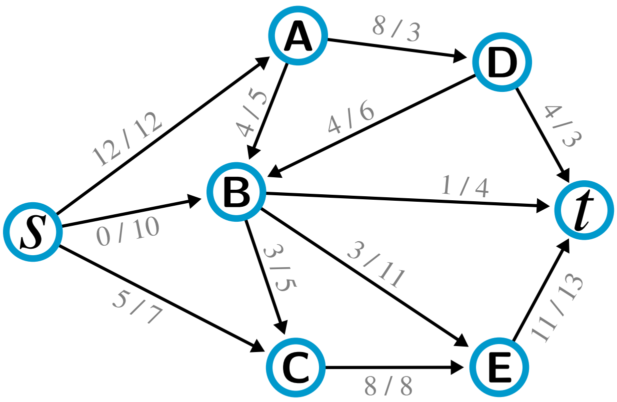
Aufgabe 1: Vermischtes ($2 + 2 + 2 + 4 + 3 + 1 = 14$ Punkte)

Geben Sie **kurze** präzise Antworten auf die Fragen.

(a) Definieren Sie, was es heißt, dass ein Graph zusammenhängend ist.

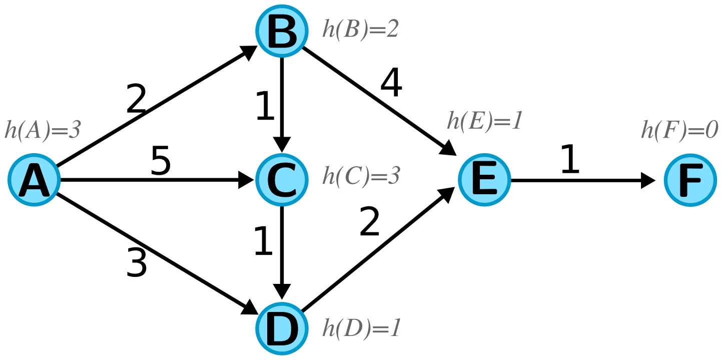
(b) Definieren Sie die Kapazität eines s von t trennenden Schnittes in einem Flussgraphen.

(c) Der 'Fluss', der in dem folgenden Flussgraphen eingezeichnet ist, enthält einen oder mehrere Verstöße gegen die Bedingungen der Flussdefinition. Listen Sie neben dem Graphen alle Verstöße auf. Geben Sie jeweils an, welche Bedingung nicht erfüllt ist.



(d) Beschreiben Sie die Breitensuche in Pseudocode. Geben Sie dabei an, welche Datenstrukturen verwendet werden.
Hinweis: Der Code braucht nur die von einem Startknoten s erreichbaren Knoten zu bestimmen.

(e) In dem folgenden gewichteten Graphen sind die Werte einer Heuristik h zur Suche der kürzesten Wege von **A** nach **F** angegeben. Ist die Heuristik zulässig? Ist sie konsistent? Begründen Sie.



(f) Welche Wachstumsordnung (in $\boldsymbol{O}$ Notation) beschreibt die Laufzeit der folgenden Methode `f(int N, int x)` am genausten? (ohne Begründung)

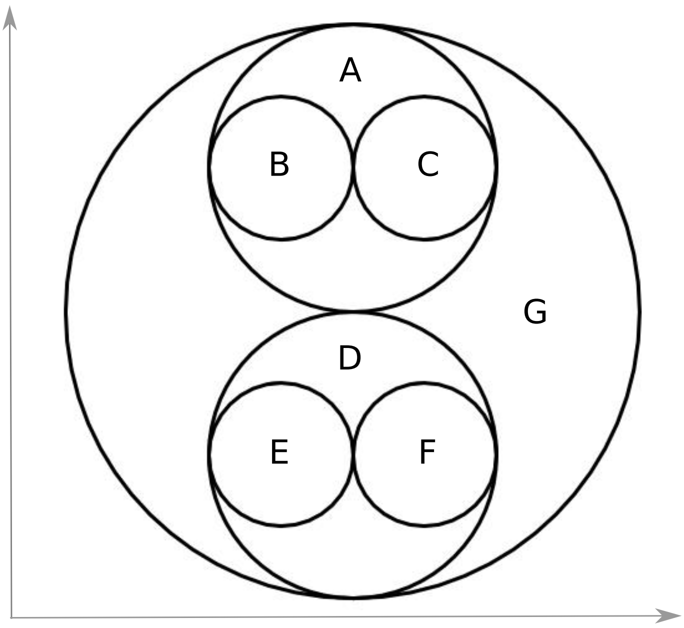
```
public static int f(int n, int x) {
    for (int i = n; i > 0; i /= 2) {
        for (int j = 0; j < i; j++) {
            x += j;
        }
    }
    return x;
}
```

