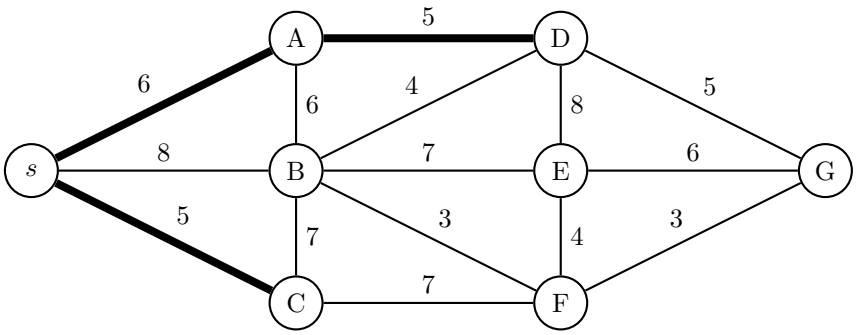
Aufgabe 1: Vermischtes ($2 + 2.5 + 2, 5 + 2 + 3 + 1 = 13$ Punkte)

(a) Der folgende Graph zeigt einen Zwischenstand eines Durchlaufes des Prim Algorithmus mit Startknoten s. Kanten mit dicken Linien sind bereits ausgewählt. Welche Kante würde der Algorithmus als nächstes auswählen? Falls es mehrere Möglichkeiten gibt, geben Sie alle Kanten an.
Hinweis: Die Notation für eine Kante ist die Kombination der beiden Knoten, zwischen denen die Kante liegt (in beliebiger Reihenfolge), z. B. (A, D).



Kante(n):

(b) Gegeben sei ein gewichteter Graph. Ist ein minimaler Spannbaum dieses Graphen notwendigerweise auch ein minimaler Spannbaum des Graphen, bei dem alle Gewichte um 10 erhöht sind? Begründen Sie kurz, oder geben Sie ein Gegenbeispiel mit maximal 4 Knoten an.

(c) Welche der folgenden Aussagen treffen im Allgemeinen auf Greedy Algorithmen zu?
Es ist mindestens eine der Antworten korrekt. Falls Sie keine Antwortmöglichkeit angeben, wird die Aufgabe als nicht bearbeitet bewertet und gibt keine Punkte.

(1) Die Suche eines Greedy Algorithmus entspricht einem Pfad von der Wurzel zu einem Blatt im Baum der Teillösungen.

(2) Greedy Algorithmen haben eine lineare Laufzeit.

(3) Mit dem Greedy Algorithmus für das teilbare Rucksack-Problem lässt sich eine Lösung für das Problem des Handlungsreisenden (*traveling salesman problem*) konstruieren.

(4) Dijkstra, Prim und Kruskal sind alle drei Greedy Algorithmen.

(5) Greedy Algorithmen können kein globales Optimum finden, da sie immer nur einer lokalen Regel folgen.

Korrekte Aussage(n):

(d) Es sei $\textsc{Opt} > 0$ der Wert einer optimalen Lösung in einem Minimierungsproblem und $\rho > 1$. Welche Ungleichungen gelten für jeden Wert V einer Lösung eines ρ-Approximationsalgorithmus für dieses Problem? Die Grenzen der Ungleichung sind so eng wie möglich zu fassen.

$\_\_\_\_ \leq V \leq \_\_\_\_$

(e) Mit Approximationsalgorithmen lassen sich die Lösungen für NP-vollständige Probleme in schnellere Laufzeit finden. Welche drei unterschiedlichen Fälle können bezüglich der Approximationsgüte auftreten? (kurze Stichpunkte)

(f) Welche Wachstumsordnung beschreibt die Laufzeit der folgenden Methode `f(int N)` am genausten? (ohne Begründung) Der Term in der ***O*** Notation soll möglichst einfach sein, also z. B. keine irrelevanten Konstanten erhalten.

```
1 public static int f(int N) {
2     int x = 0;
3     for (int i = 0; i < N; i += 2) {
4         for (int j = N; j > 0; j--) {
5             x += i * j;
6         }
7     }
8     return x;
9 }
```

Wachstumsordnung:

Aufgabe 2: Java ($4 + 2 + 2 + 2 = 10$ Punkte)

(a) Schreiben Sie eine Java Methode mit Methodenkopf, die für ein übergebenes **int** Argument `N` die Zahlen von `N-1` bis `0` in absteigender Reihenfolge zu einer `PriorityQueue` (aus den *Java Collections*) hinzufügt und diese zurückgibt.

