TECHNISCHE UNIVERSITÄT BERLIN

Fakultät IV – Elektrotechnik und Informatik
Fachgebiet Neurotechnologie (MAR 4-3)
Prof. Dr. Benjamin Blankertz
Röhr / Stahl

Algorithmen und Datenstrukturen, SoSe 18
Klausur am 04.10.2018

Bitte füllen Sie alle folgenden Felder aus:

Klausur-ID:	**666B**
tubIT-login:	
Vorname:	
Nachname:	
Matrikelnummer:	
Studiengang:	
Hochschule:	

Durch meine Unterschrift bestätige ich die Korrektheit obiger Angaben.

Ort, Datum — Unterschrift

Beachten Sie die folgenden Hinweise!

- Die Klausuren sind nummeriert und werden anonymisiert korrigiert. Bitte schreiben Sie Ihren Namen **nur** auf das Deckblatt.
- Insgesamt können in der Klausur **100 Punkte** erreicht werden.
- Diese Klausur besteht mit diesem Deckblatt aus den (nummerierten) Seiten **1-14**.
- Schreiben Sie **nicht** mit roter Farbe, grüner Farbe oder Bleistift. Diese Lösungen werden nicht bewertet!
- Notieren Sie Ihre Antworten nur auf dem Blatt (oder dessen Rückseite), auf dem auch die zugehörige Aufgabe steht, da die Aufgaben getrennt korrigiert werden.
- Geben Sie nur eine Lösung pro Aufgabe ab, streichen Sie alle alternativen Lösungsansätze auf Schmier-/Notitzblättern durch.
- Bitte den Barcode am Ende der Seiten nicht beschädigen oder überschreiben.

Zusatzblätter No.:	

Punktetabelle

Aufgabe	Punkte		
1			
2			
3			
4			
5			
6			
7			
8			
9			

Aufgabe 1: Multiple Choice ($7 \times 2 = 14$ Punkte)

Bitte kreuze alle wahren Aussagen an.

- Es ist bei allen Teilaufgaben mindestens eine Aussage wahr, aber auch mindestens eine Aussage falsch.
- Ein gesetztes Kreuz kann durch ein leeres ◯-Symbol links neben der Aussage aufgehoben werden.
- Es werden nur für vollständig richtige Teilaufgaben Punkte vergeben.

(a) Welche Wachstumsordnung beschreibt die Laufzeit der folgenden Methode `f()` am genausten?

```
public static int f(int N) {
    int sum=0;
    for(int i=1; i<N; i*=2){
        for(int j=1; j<N; j++)
            sum++;
    }
}
```

◯	$\boldsymbol{O}(\log(N))$
◯	$\boldsymbol{O}(N)$
◯	$\boldsymbol{O}(2N)$
◯	$\boldsymbol{O}(N \log(N))$
◯	$\boldsymbol{O}(N^2)$
◯	$\boldsymbol{O}(2^N)$

(b) Wie groß ist der Speicherbedarf für die Nutzung von Adjazenzlisten für einen Graphen $\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E})$?

◯	$\boldsymbol{O}(V)$
◯	$\boldsymbol{O}(E)$
◯	$\boldsymbol{O}(V + E)$
◯	$\boldsymbol{O}(V^2)$
◯	$\boldsymbol{O}(E^2)$

(c) Welcher Algorithmus ist zum Finden kürzester Wege in azyklischen Graphen geeignet und am effizientesten? Beachten Sie dabei, dass Sie nicht über den Graphen wissen, außer dass er azyklisch ist.

◯	Dijkstra Algorithmus
◯	Bellmann-Ford Algorithmus
◯	Breitensuche
◯	Ford-Fulkerson Algorithmus
◯	Topologische Sortierung

(d) Sei Δ ein Parameter zur Kapazitätskontrolle. Welche Aussagen treffen auf den Capacity Scaling Algorithmus zu.

◯	Zur Pfadauswahl, werden die Pfade gewählt, die den Fluss maximal vergrößern.
◯	Der Algorithmus terminiert, sobald $\Delta = 1$.
◯	Der Algorithmus terminiert, wenn $\Delta = 1$ und keine Pfade mehr von der Quelle zur Senke gefunden werden.
◯	Der Algorithmus bestimmt den maximalen Fluss für Flussgraphen mit beliebigen Gewichten.
◯	Es werden vergrößernde Pfade mit einem Fluss $\geq \Delta$ ausgewählt.

