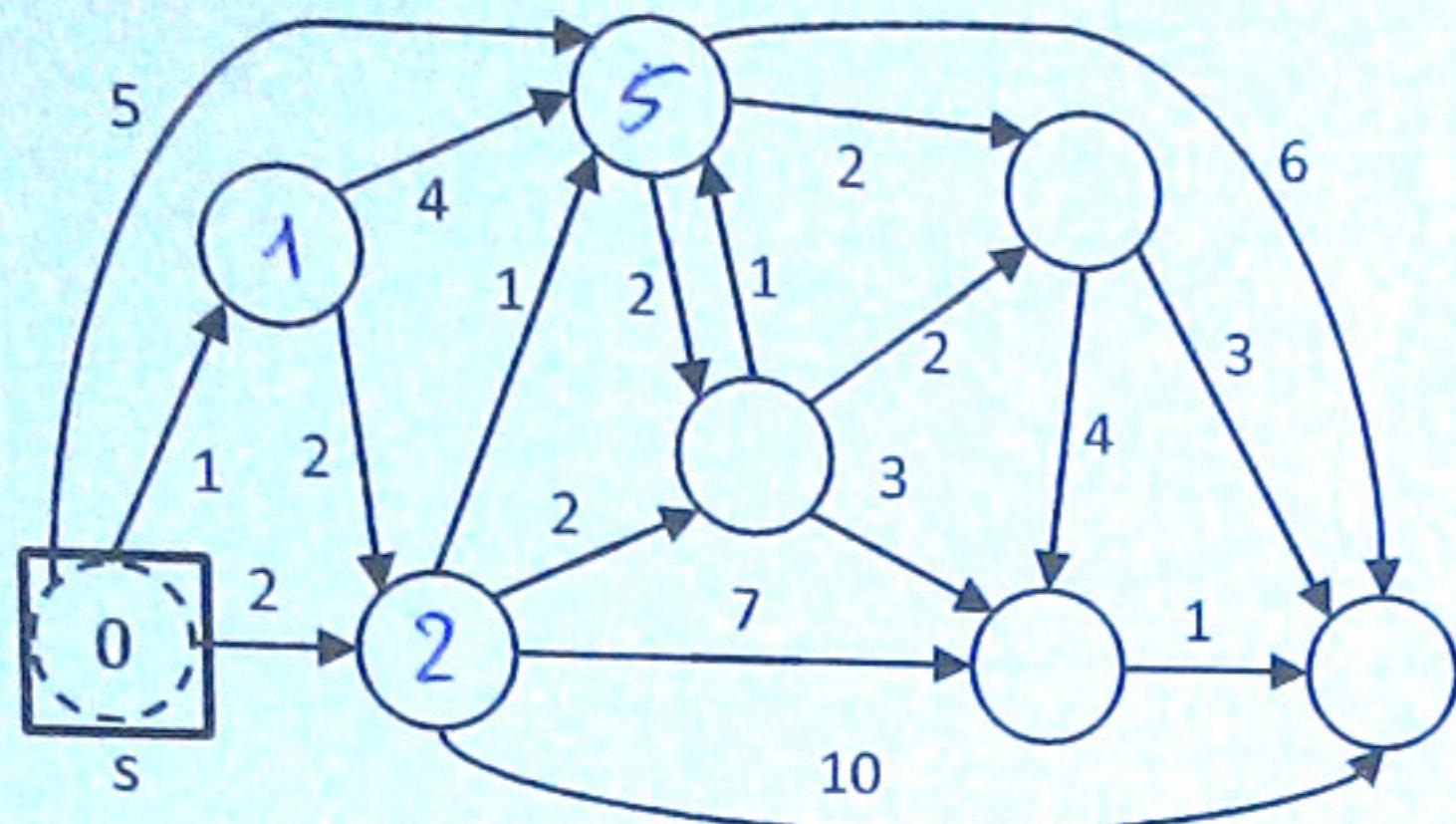
Beispiel: $L = (+2, +1, +\pi, -1, +e, -\pi, +1, -1) \rightarrow \{(-\pi, +\pi), (-1, +1)\}$