(b) Was ist die Größenordnung der Laufzeit für einen Durchlauf der in a) geforderten Methode in Abhängigkeit von der Eingabe `N` in ***O***-Notation? Begründen Sie.

(c) Beschreiben Sie in einem Satz, was die folgende Methode `mystery` macht.

```
1  public static int mystery(Stack<Integer> stack) {
2      int a = 0;
3      Integer w = null;
4      while (!stack.isEmpty()) {
5          Integer v = stack.pop();
6          if (!v.equals(w)) {
7              a += v;
8              w = v;
9          }
10     }
11     return a;
12 }
```

(d) Die folgenden Werte werden der Reihe nach auf einen Stack (Variable `stack`) gelegt: `1, 1, 2, 3, 3, 3, 5, 3, 3, 3, 1, 2`. Welche beiden Zahlen würden dann durch den folgenden Code ausgegeben werden?

```
1 System.out.println(mystery(stack));
2 stack.add(-10);
3 System.out.println(mystery(stack));
```


Aufgabe 3: Hashing ($5 + 3 + 2 = 10$ Punkte)

(a) Bei dem "vereinfachten *Two-Sum* Problem" geht es darum zu entscheiden, ob in einem gegebenen Integer Array `num` zwei Werte (mit unterschiedlichen Indizes) existieren, die als Summe einen gegebenen Wert `sum` besitzen. Mit Hilfe der Datenstruktur `HashSet` lässt sich dies effizient implementieren. Füllen Sie die Lücken in dem folgenden Java Code so aus, dass eine korrekte und effiziente Implementierung entsteht.

```
1  public boolean twoSum(int[] num, int sum) {
2      HashSet<Integer> hash = ____________________________ ;
3      for (int i = 0; i < num.length; i++) {
4          if (hash.contains( ____________________________ )) {
5              return _________ ;
6          }
7          hash.add( ____________________ );
8      }
9      return _________ ;
10 }
```

(b) Welche Laufzeit hat die Implementation aus (a) wenn die Hashadressen gleichmäßig verteilt sind? Geben Sie eine Größenordnung in $\boldsymbol{O}$-Notation für ein Array der Länge N an und begründen Sie. Der Term in der $\boldsymbol{O}$ Notation soll möglichst einfach sein, also z. B. keine irrelevanten Konstanten erhalten.

Hinweis: Sie können von einer erfolglosen Suche ausgehen. Die Größenordnung der durchschnittlichen Laufzeit ist dieselbe bei erfolgloser und erfolgreicher Suche.

(c) Gehen Sie davon aus, dass das Array `num` unterschiedliche Werte beinhaltet und die Suche erfolglos ist, es also keine zwei Werte mit der Summe `sum` gibt. Unter welchen Bedingungen hat dieser Fall eine besonders lange Laufzeit?

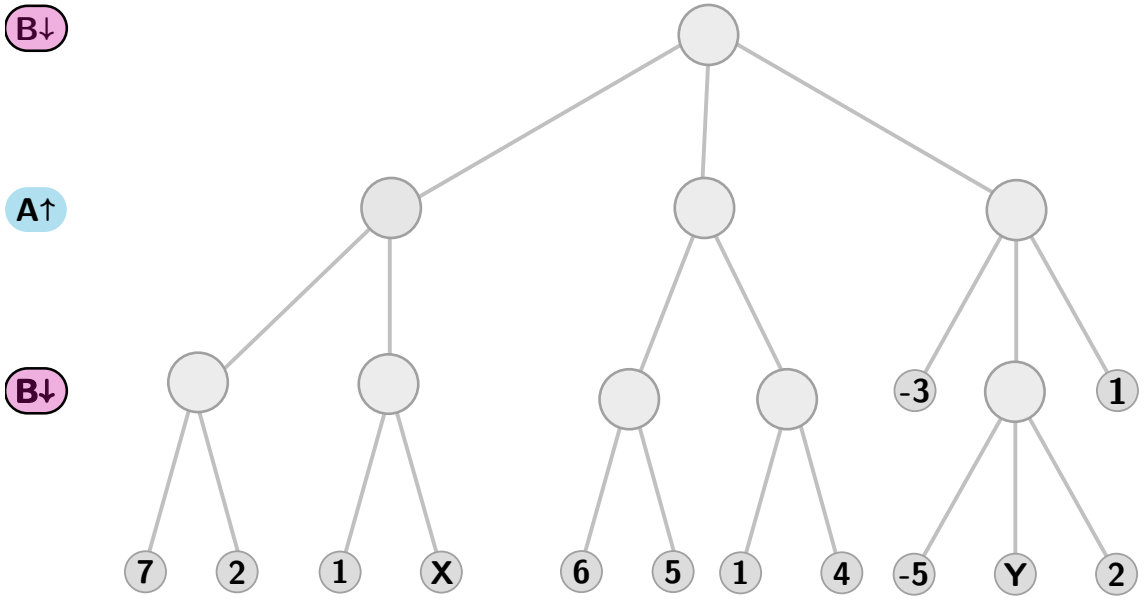
Aufgabe 4: Durchmesser eines Graphen ($2 + 6 = 8$ Punkte)