(e) Vererbung in Java: Nehmen Sie an, dass die Klassen B und C von der Klasse A erben. Welche der folgenden Aussagen sind korrekt?

◯	Wenn A eine öffentliche Methode $x()$ hat, haben B und C auch eine Methode $x()$.
◯	Wenn B ein Attribut y hat, hat C auch ein Attribut y.
◯	Wenn A eine öffentliche Methode $x()$ hat, nutzt die Methode $x()$ auf B garantiert dieselbe Implementierung wie die Methode $x()$ auf C.
◯	Wenn eine Klasse D von B erbt, erbt sie gleichzeitig alle Methoden und Attribute von A.

(f) Es sei eine Hashtabelle der Größe M mit quadratischer Sondierung gegeben, in der N Schlüssel gespeichert sind. Welche Wachstumsordnung beschreibt die Laufzeit für eine Suche im schlechtesten Fall am genausten?

◯	$\boldsymbol{O}(1)$
◯	$\boldsymbol{O}(M)$
◯	$\boldsymbol{O}(N)$
◯	$\boldsymbol{O}(N \log(M))$
◯	$\boldsymbol{O}(MN)$
◯	$\boldsymbol{O}(M^2)$
◯	$\boldsymbol{O}(N^2)$

(g) Welche der folgenden Aussagen zu Greedy-Algorithmen sind korrekt?

◯	Der Kruskal Algorithmus für minimale Spannbäume ist ein Greedy-Algorithmus.
◯	Der Bellman-Ford Algorithmus für kürzeste Wege ist ein Greedy-Algorithmus.
◯	Bei einem Greedy-Algorithmus ist die in einem Schritt gewählte Teillösung immer Bestandteil der Gesamtlösung.
◯	Wenn es zu einem Problem einen Greedy-Algorithmus gibt, dann ist dies immer die effizienteste Strategie.
◯	Das 0/1-Rucksackproblem kann mit einem Greedy-Algorithmus effizient gelöst werden.

Aufgabe 2: Hashing ($2 + 4 + 3 + 3 = 12$ Punkte)

Hinweis: Lesen Sie diese Aufgabe vollständig, bevor Sie sie bearbeiten.
Benutzen Sie die Hashfunktion: $h((x, y)) = (x + 4y) \mod 7$.

(a) Berechnen und notieren Sie die fehlenden Hashwerte in der folgenden Tabelle:

Schlüssel	Hashcode	Hashwerte
A	$(1, 2)$	2
B	$(2, 2)$	3
C	$(0, 4)$	2
D	$(3, 4)$	4
E	$(2, 5)$	1
F	$(1, 3)$	6
G	$(0, 2)$	1
H	$(9, 5)$	
I	$(3, 3)$	

(b) Fügen Sie die Schlüssel A-G in alphabetischer Reihenfolge in die Hashtabelle ein. Verwenden Sie dabei zur Kollisionsauflösung lineares Sondieren mit Inkrement 2.

0	1	2	3	4	5	6

(c) Nehmen Sie an, die Hashtabelle hätte ein weiteres Feld 7. Gehen Sie davon aus, dass die Hashvalues die gleichen bleiben und Sie nur die Sondierung auf das weitere Feld anpassen müssen. Wenn Sie nun wieder die Schlüssel A-G in die Tabelle einfügen, wie viele Kollisionen können durch das weitere Feld vermieden werden?

(d) Geben Sie alle möglichen Reihenfolgen an, in der die Schlüssel nach obigen Voraussetzungen in die anfangs leere Hashtabelle eingetragen worden sein können. Gehen Sie davon aus, dass Sie mit den Buchstaben E und B anfangen.

0	1	2	3	4	5	6
G	E	C	B	A	D	F

Aufgabe 3: Laufzeit ($5 + 2 + 5 = 12$ Punkte)

Sei $\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E})$ ein gerichteter Graph, der wie in der Vorlesung mit Adjanzenlisten implementiert ist. Der Graph habe keine reflexiven Kanten (d.h. keine Kanten $v \to v$ von einem Knoten zu sich selbst).