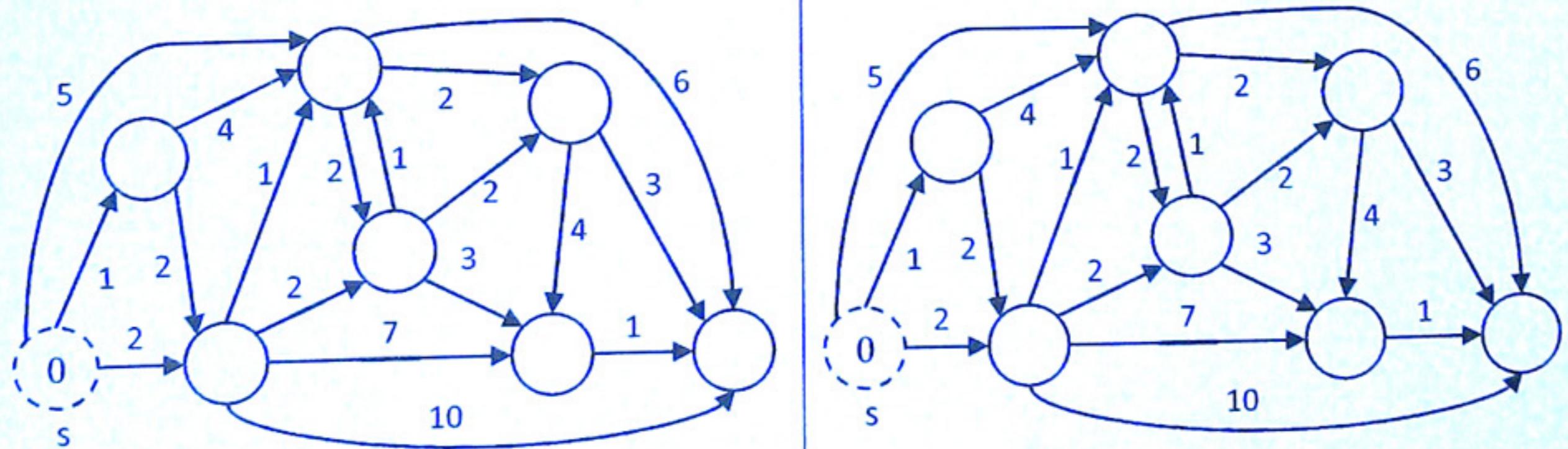
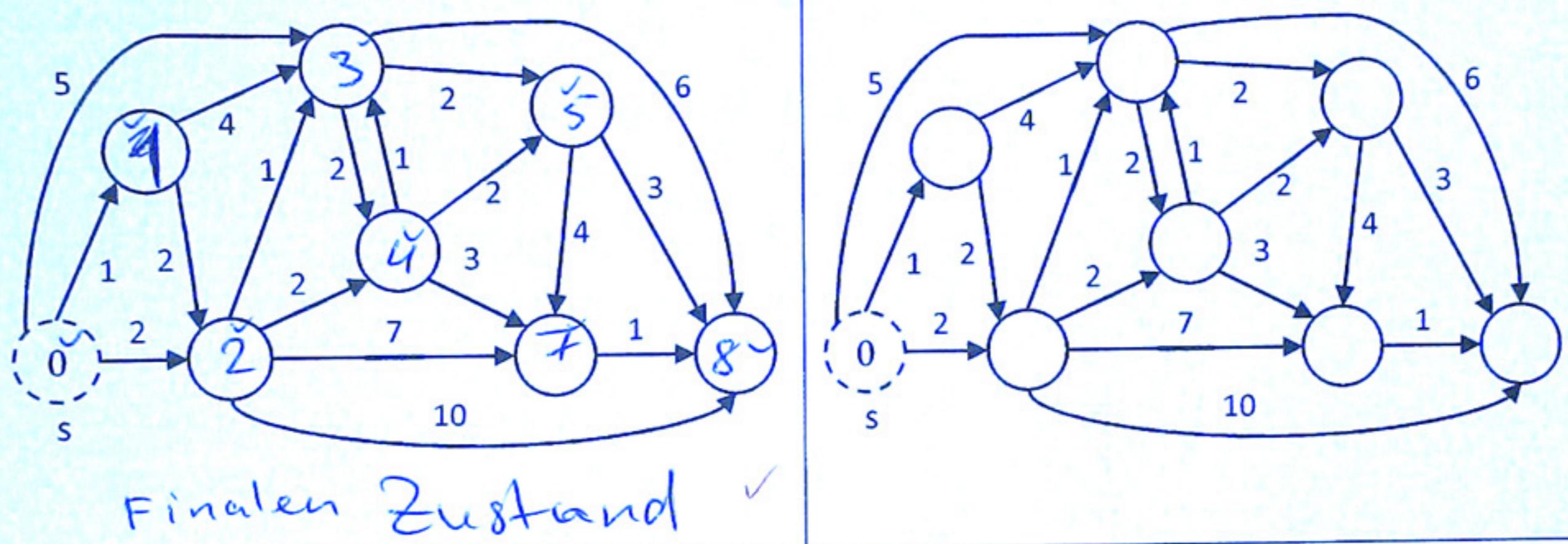
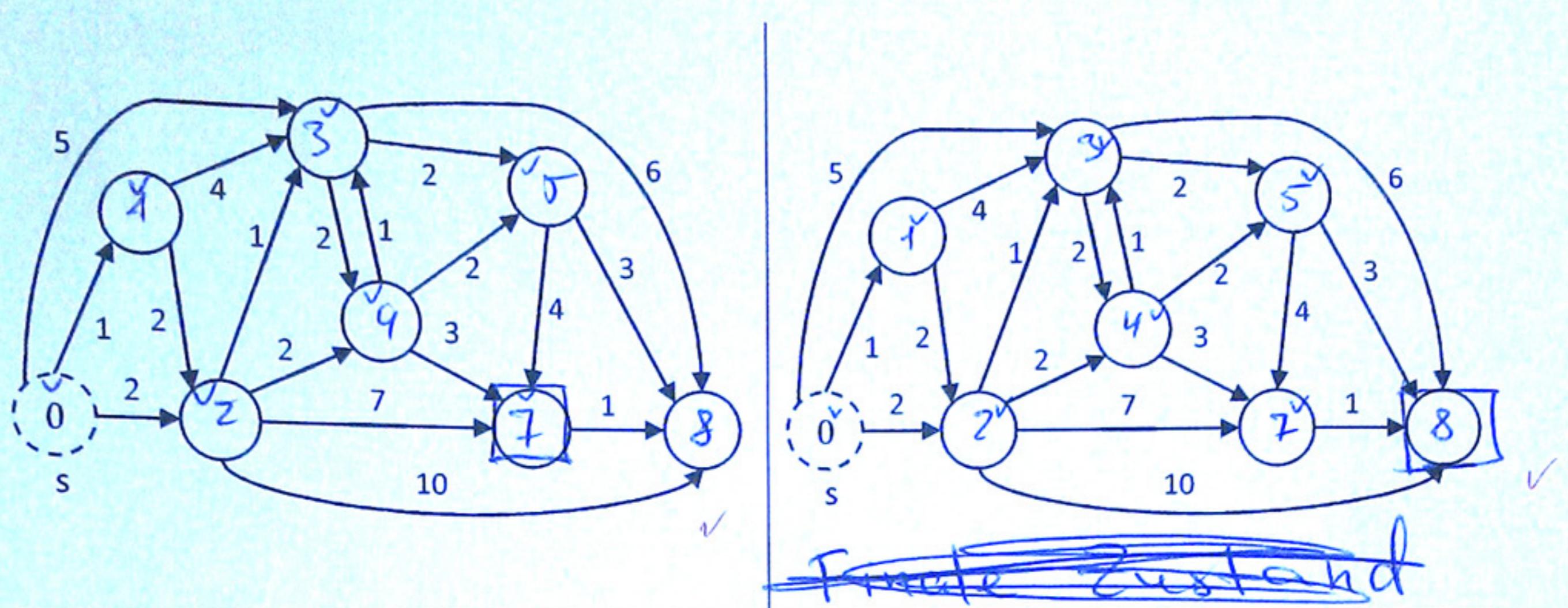
- a) Geben Sie einen effizienten Algorithmus an, der das Problem mit einer Worst-Case-Zeitkomplexität von $T(n) \in O(n \log n)$ löst. (8 P)
- b) Angenommen n ist groß und die Einträge in der Liste kommen aus der Menge $M := \{-9, -8, \dots, 0, \dots +8, +9\}$. Geben Sie stichwortartig einen effizienten Algorithmus an, der das Problem in $T(n) \in O(n)$ löst. (2 P)