Gegeben sei ein zusammenhängender gewichteter Graph mit positiven Kantengewichten. Die Länge eines Weges ist die Summe der Kantengewichte. Der *Abstand* zwischen zwei Knoten v, w ist die Länge eines kürzestes Weges zwischen v und w. Den größten Abstand zwischen zwei Knoten in einem Graphen nennt man den *Durchmesser* des Graphen.

(a) Skizzieren Sie einen zusammenhängenden Graphen mit 5 Knoten und Durchmesser 10.

(b) Geben Sie einen Pseudocode (oder eine exakte Beschreibung eines Verfahrens) an, um den Durchmesser des gegebenen Graphen zu bestimmen. Sie dürfen dabei die aus der Vorlesung bekannten Verfahren benutzen (genauen Namen des Algorithmus angeben!). Ihre Lösung sollte eine möglichst kurze *worst case* Laufzeit (für dünne Graphen) haben. Für mangelnde Effizienz kann es Punktabzug geben.

Aufgabe 5: Minimax- und Alpha-Beta-Algorithmus ($4.5 + 2.5 + 3 = 10$ **Punkte**)



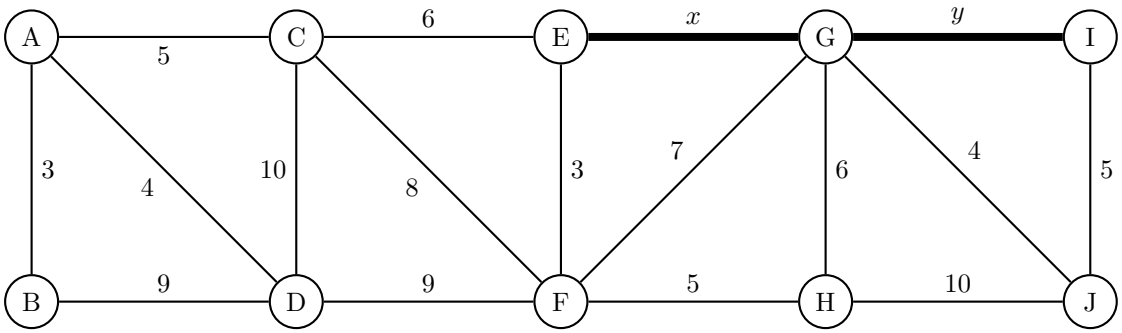
(a) Vervollständigen Sie den obigen Minimax-Suchbaum für $X = 2$ und $Y = -1$.

(b) Wie hängt der Wert des Ergebnisknoten des Minimax-Suchbaums (Knoten ganz oben) von den Knotenwerten **X** und **Y** ab? Begründen Sie.

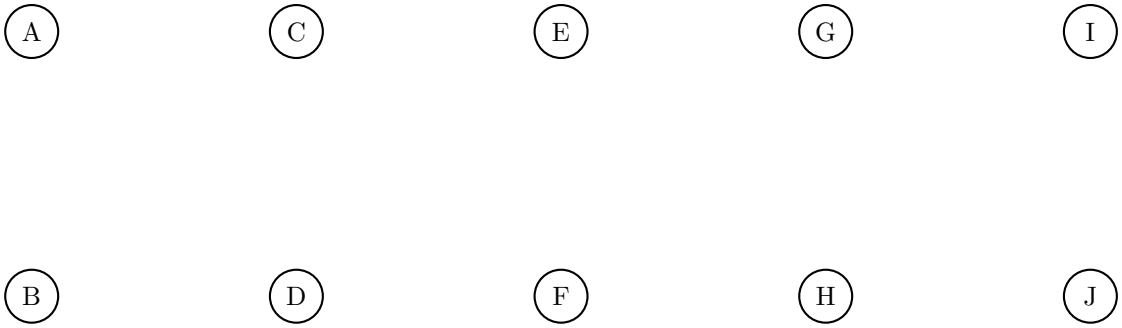
(c) Nehmen Sie an, Sie würden auf dem obigen Suchbaum eine Alpha-Beta-Suche ausführen, die von links nach rechts läuft. Welche Zweige würden nicht besucht? Tragen Sie α- und β- Cutoffs in den Baum ein.