Aufgabe 2: Java - Rekursion ($7 + 1 + 3 = 11$ Punkte)

Im Folgenden ist eine Klasse gegeben, welche die Kreise so zeichnet, wie sie in der Abbildung 1 zu sehen sind. Gehen Sie davon aus, dass `StdDraw.circle(x,y,r)` einen Kreis mit dem Radius r um den Punkt (x, y) zeichnet.

```java
public class Circular {
   int n;
   public Circular(int n) {
      this.n=n;
   }
   public void draw(double x, double y, double r) {
      vert(n, x, y, r);
   }
   public static void horiz(int n, double x, double y, double r) {
      if (n == 0) return;
      vert(n - 1, x + r/2, y, r/2); // recur right
      vert(n - 1, x - r/2, y, r/2); // recur left
      StdDraw.circle(x, y, r);
   }
   public static void vert(int n, double x, double y, double r) {
      if (n == 0) return;
      horiz(n - 1, x, y + r/2, r/2); // recur up
      StdDraw.circle(x, y, r);
      horiz(n - 1, x, y - r/2, r/2); // recur down
   }
   public static void main(String[] args) {
      Circular c= new Circular(3);
      c.draw(0.5, 0.5, 0.5);
   }
}
```

(a) Vervollständigen Sie die Abbildung 1, indem Sie die Buchstaben, die den Kreisen zugeordnet sind, in der Reihenfolge in die graue Tabelle eintragen, in der sie von der Klasse `Circular` gezeichnet werden.



1.	
2.	
3.	
4.	
5.	
6.	
7.	

Abbildung 1

(b) Wie viele Kreise werden gezeichnet, wenn Sie `vert(4, .5, .5, .5);` aufrufen?

(c) Angenommen es gäbe eine Klasse `Mandala`, die von der Klasse `Circular` erbt. Vervollständigen Sie die Lücken und schreiben Sie einen Konstruktor `Mandala(int n)` für diese unfertige Klasse. Die einzige Anforderung an den Konstruktor ist, dass damit die Klasse kompiliert.

```
public ______________ Mandala ______________________________{




}
```

Aufgabe 3: Hashing $(2 + 5 + 3 = 11$ Punkte)

Betrachten Sie den Ausschnitt des Codes der Klasse `Location`. Für diese Aufgabe wird die Hashfunktion

$$h(x) = abs(x) \mod 5$$

benutzt. Dabei ist $abs(x)$ der Absolutbetrag von x.

(a) Berechnen und notieren Sie die fehlenden Hashcodes und Hashadressen in der folgenden Tabelle. Die 'Namen' der Schlüssel in Spalte 1 dienen der einfacheren Notation der Schlüssel in Aufgabenteil (b).

```java
public class Location {
    public int x;
    public int y;

    // ...
    public int hashCode() {
        return 11 * x + y + 1;
    }
}
```

Name	Schlüssel	Hashcode	Hashadresse
A	$(2, 7)$	30	0
B	$(3, 5)$	39	4
C	$(-2, 6)$	-15	0
D	$(8, -1)$	88	3
E	$(6, 1)$	68	3
F	$(0, 5)$	6	1
G	$(-1, 0)$	-10	0
H	$(5, 5)$	61	1
I	$(3, 0)$	34	4
J	$(3, 4)$		
K	$(-1, 5)$		

(b) Fügen Sie die ersten neun Schlüssel der Tabelle aus (a) der Reihe nach (von oben nach unten) in die Hashtabelle ein. Tragen Sie dazu die Namen der Schlüssel (A bis I) in die Tabelle ein. Verwenden Sie Kollisionsauflösung durch Verkettung (*seperate chaining*). Die Listen verlaufen dabei horizontal von links nach rechts.