(a) Geben Sie eine möglichst niedrige (bzw. genaue) Wachstumsordnung für folgenden Pseudocode an. Begründen Sie Ihre Herleitung der Laufzeit mit Bezug auf Zeilennummern.

```
1  // gegeben ein gerichteter Graph G = (V, E)
2  bool ← false
3  für alle u ∈ V
4    für alle v ∈ V
5      für alle w ∈ V mit w ≠ u
6        falls (u,v) ∈ E und (v,w) ∈ E und (w,u) ∈ E
7          bool ← true;
8        end
9      end
10   end
11 end
```

(b) Beschreiben Sie in einem Satz, was der Code in a) macht.

(c) Formulieren Sie einen schnelleren Algorithmus in Pseudocode und geben Sie eine möglichst genaue Abschätzung der Laufzeit an.

Aufgabe 4: Objektorientierte Programmmierung (10+3 = 13 Punkte)

(a) Es sind drei Klassen gegeben, wobei `Repeat` und `Multiple` von der Klasse `Echo` erben. In diesem Code sind vier Fehler eingebaut. Geben Sie zu jedem Fehler jeweils die Methode, die Zeile, die Art des Fehlers und eine Berichtigung an.

```
1  abstract class Echo {
2     protected int volume;
3     public Echo(int volume) {
4        this.volume = volume;
5     }
6     abstract void print(String word);
7
8     public void echo(String [] said) {
9        for (int i = 0; i <= word.length; i++) {
10          print(said[i]);
11       }
12    }
13 }
```

```
1  public class Repeat extends Echo {
2     public Repeat() {
3        this(4);
4     }
5     public void print(String word) {
6        System.out.println(word + " (" + this.volume + ") ");
7     }
8  }
```

```
1  public class Multiple extends Echo {
2     public Multiple() {
3        super(3);
4     }
5     public void print(String word) {
6        int i = 0;
7        int vol = this.volume;
8     while (i < vol)
9        System.out.print(word + " (" + vol + ") ");
10       vol = vol - 1;
11    }
12 }
```

