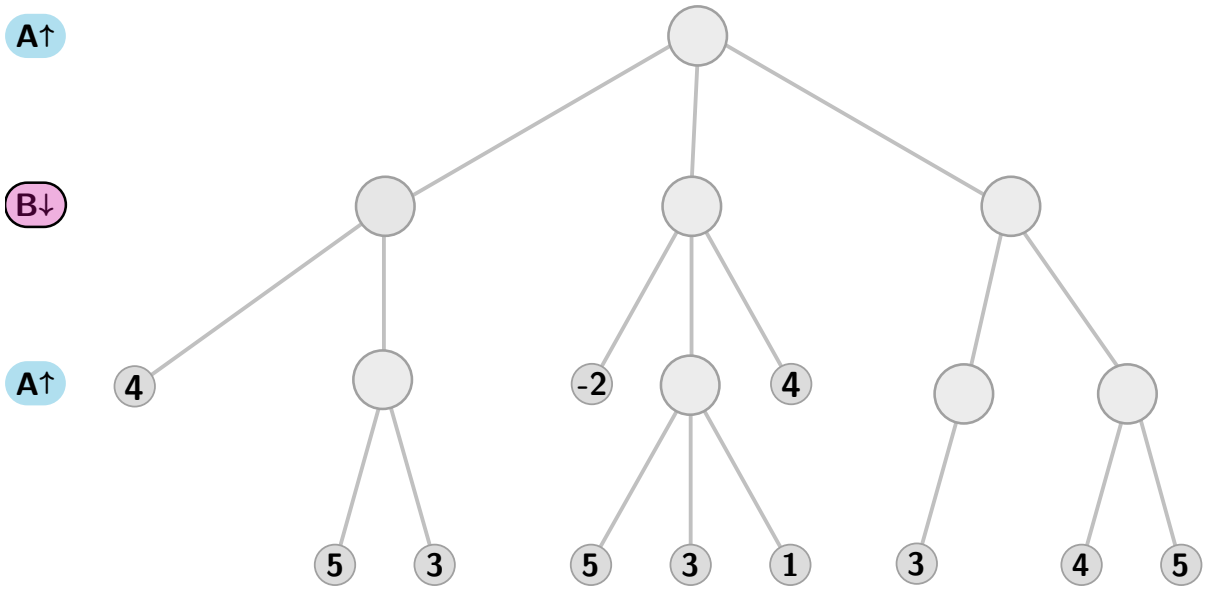
Klausurvorbereitende Aufgaben

Dies ist eine Sammlung von Aufgaben, die Sie zur Klausurvorbereitung zusätzlich zu den Aufgaben auf den Übungsblättern nutzen können. Uu diesen Aufgaben wird es Lösungen geben, die am Ende des Semesters für Sie ins git geladen werden. Sie sollten alle Aufgaben bearbeiten, ohne sich die Lösungen anzuschauen und Ihre Lösungen dann damit vergleichen. Es besteht kein Anspruch auf Vollständigkeit. Schauen Sie sich zur weiteren Ergänzung auch die Altklausuren an.
Denken Sie daran, dass es dieser Jahr erlaubt sein wird ein von Hand beidseitig beschriebenes DIN A4 Blatt (`Spickzettel`) zuverwenden. Es wurden einige Themen als Klausurthemen ausgeschlossen als Erleichterung für Sie aufgrund des Covid-Semesters: Greedy-Algorithmen, Branch-and-Bound, Flussgraphen (Woche 10), Heuristische Algorithmen und Approximative Algorithmen (Woche 11). Wir empfehlen, dass Sie sich gut überlegen, welche Inhalte Sie auswählen und wie Sie diese auf dem Zettel kurzgefasst darstellen. Die sorgfältige Erstellung ist nach unserer Erfahrung der wichtigste Aspekt eines Spickzettels.