5 Algorithmus von Dijkstra

3 / (10 P)

Gegeben ist der unten abgebildete gerichtete Graph. Finden Sie die kürzesten Wege von s zu allen anderen Knoten. Führen Sie dazu den Algorithmus von Dijkstra mit Hilfe der Abbildungen aus. Markieren Sie jeweils den momentan bearbeiteten Knoten mit einem Rechteck und tragen Sie in jedem Schritt in alle Knoten die aktuellen Distanzwerte ein. Kennzeichnen Sie die Abbildung, die dem finalen Zustand entspricht.



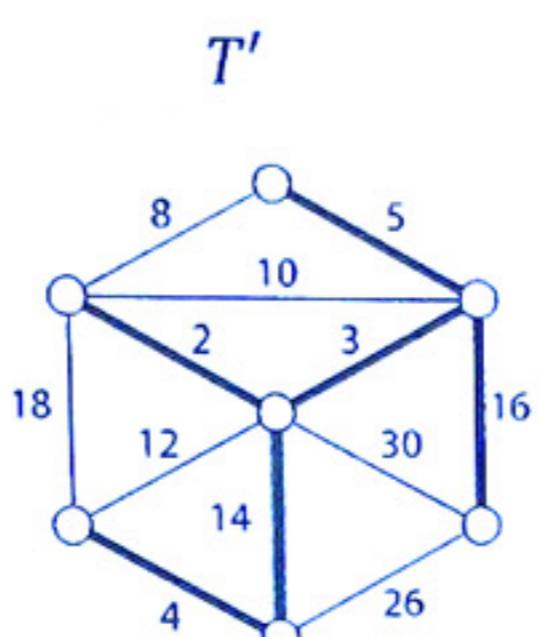
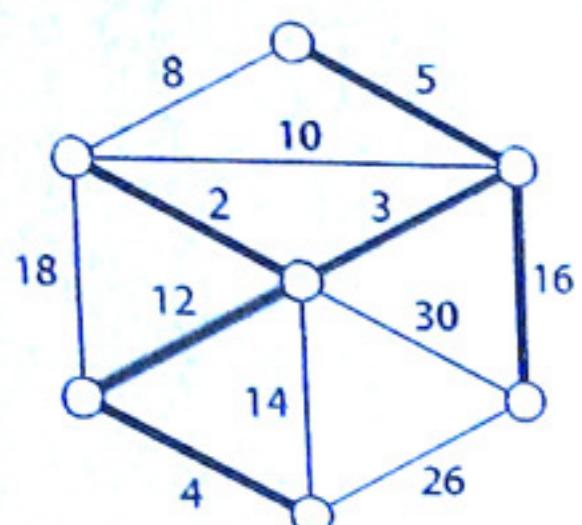


9 / 10

6 Zweitbester MST

Gegeben ein gewichteter, ungerichteter Graph $G = (V, E)$ mit paarweise verschiedenen Kantengewichten $w: E \rightarrow \mathbb{R}$ und $|E| \geq |V|$. Sei \mathcal{T} die Menge aller minimalen Spannbäume auf G und sei $T \in \mathcal{T}$ ein minimaler Spannbaum. Wir definieren den zweitbesten Spannbaum T' so, dass $w(T') = \min_{T'' \in \mathcal{T} - \{T\}} \{w(T'')\}$.