0						
1						
2						
3						
4						

(c) Was ist ein wichtiges Kriterium für eine gute Hashfunktion in Bezug auf die Laufzeit für das Suchen und Einfügen von Elementen in Hashtabellen mit Kollisionsauflösung durch Verkettung?

Vorausgesetzt die Hashfunktion erfüllt das Kriterium (und Hashcodes sind sinnvoll definiert): Geben Sie die ungefähre Anzahl der Schlüsselvergleiche für Suchen und Einfügen bei einer Hashtabelle mit M verketteten Listen und N eingefügten Elementen an.

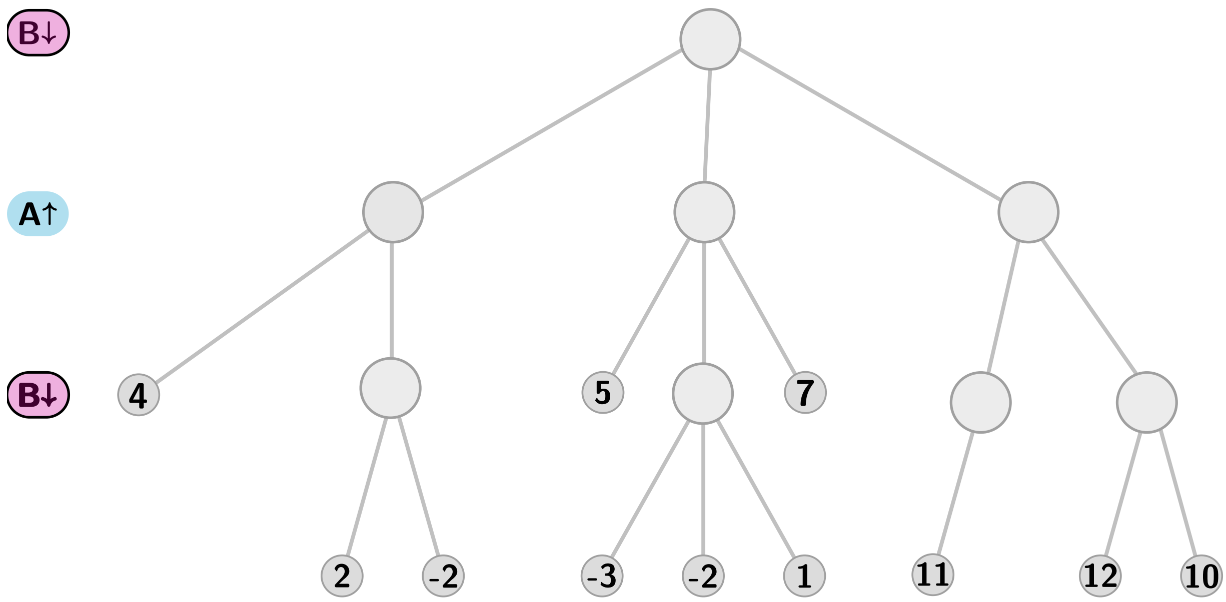
Aufgabe 4: Laufzeit ($2 + 8 = 10$ Punkte)

Gegeben sei ein gewichteter Graph $\boldsymbol{G}(\boldsymbol{V}, \boldsymbol{E})$, dessen Kantengewichte alle entweder 1 oder 2 sind. Die Aufgabe ist es, kürzeste Wege von einem Startknoten s zu allen anderen Knoten zu finden.

(a) Welche Laufzeit hätte die Verwendung von Dijkstra in diesem Fall (in der Version, die in der Vorlesung als Pseudocode besprochen wurde)? Geben Sie auch die Datenstruktur an, mit der diese Laufzeit erreicht wird.