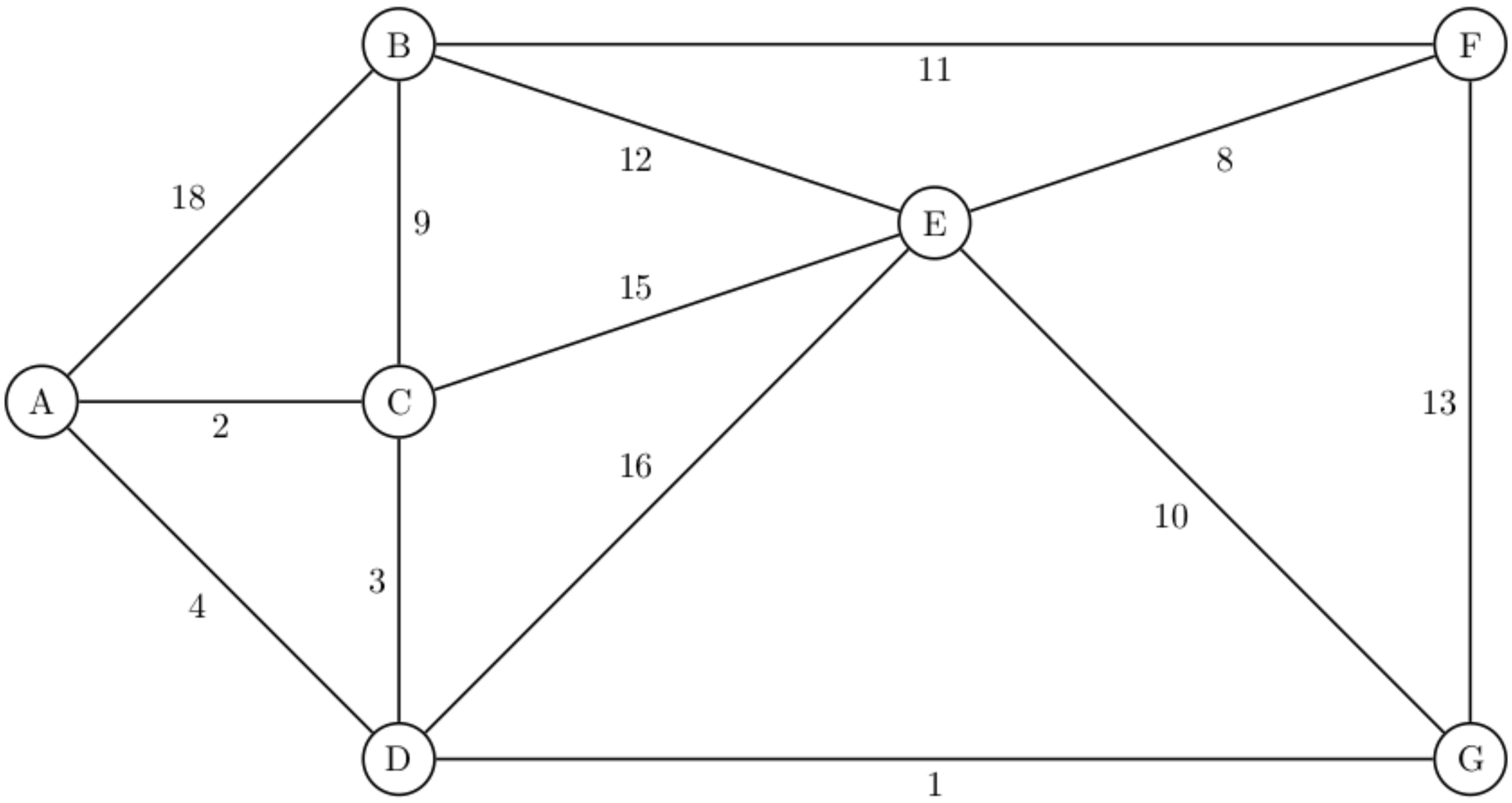

(b) Betrachten Sie nun die folgende Methode:

```
1  public class Shout {
2     public static void main(String[] args) {
3        Echo[] Shouted = new Echo[2];
4        Shouted[0] = new Repeat();
5        Shouted[1] = new Multiple();
6        for (Echo v : Shouted){
7           v.echo(args);
8        }
9     }
10 }
```

Was wird nach einer vollständigen Korrektur des obigen Codes bei der Ausführung auf der Konsole ausgegeben, wenn Sie „Viel Erfolg!“? eingeben?

Aufgabe 5: Minimum Spanning Tree ($5 + 2 + 5 =$ **12 Punkte**)

Hinweis: Der folgende Graph hat unterschiedliche ganzzahlige Gewichte zwischen 1 und 18.