Aufgabe 6: Minimaler Spannbaum ($7 + 4 = 11$ Punkte)

Nehmen Sie an, der *Minimum Spanning Tree* (MST) dieses Graphen enthalte die Kanten (E,G) mit dem unbekannten Gewicht x und (G,I) mit dem unbekannten Gewicht y.



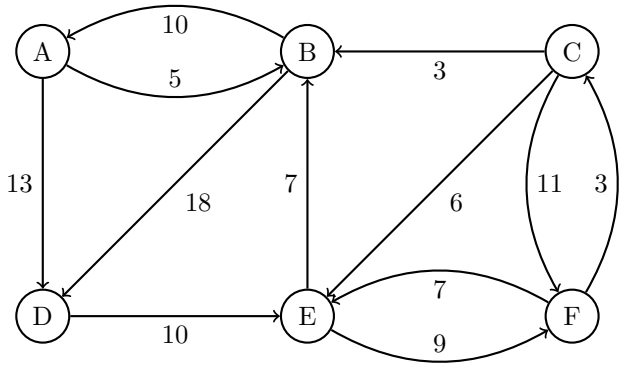
(a) Zeichnen Sie die Kanten, die unter der obigen Annahme zum MST dieses Graphen gehören müssen, ein.



(b) Geben Sie jeweils die größte obere Schranke für die Gewichte x und y der Kanten (E,G) und (G,I) an, mit der garantiert ist, dass beide Kanten tatsächlich Teil des MST sind.
Hinweis: Geben Sie individuell für jedes Gewicht x, y eine Schranke in Form einer Ungleichung an.

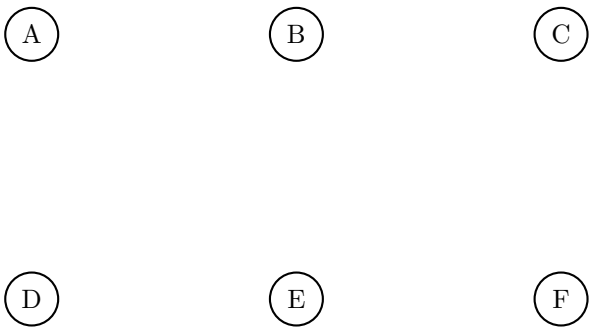
Aufgabe 7: Edmonds-Karp Algorithmus ($8 + 4 = 12$ Punkte)

Betrachten Sie den folgenden Restflussgraphen. Der Restflussgraph ist nach zwei Edmonds-Karp Iterationen eines DAG mit der Quelle **A** und der Senke **F** entstanden.
Hinweis: Alle Kanten mit dem Gewicht 0 sind in dem Restflussgraphen weggelassen.



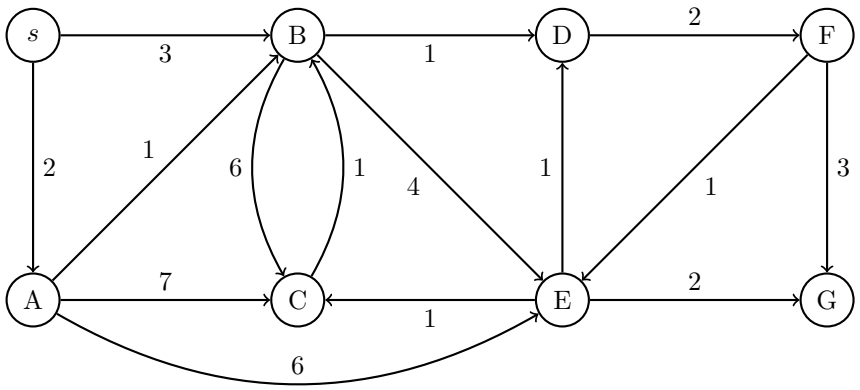
(a) Welche zwei Pfade wurden vom Edmonds-Karp Algorithmus ausgewählt? Geben Sie beide Pfade an, sowie den Fluss um den jeweils erweitert wurde.

(b) Zeichnen Sie auf Grundlage des Restflussgraphen den Flussgraphen, der nach diesen zwei Iterationen entstanden ist.