Beispiel: T



- a) Zeigen Sie, dass G Kanten $(u, v) \in T$ und $(x, y) \notin T$ enthält, so dass $T - \{(u, v)\} \cup \{(x, y)\}$ ein zweitbester Spannbaum ist. (3 P)

b) Sei $\max[u, v]$ eine Kante mit maximalem Gewicht auf dem einfachen Pfad von u nach v im minimalen Spannbaum T . Geben Sie einen $\mathcal{O}(|V|^2)$ -Algorithmus an, der bei gegebenem T die $\max[u, v]$ für alle $u, v \in V$ berechnet. (3 P)

c) Geben Sie einen effizienten Algorithmus an, der den zweitbesten Spannbaum auf G berechnet. (3 P)

$$a) T \cup T' = \{(u, v)\} \cup \{(x, y)\} \Rightarrow \\ T - \{(u, v)\} \cup \{(x, y)\} = T'$$

c) Mit PRIM kann man den zweit besten Spannbaum G berechnen. ()

7 Vermischtes

6/ (13 P)

Lösen Sie die folgenden Aufgaben. Halten Sie Ihre Antworten **kurz** (stichwortartig) und **präzise**.

7.A Hashing

0/ (2 P)

Gegeben ist eine Hashtabelle mit m Plätzen, die eine 1-universelle Hashfunktion $h: U \rightarrow H$ verwendet und Kollisionen mittels Verkettung behandelt. Die Tabelle wurde mit n Schlüsseln gefüllt. Geben Sie die Wahrscheinlichkeit dafür an, dass der erste Platz in der Tabelle leer ist.

$$\frac{1}{m^n}$$

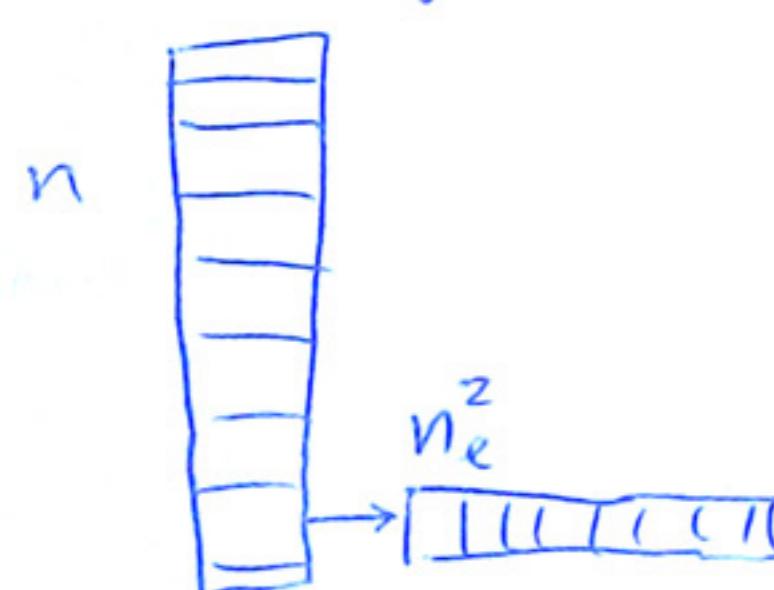
Falsch

7.B Bilder finden

2/ (3 P)

Ein Bild soll in einer Bilddatenbank im **Worst-Case** in $\mathcal{O}(1)$ gefunden werden. Der Datenbankinhalt wird einmal eingelesen und ändert sich danach nicht mehr. Geben Sie einen Algorithmus bzw. eine geeignete Datenstruktur dafür an! Was können Sie über die Speicherkomplexität Ihres Ansatzes sagen?

Man kann dafür Universelles Perfect Hashing benutzen, das war auf der Vorlesung dargestellt.



Speicherkomplexität ist $\mathcal{O}(n) + \sum_{l=0}^n \mathcal{O}(n_e^2)$

d.h. es liegt in $\mathcal{O}(n^2)$

sucht man Hashfunktion, so dass Anzahl der Kollisionen gering ist. Und für jede Kollision erzeugt Haupthash-Tabelle. Dann für jeder Platz wo Kollisionen gibt erzeugt man neue Tabellen Größe n_e^2 (wo e ist Anzahl von Kollisionen) und wählt Hashfunktion so dass das es keine Kollisionen mehr gibt.