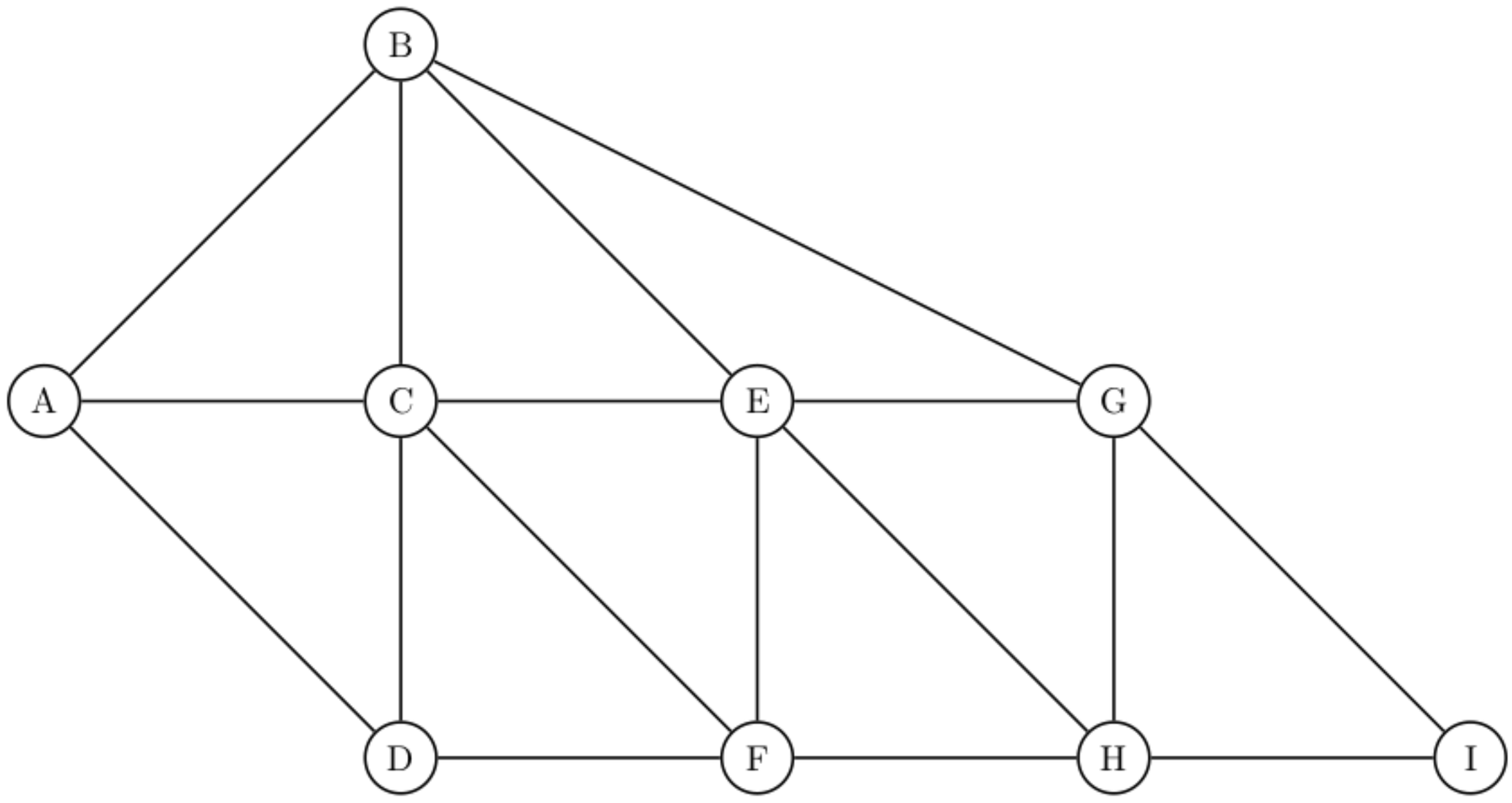


(a) Notieren Die die Gewichte der Kanten, die zum MST des oben gegebenen Graphen gehören in der Reihenfolge, in der Kruskals Algorithmus sie hinzufügen würde.

(b) Notieren sie die Gewichte der Kanten, die zum MST des Graphen aus a) gehören in der Reihenfolge, in der Prims Algorithmus sie hinzufügen würde.
Starten Sie im Knoten A.

(c) Nehmen Sie an, die Kante A-E mit dem Gewicht α wird dem Graphen hinzugefügt. Unter welcher Bedingung ist diese Kante Teil des MST? Geben Sie dabei das größtmögliche ganzzahlige Kantengewicht an.

Aufgabe 6: Breitensuche ($6 + 2 = 8$ Punkte)