Aufgabe 1: Minimax und Alpha-Beta



1.1 Vervollständigen Sie den obigen Minimax Suchbaum.

1.2 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 2: Dynamische Programmierung

Nehmen Sie an, es ist ein Array an Münzwerten `coins` gegeben, z. B. `[10, 20, 50]`, sowie ein Gesamtwert, z. B. 90 Cent. Nun sollen mithilfe von dynamischer Programmierung die Frage beantwortet werden, welches die minimale Anzahl von Münzen ist, mit der der Gesamtwert zusammengestellt werden kann. Dabei dürfen nur Münzen mit den angegebenen Münzwerten verwendet werden. Gehen Sie davon aus, dass der Betrag immer passend mit dem Münzen darstellbar ist.

2.1 **Vorübung** Welche beiden Voraussetzungen muss ein Problem erfüllen, damit dynamische Programmierung erfolgreich angewendet werden kann? Welche beiden Varianten werden zur Speicherung der Lösungswert verwendet?

2.2 Stellen Sie sich vor, Sie sollen die Lösung des Münzproblems für einen Betrag v angeben. Für welche Subprobleme (also Probleme mit kleineren Beträgen als v) benötigen Sie die Lösung, um daraus die Lösung für v einfach bestimmen zu können? Welche Randfälle gibt es zu beachten?

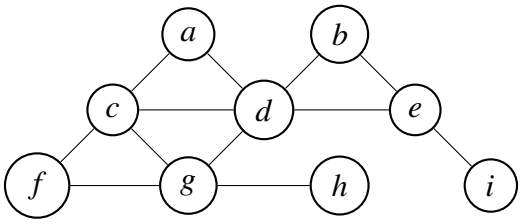
2.3 Wie oft wird die Teillösung für den Betrag 40 Cent benötigt, um die Aufgabe für den Betrag 90 Cent zu lösen?

2.4 Stellen Sie eine Opt-Funktion für das Problem auf, die die minimale Anzahl der benötigten Münzen angibt. Die Funktion sollte für ein beliebiges Array `coins` definiert werden.
Tipp: Bei der Fallunterscheidung ist es sinnvoll, für den Fall, dass der Geldwert < 0 ist, die Anzahl als *unendlich* zu definieren. Dies bedeutet, dass der Geldwert mit den vorhandenen Münzwerten nicht darstellbar ist.

Aufgabe 3: Breitensuche, Tiefensuche

3.1

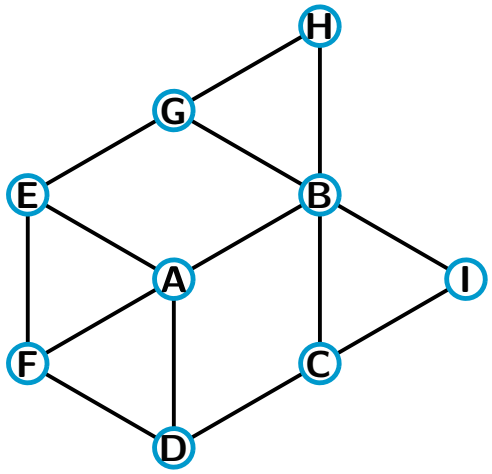
1. **Vorübung:** Wie unterscheiden sich Breitensuche (BFS) und Tiefensuche (DFS) voneinander?
2. Entscheiden Sie für folgende Applikationen, ob DFS oder BFS oder beide besser geeignet sind.
   - topologische Sortierung
   - Bestimmung von Zusammenhangskomponenten in einem ungerichteten Graphen
   - Entdeckung eines Zyklus in einem Graphen
   - Suche von Wegen, die möglichst wenige Kanten benutzen
3. Können BFS und DFS in einem Graphen gleich sein? Falls nein, warum nicht? Falls ja, geben Sie ein Beispiel.
4. Gegeben ist der folgende Graph:



Wenn BFS und DFS auf den Graphen angewendet werden, hängt die Folge, in der die Knoten besucht werden davon ab, in welcher Reihenfolge die adjazenten Knoten durchlaufen werden. Welche der Folgen 1-6 können tatsächlich bei geeigneter Reihenfolge der Adjanzenlisten in der Durchsuchung mit DFS bzw. BFS vorkommen?