7.C Dijkstra

2/ (3 P)

Gegeben ein gerichteter Graph mit positiven Kantengewichten. Sie möchten die Knoten in einem stets unsortierten Array speichern und keine Prioritätswarteschlange verwenden. Wie ist die zu erwartende Laufzeitkomplexität einer darauf basierenden Implementierung des Algorithmus von Dijkstra (für das Single-Source-Shortest-Path-Problem)?

mit ~~min~~ Heap liegt Dijkstra Algorithm im $\Theta(|V| + |E| \cdot \log |V|)$

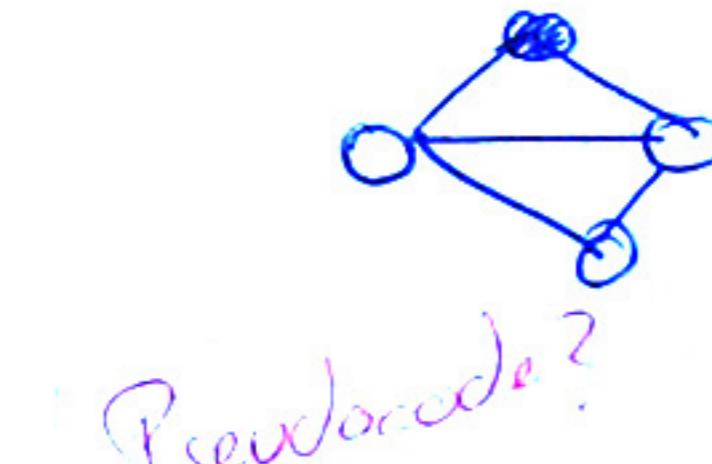
Die Laufzeit von Suche im unsorted Array ist $\Theta(n) = \Theta(|V|) \Rightarrow$ Laufzeit von Dijkstra mit unsortierte Array als Speicher ist $\Theta(|V| + |E| \cdot |V|) \in \underline{\Theta(|V|^2)}$

7.D 2-Färbbarkeit

2/ (5 P)

Sei $G = (V, E)$ ein ungerichteter Graph. Wir nennen G 2-färbbar, wenn man seine Knoten so mit zwei Farben markieren kann, dass keine zwei benachbarte Knoten die gleiche Farbe haben. Geben Sie einen Algorithmus in Pseudocode an, der im Worst-Case in $\Theta(|V| + |E|)$ entscheidet, ob der Graph 2-färbbar ist.

farbe = 0
 Markieren (V, E, farbe)
~~for each v aus V~~
 for each v aus V



Pseudocode?