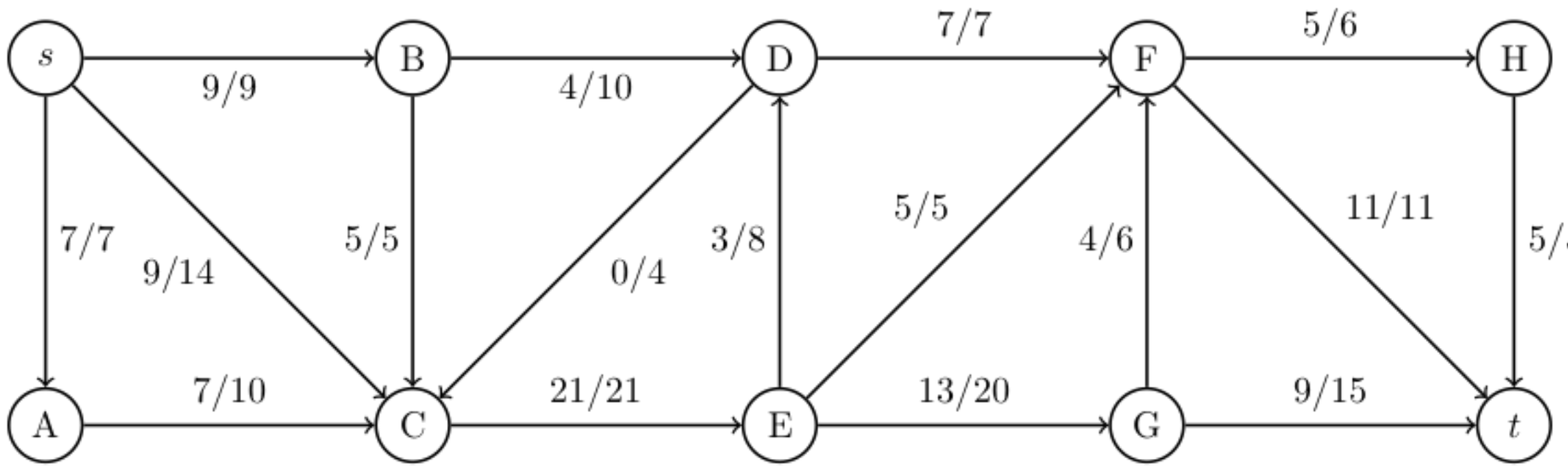
(a) Führen Sie die Breitensuche auf dem gegeben Graphen aus. Fangen Sie bei Knoten A an und notieren Sie alle Knoten in der Reihenfolge, in der sie von der Breitensuche in die Warteschlange geschrieben werden.
Gehen Sie dabei davon aus, dass in jedem Konten die benachbarten Knoten in alphabetischer Reihenfolge abgearbeitet werden. Z.B. wird die Kante F-C vom Algorithmus vor der Kante F-D bearbeitet.

(b) Wir betrachen eine Breitensuche (auf einem beliebigen ungerichteten Graphen), die in einem Knoten a beginnt. Gehen Sie davon aus, dass die Knoten x und y zu einem Zeitpunkt während dieser Breitensuche gleichzeitig in der Warteschlange stehen und kreuzen Sie die auf diese Situation zutreffenden Aussagen an.
Hinweis: Es ist mindestens eine Antwort richtig und eine Antwort falsch.

○	Die Anzahl der Kanten im kürzesten Weg zwischen a und x ist höchstens um 1 Kante größer als die Anzahl der Kanten im kürzesten Weg zwischen a und y.
○	Die Anzahl der Kanten im kürzesten Weg zwischen a und x ist höchstens um 1 Kante kleiner als die Anzahl der Kanten im kürzesten Weg zwischen a und y.
○	Die Anzahl der Kanten im kürzesten Weg zwischen a und x ist mindestens um 1 Kante größer als die Anzahl der Kanten im kürzesten Weg zwischen a und y.
○	Es gibt einen Pfad zwischen x und y.

Aufgabe 7: Edmonds-Karp Algorithmus ($1+6+2+1=10$ Punkte)

Betrachten Sie den folgenden Flussgraphen mit Quelle s und Senke t und dem eingetragenen Fluss:



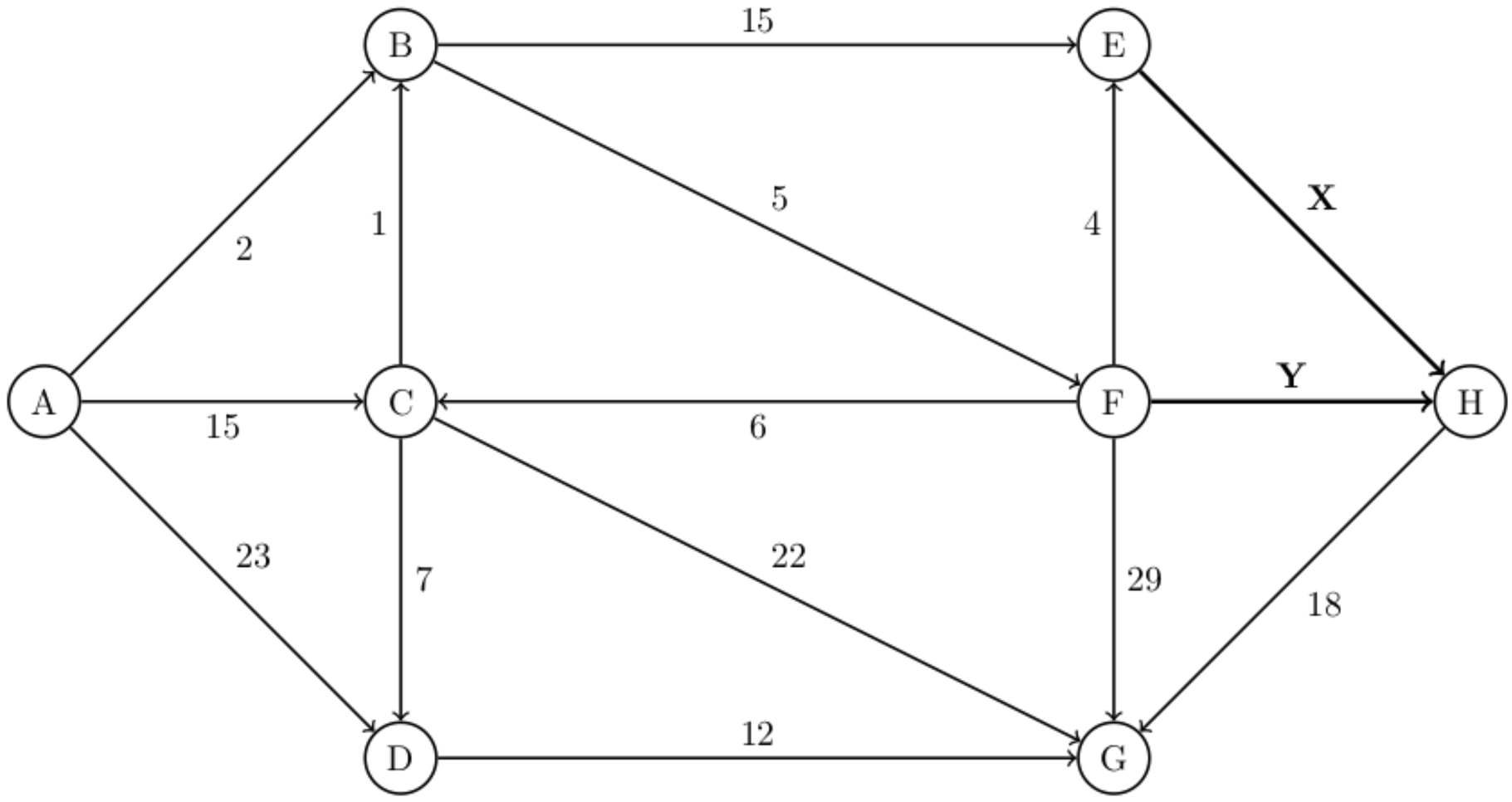
(a) Notieren Sie den Wert des Flusses im obigen Netzwerk.

(b) Führen Sie eine Iteration des Edmonds-Karp Algorithmus aus und notieren Sie die Knoten des vergrößernden Pfades beginnend in s und endend in t.

(c) Notieren Sie den Wert des maximalen Flusses im obigen Netzwerk.

(d) Notieren Sie die Kapazität des minimalen Schnittes im obigen Netzwerk.

Aufgabe 8: Dijkstra Algorithmus ($4 + 4 + 2 = 10$ Punkte)



In der folgenden Tabelle finden Sie `dist[]` und `parent[]` nachdem der Knoten E aus der Priorityqueue entfernt und and alle inzidenten Kanten relaxiert wurden (also in der 'Mitte' und *nicht* am Ende des Algorithmus).
Hinweis: In `parent[v]` finden Sie den Vorgängerknoten im Baum der aktuell kürzesten Wege.

Knoten v	`dist[v]`	`parent[v]`
A	0	`null`
B	2	A
C	13	F
D	23	A
E	11	F
F	7	B
G	36	F
H	19	E