Aufgabe 8: Kürzeste Wege ($8 + 2 + 2 + 2 = 14$ Punkte)

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

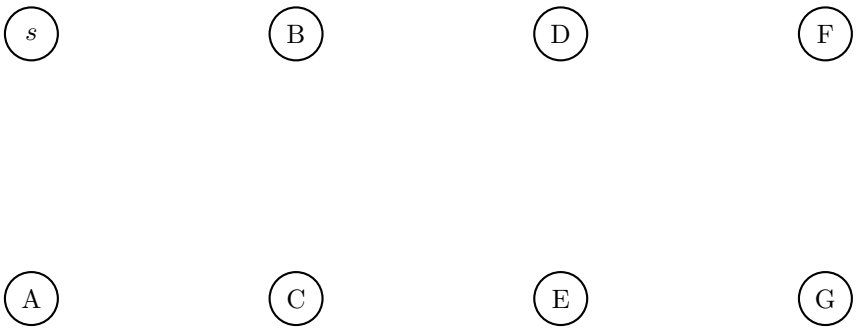


(a) Führen Sie Dijkstra ausgehend von s auf dem gegebenen Graphen aus und tragen Sie die Knoten und deren Distanz in der Reihenfolge, in der die Knoten aus der Priority Queue entfernt werden, in der Tabelle ein. Verwenden Sie die alphabetische Sortierung für die Reihenfolge der Knoten in den Adjanzenlisten und zur Auflösung von Konflikten bei gleicher Priorität.

#	0.	1.	2.	3.	4.	5.	6.	7.
Knotenname	s							
Distanz	0							

(b) An welcher Stelle hätte Bellman-Ford (in der Version mit Queue aus der Vorlesung) zum ersten Mal die Knoten in einer anderen Reihenfolge betrachtet als Dijkstra? Auch hier werden die Knoten in alphabetischer Reihenfolge entdeckt.

(c) Zeichnen Sie den Baum der kürzesten Wege (SSSP-Baum) zu dem gegebenen Graphen, der durch Dijkstra erstellt wird.



(d) Würde Bellman-Ford (in der Version mit Queue aus der Vorlesung) einen anderen SSSP-Baum für den gegebenen Graphen liefern als Dijkstra? Geben Sie ggf. die Unterschiede an.

Aufgabe 9: Dynamisches Programmieren ($6 + 6 = 12$ Punkte)

Diese Aufgabe besteht aus zwei Teilaufgaben zu zwei unterschiedlichen Anwendungen der Dynamischen Programmierung. Sie können unabhängig voneinander gelöst werden. In (a) geht es um das Aufstellen einer rekursiven Opt Funktion und in (b) um das Ablesen einer Lösung aus einer Tabelle der Dynamischen Programmierung.

(a) Für zwei Zeichenketten a und b soll die maximale Länge gemeinsamer Teilfolgen bestimmt werden. Dabei muss die Teilfolge nicht an einem Stück in den Zeichenketten vorkommen. Es ist also "GOMUS" eine Teilfolge von "ALGORITHMUS". Für

$a =$ FAHNE

$b =$ FRAGEN

sind "FAN"und "FAE" gemeinsame Teilfolgen der Länge 3. Dies ist die maximale Länge gemeinsamer Teilfolgen von a und b.

Geben Sie eine rekursive Definition der Funktion $\textsc{Opt}(i, j)$ an, die die maximale Länge aller gemeinsamen Teilfolgen der ersten i Zeichen von a und der ersten j Zeichen von b bestimmt. Diese Funktion kann als Grundlage für dynamisches Programmieren verwendet werden.

(b) Es ist ein Array von ganzen Zahlen $a[]$ der Länge N gegeben und es geht um das Problem, eine längste (streng) aufsteigende Teilfolge von a zu bestimmen. Die Teilfolge muss nicht an einem Stück in a vorkommen. Um eine Lösung mit Dynamischer Programmierung formulieren zu können, definieren wir zu einem gegeben Index $i < N$ die Menge alle Indizes j von Elementen $a[j]$, die kleiner als $a[i]$ sind durch

$$J(i) := \{j \in \mathbb{N} \mid j < i \ \ \& \ \ a[j] < a[i]\}$$

Damit lässt sich die Funktion Opt rekursiv definieren als

$$\textsc{Opt}(i) = \begin{cases} 1 & \text{falls } J(i) \text{ leer ist} \\ \max_{j \in J(i)}(\textsc{Opt}(j)) + 1 & \text{sonst} \end{cases}$$

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 eine Teilfolge von `a[]` mit maximaler Länge bestimmt werden kann.