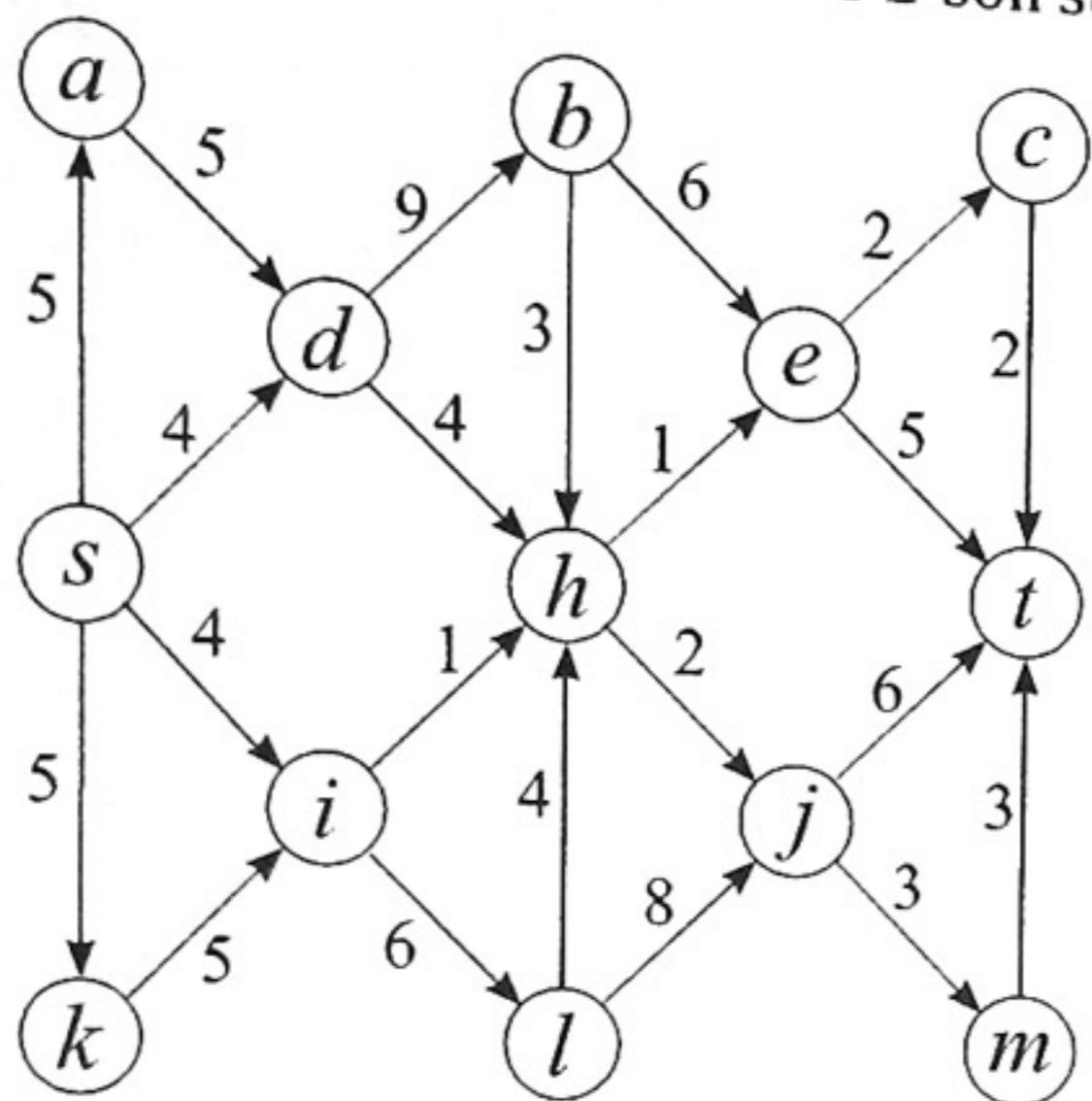
Behutzen wie erst Tiefensuche um alle Knoten nach einander in 2 farben zu färbten.

Und danach Breitensuche um bestimmen, ob Nachbaren von alle Knoten haben andere Farbe. wozu?

8 MKM-Verfahren

(11 P)

Gegeben sei folgendes Flussnetzwerk G mit dem Startknoten s und dem Zielknoten t . Sei $c: E \rightarrow \mathbb{N}$ eine Kapazitätsfunktion sowie $f: E \rightarrow \mathbb{N}_0$ eine Flussfunktion auf den Kanten. Der Fluss sowie die Kapazität auf einer Kante $e \in E$ soll stets mit der Notation $f(e)/c(e)$ vermerkt werden.



- Bestimmen Sie für den initialen Fluss mit Wert $f = 0$ das geschichtete Restnetzwerk G_f^L . Dieses soll nur Knoten und Kanten enthalten, die auf einem kürzesten Weg von s nach t liegen. (2 P)
- Bestimmen Sie den Knoten mit minimaler Knoten(rest)kapazität in G_f^L . (1 P)
- Bestimmen Sie einen Sperrfluss g in G_f^L nach dem Verfahren von Malhotra, Kumar und Maheshwari. Schreiben Sie Ihre einzelnen Schritte nachvollziehbar auf. (4 P)
- Bestimmen Sie das Restnetzwerk, das sich ergibt, wenn man den anfänglichen Null-Fluss f um den Sperrfluss g erhöht. (2 P)
- Bestimmen Sie den maximalen Fluss. Sie brauchen nur das Endergebnis zu zeichnen. (1 P)
- Bestimmen Sie den minimalen s - t -Schnitt. (1 P)

9 Alle Blätter aufzählen

(6 P)

Zum Speichern von Zahlen verwenden wir vollständige binäre Bäume der Höhe h , also mit $n := 2^h$ Blättern und Wurzel r . Jeder Knoten v hat drei Zeiger: $lc(v)$ zum linken Kind, $rc(v)$ zum rechten Kind und $p(v)$ zu seinem Vater, wobei $p(r) = \text{null}$. Mit Hilfe der beiden angegebenen Algorithmen möchten wir nun alle Blätter des Baumes ausgeben. Wir rufen dazu initial `FirstLeaf` auf und dann für jedes weitere Blatt `NextLeaf`.