(b) Beschreiben Sie eine Methode, um eine Laufzeit in $\boldsymbol{O}(\max(V, E))$ zu erreichen, und begründen Sie die Laufzeit.

Aufgabe 5: Minimax- und Alpha-Beta-Algorithmus (4 + 4 = 8 Punkte)

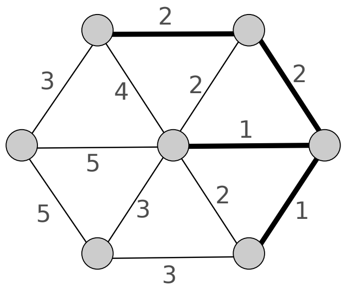


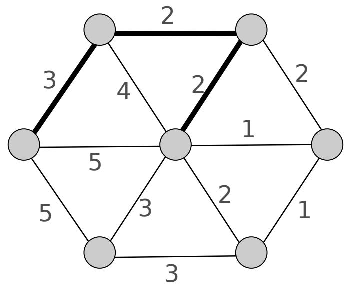
(a) Vervollständigen Sie den obigen Minimax Suchbaum.

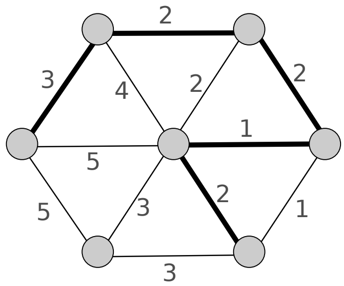
(b) Nehmen Sie an, Sie würden auf dem obigen Suchbaum eine Alpha-Beta-Suche ausführen, die von links nach rechts läuft. Welche Zweige würden nicht besucht? Tragen Sie α- und β- Cutoffs in den Baum ein. Kennzeichnen Sie, welcher Cut ein α oder ein β-Cutoff ist.

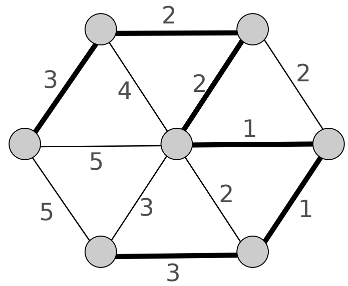
Aufgabe 6: Minimum Spanning Tree ($6 \times 2 = 12$ Punkte)

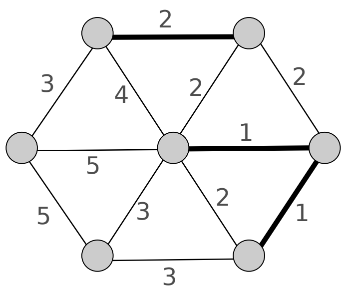
In der folgenden Abbildung sehen Sie sechs mal den gleichen Graphen. In diesen Graphen sind Kanten markiert, die durch den Prim (mit beliebigem Startknoten) oder den Kruskal Algorithmus oder durch keinen von beiden ausgewählt wurden. Dabei muss der jeweilige Algorithmus nicht bis zum Ende durchgelaufen sein, es können also auch partielle Lösungen markiert sein. Geben Sie unter jedem Graphen an, welcher der beiden Algorithmen die Kanten markiert hat, dazu schreiben Sie entweder **Kruskal** oder **Prim** unter den jeweiligen Graph in das graue Kästchen. Wenn es beide hätten sein können, schreiben Sie **beide**, wenn es keiner von beiden hätte sein können, schreiben sie **keiner** darunter.