1. BFS: a, c, d, g, f, b, e, h, i
2. BFS: a, d, c, f, g, e, b, h, i
3. BFS: a, d, c, e, b, g, f, i, h
4. DFS: a, c, d, g, f, h, b, e, i
5. DFS: a, c, d, b, e, i, f, g, h
6. DFS: a, c, d, g, h, f, b, e, i

3.2 Führen Sie BFS und DFS auf dem unten gegebenen Graph mit Startknoten a aus und notieren Sie dabei die Reihenfolge, in der die Knoten besucht werden. Gehen Sie dabei davon aus, dass die benachbarten Knoten in alphabetischer Reihenfolge durchlaufen werden.



	0	1	2	3	4	5	6	7	8
BFS									
DFS									

3.3 Geben Sie die worst-case Laufzeiten von BFS und DFS auf einem Graphen $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ als Wachstumordnungen an.

BFS	DFS

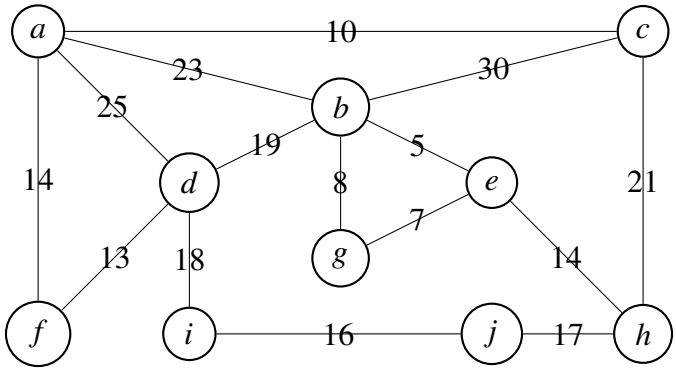
Aufgabe 4: MST

4.1 Grundlagen (Vorübung)

1. Was ist ein Spannbaum?
2. Was ist ein minimaler Spannbaum (MST)?
3. Ist der MST immer eindeutig? Falls nein, geben Sie ein Gegenbeispiel.

4.2 Schnitteigenschaft

1. **Vorübung:** Wie ist ein Schnitt durch einen Graphen definiert? Was sind kreuzende Kanten? Was ist die Schnitteigenschaft?
2. Geben Sie für den folgenden Graphen vier verschiedene Schnitte an: Einen mit 2, einen mit 3, einen mit 5 und einen mit 7 kreuzenden Kanten.



3. Wie lautet der allgemeine Ansatz um einen minimalen Spannbaum zu finden?

4.3 Algorithmen von Prim und Kruskal

1. Was ist der Hauptunterschied in der Kantenauswahl zwischen den Algorithmen von Prim und Kruskal?
2. Führen Sie die beide Algorithmen auf dem obigen Graphen aus. Schreiben Sie die Kanten in der Reihenfolge notieren, in der sie der Algorithmus auswählt.

   Prim:

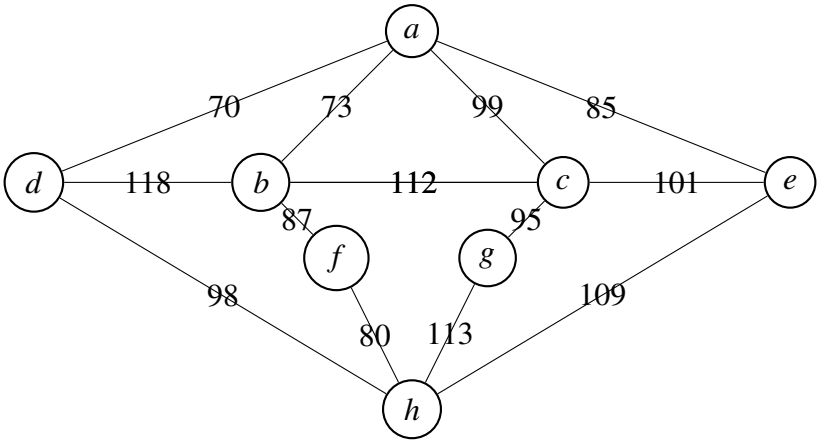
   Kruskal:

3. Das Gewicht von jeder Kante wird mit sich selbst multipliziert. Ändert sich der minimale Spannbaum? Begründen Sie Ihre Antwort.
4. Welche Datenstruktur sollte benutzt werden, um eine effiziente Implementation des Algorithmus von Prim zu erreichen?

4.4 Maximaler Spannbaum

Ein maximaler Spannbaum ist ein Spannbaum mit maximalem Gewicht.

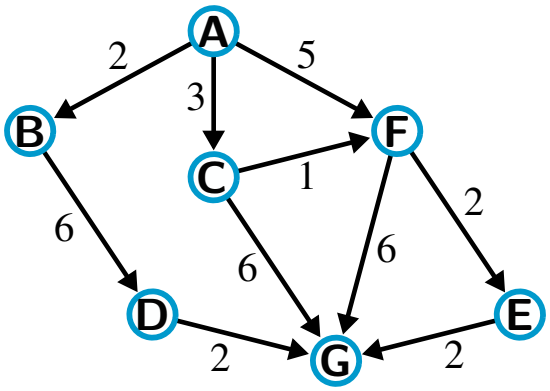
1. Wie kann der Kruskal Algorithmus geändert werden, um den maximalen Spannbaum zu finden?
2. Was ist das Gewicht des maximalen Spannbaums des folgenden Graphens?



4.5 Geben Sie die worst-case Laufzeiten der MST Algorithmen auf einem Graphen $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ als Wachstumordnungen an.

Prim	Kruskal

Aufgabe 5: SSSP



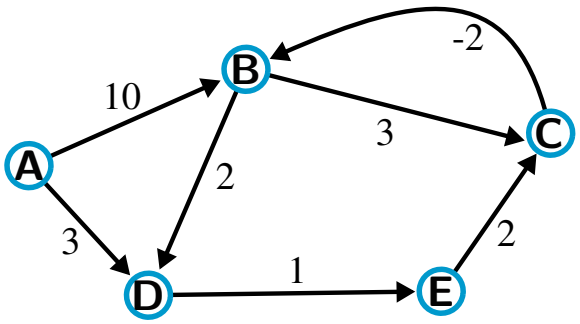
5.1 Tragen Sie in die unten stehenden Tabelle die Distanzen (`dist`) der kürzesten Wegen von Knoten **A** zu allen anderen Knoten mit Hilfe des Dijkstra Algorithmus ein. Notieren Sie auch zu jedem Knoten den Vorgänger Knoten auf einem kürzesten Weg (`parent`).

	A	**B**	**C**	**D**	**E**	**F**	**G**
`dist`							
`parent`							

5.2 Entfernen Sie eine Kante, damit die kürzeste Distanz von **A** nach **G** 9 beträgt.

5.3 Beschreiben Sie, für Graphen der Dijkstra-Algorithmus nicht so gut geeignet ist und erläutern Sie den Grund.

5.4 Führen Sie auf folgendem Graphen den Bellman-Ford Algorithmus durch und notieren Sie den Ablauf (vergleiche Screencast Video).



#	Knoten	dist	parent

5.5 Was muss für den Graph gelten, damit man topologische Sortiertung anwenden kann? Ist die topologische Sortierung (wenn sie existiert) eindeutig?