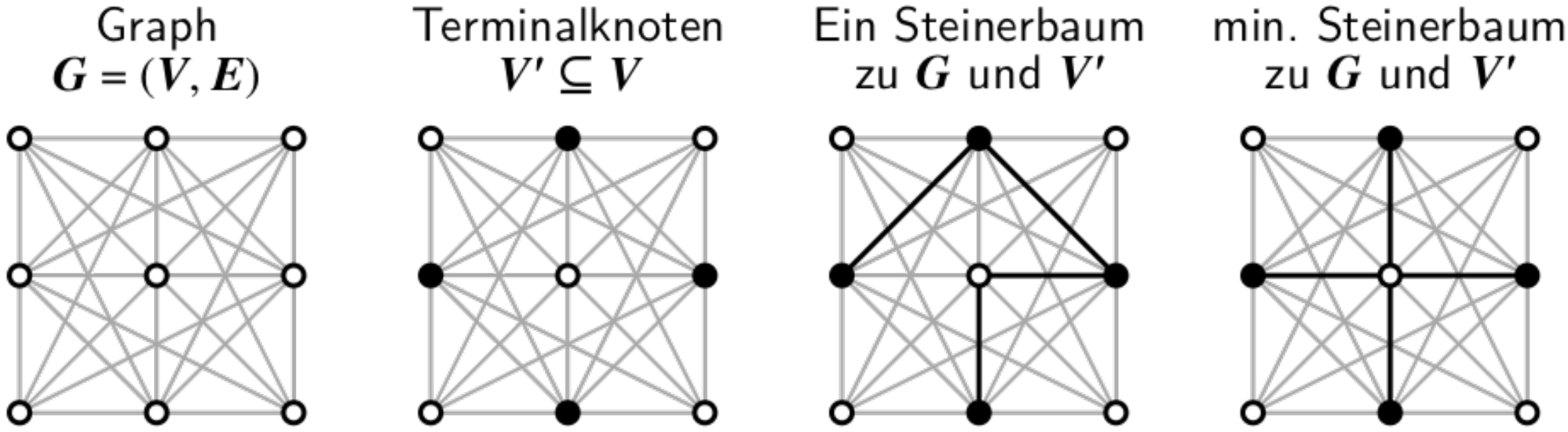
(a) Notieren Sie die ersten 4 Knoten, die aus der Priorityqueue entfernt wurden.

(b) Notieren Sie je eine Bedingung für die Gewichte der Kanten X und Y, sodass die obige Tabelle stimmt.

(c) Welcher Knoten wird nach dem Knoten E aus der Priorityqueue entfernt?

Aufgabe 9: Approximative Algorithmen ($2 + 7 = 9$ Punkte)

Es sei ein vollständiger Graph $\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E})$ mit Kostenfunktion c gegeben. Ein *Steinerbaum* zu $\boldsymbol{G}$ und einer Menge von Terminalknoten $\boldsymbol{V'} \subseteq \boldsymbol{V}$ ist ein Teilgraph $\boldsymbol{T}$ von $\boldsymbol{G}$, der ein Baum ist und alle Terminalknoten $\boldsymbol{V'}$ enthält, siehe Abbildung. Ein *minimaler Steinerbaum* zu $\boldsymbol{G}$ und $\boldsymbol{V'}$ ist ein Steinerbaum, dessen Kanten die geringste Summe von Kantengewichten unter allen Steinerbäumen besitzt. Bei der Definition ist zu beachten, dass der Steinerbaum auch nicht-terminale Knoten (d.h. Knoten aus $\boldsymbol{V} - \boldsymbol{V'}$) beinhalten kann.



Wir betrachten die Aufgabe, einen minimalen Steinerbaum zu $\boldsymbol{G}$ und $\boldsymbol{V'}$ in dem metrischen Fall zu bestimmen ('metrisch' bedeutet, dass die Kosten, bzw. Kantengewichte, die Dreiecksungleichung erfüllen: $c(u, w) \leq c(u, v) + c(v, w)$). Diese Aufgabe ist NP-vollständig.

Hinweis: Ein Graph $\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E})$ heißt *vollständig*, wenn er alle (nicht-reflexiven) Kanten enthält, also $\boldsymbol{E} = \{(v, w) \mid v, w \in \boldsymbol{V} \text{ mit } v \neq w\}$ gilt.

(a) Beschreiben Sie in einem Satz, was ein ρ-Approximationsalgorithmus zu einem Minimierungsproblem ist.

(b) Geben Sie einen effizienten Algorithmus (d.h. mit polynomieller Laufzeit) an, dessen Lösung maximal die doppelten Kosten des minimalen Steinerbaums besitzt. Belegen Sie die Approximationsgüte.

Tipp: Denken Sie an die Approximation des metrischen TSP in der Vorlesung.

Diese Seite können Sie für Notizen verwenden. Bitte nur im Ausnahmefall für Lösungen verwenden!

ID **666B**-14