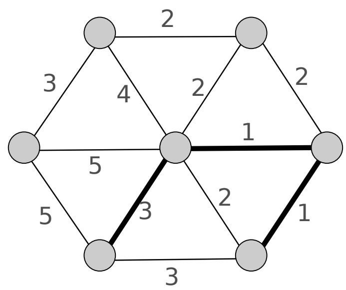






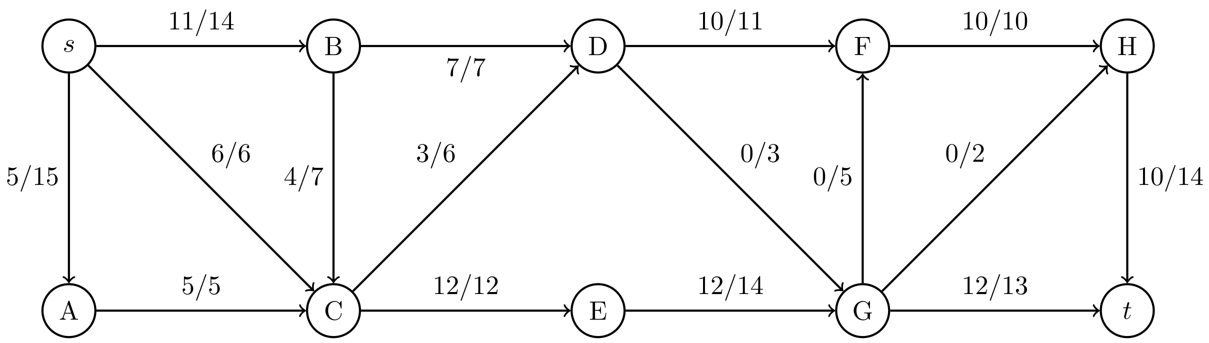






Aufgabe 7: Edmonds-Karp Algorithmus ($1+6+2+1=10$ Punkte)

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.

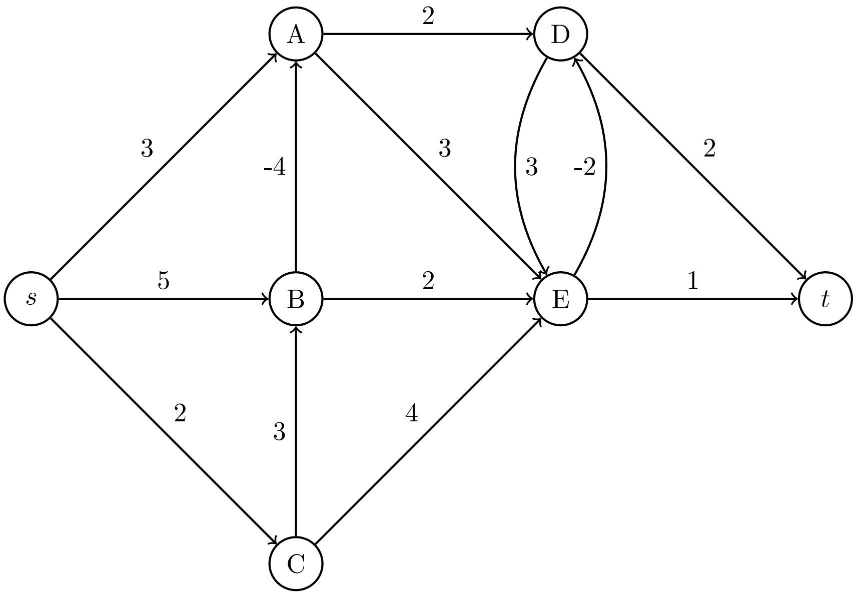
(b) Führen Sie eine Iteration des Edmonds-Karp Algorithmus aus und notieren Sie die Knoten des vergrößernden Pfades beginnend in s und endend in t.

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

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

Aufgabe 8: Kürzeste Wege ($2 + 9 + 2 = 13$ Punkte)

In dieser Aufgabe betrachten wir das Problem der kürzesten Pfade auf dem folgenden Graphen:

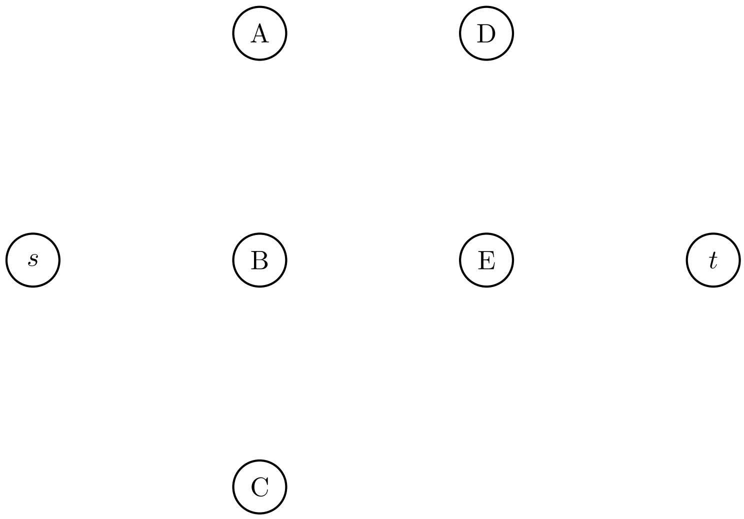


(a) Welcher Algorithmus sollte gemäß der VL zur Lösung des SSSP (Single-Source Shortest Paths) Problems für den gegebenen Graphen verwendet werden?

(b) Führen Sie den Algorithmus auf dem obigen Graphen aus. Vervollständigen Sie dabei die gegebene Tabelle so, dass Sie für jeden Knoten, dessen ausgehende Kanten relaxiert werden, eine neue Zeile nutzen und die Distanz in der Tabelle aktualisieren. Tragen Sie dabei die Knoten, von denen die Relaxierung ausgeht, in die 2. Spalte ein.
Hinweis: Es müssen nicht alle Zeilen der Tabelle genutzt werden, aber achten Sie darauf, dass Sie die Tabelle konsistent ausfüllen.

Relaxierungsschritt	ausgehender Knoten der Relaxierung	Distanz zu s						
		s	A	B	C	D	E	t
Start	-	0	∞	∞	∞	∞	∞	∞
1. Schritt								
2. Schritt								
3. Schritt								
4. Schritt								
5. Schritt								
6. Schritt								
7. Schritt								
8. Schritt								
9. Schritt								
10. Schritt								

(c) Zeichnen Sie den Baum aller kürzesten Pfade zu dem gegebenen Graphen.



Aufgabe 9: Dynamisches Programmieren ($2 + 7 + 3 = 12$ Punkte)

Es ist ein Array von ganzen Zahlen $a[]$ der Länge N gegeben und es werden die zusammenhängenden Teilfolgen (Subarrays) $a[i], \ldots, a[j]$ für alle $0 \leq i \leq j < N$ betrachtet. Mittels Dynamischer Programmierung soll die maximale Summe $a[i] + \cdots + a[j]$ aller Subarrays von a bestimmt werden. Für

$$a[] = 2, 2, -5, 4, 0, -2, 4, -7, 3$$

ist diese maximale Summe 6, und sie wird von der Teilfolge $4, 0, -2, 4$ erzielt.

(a) Ein *brute-force* Ansatz wird durch den folgenden Pseudocode beschrieben:

```
// gegeben numerisches Array a[] der Länge N
max = − inf
for all subarrays t of a
  sum(t) ←sum of subarray t
  if sum(t) > max
    max ←sum(t)
  end
end
return max   // maximale Summe aller Teilfogen von a[]
```

Welche Laufzeit hat dieser Algorithmus (minimale Größenordnung mit kurzer Begründung)?

(b) Geben Sie eine rekursive Definition der Funktion $\mathrm{OPT}(i)$ an, die die größte Summe aller zusammenhängenden Teilfolgen von `a[]` bestimmt, die mit dem Element `a[i]` enden. Diese Funktion kann als Grundlage für dynamisches Programmieren verwendet werden.

Tipp: Entweder wird eine vorherige Teilfolge durch $a[i]$ verlängert, oder $a[i]$ beginnt eine neue Teilfolge.

(c) Nehmen Sie an, dass alle Werte $OPT(i)$ für $i = 0, \ldots, N - 1$ per *bottom-up* Verfahren durch dynamische Programmierung bestimmt wurden. Beschreiben Sie, wie daraus die maximale Summe aller zusammenhängenden Teilfolgen von `a[]` bestimmt werden kann.

Geben Sie die Laufzeit des gesamten Verfahrens an.

Bemerkung: Sie können diese Teilaufgabe unabhängig von (b) lösen. Es reicht die in (b) gegebene Beschreibung.