Zeigen Sie: Der `FirstLeaf`-Aufruf hat amortisierten Aufwand von $\mathcal{O}(\log n)$ und jeder weitere `NextLeaf`-Aufruf hat amortisierten Aufwand von $\mathcal{O}(1)$.

Algorithmus FirstLeaf

Ausgabe: Blatt ganz links im Baum

```
1   $v = r$ 
2  while ( NotLeaf( $v$ ) )
3       $v = lc(v)$ 
4  return  $v$ 
```

Algorithmus NextLeaf(v)

Eingabe: ein Blattknoten v

Ausgabe: Nächstes Blatt rechts von v

```
1  while (  $v == rc(p(v))$  )
2       $v = p(v)$ 
3  if (  $v == r$  )
4      return null
5   $v = rc(p(v))$ 
6  while ( NotLeaf( $v$ ) )
7       $v = lc(v)$ 
8  return  $v$ 
```

10 FastBin

2 / (12 P)

Die Datenstruktur *FastBin* verwaltet n Elemente (die bezüglich einer Ordnungsrelation vergleichbar sind). Intern speichert eine *FastBin* zwei Listen:

O : die *ordentliche Liste* der Länge $k \leq n$, die immer sortiert ist

U : die *unordentliche Liste*, die nicht sortiert sein muss und deren Länge maximal \sqrt{k} ist

Sie können annehmen, dass das Anhängen eines Elementes an das Ende einer Liste in konstanter Zeit erfolgt.

Eine *FastBin* unterstützt die folgenden Operationen:

$LOOKUP(v)$ fragt ab, ob v in einer *FastBin* enthalten ist:

verwende binäre Suche, um v in O zu finden; wenn v dort nicht gefunden wurde, laufe durch alle Einträge von U , um v dort zu finden.

$INSERT(v)$ fügt ein neues Element v in eine *FastBin* ein:

hänge v zunächst an das Ende von U an; falls die Länge von U danach größer als \sqrt{k} ist, rufe *CLEANUP* auf.

$CLEANUP$ ist eine interne Verwaltungsroutine:

sortiere U ; verschmelze U und O zu einer sortierten Liste; mache das Ergebnis der Verschmelzung zur neuen O und leere danach U .

1) Leiten Sie die **Worst-Case**-Laufzeitkomplexität von $LOOKUP$ her.

$\Theta(\log k)$ 1 / (2 P)

binäre Suche von v in $O \in \Theta(\log k)$ (worst case)

Suche im unordentlichen Liste in $\Theta(\sqrt{k})$

$$\cancel{\Theta(k)} + \Theta(\sqrt{k}) = \Theta(\sqrt{k}) \quad \Theta(\log k)$$

2) Leiten Sie die **Worst-Case**-Laufzeitkomplexität von $INSERT$ her.

1 / (3 P)

$$\text{CLEANUP} = \underbrace{\Theta((\sqrt{k}+1) \cdot \log(\sqrt{k}+1))}_{\sqrt{k}+1 \text{ Elemente zu sortieren}} + \underbrace{\Theta(\log(\sqrt{k}+1+k))}_{\text{um } O \text{ und } U \text{ zu verschmelzen}}$$

$$\text{CLEANUP} \in \Theta((\sqrt{k}+1+k) \cdot (\log(\sqrt{k}+1+k)))$$

$$\text{INSERT} \in \Theta((\sqrt{k}+1+k) \cdot (\log(\sqrt{k}+1+k)))$$

$$\sqrt{k} \log \sqrt{k} \in \Theta(\sqrt{k}) \dots \text{aber Menge} \text{ sortet } \Theta(n)!$$

-
- 3) Zeigen Sie, dass die **amortisierte** Laufzeitkomplexität von *INSERT* in $\mathcal{O}(\sqrt{n})$ liegt. (7 P)