# Gedächtnisprotokoll AuD (SoSe 2023)

## Aufgabe 1: Doppelt verkettete Liste als Ring

- public static void insert_after (RNode n, RNode position) {...} implementieren
- public static node length (RNode ring) {...} implementieren, wobei ring == null eine Länge von 0 hat

## Aufgabe 2: AVL-Bäume

- T1, T2, T3, T4 gegeben: entscheiden, ob diese AVL-Bäume sind, sowie kurze Begründung
- Füge eine Zahlenfolge in einen (anfangs leeren) AVL-Baum ein und kennzeichne, falls das AVL-Kriterium verletzt wird

## Aufgabe 3: Rot-Schwarz-Bäume

- Rot-Schwarz-Baum T gegeben
  - T als 2-3-4-Baum darstellen
  - Drei Zahlen in T einfügen (immer wieder in das gleiche T, nicht aufbauend)

## Aufgabe 4: Heaps mit Suchbaum

- Kombination aus Suchbaum und Heap gegeben:
  - Knoten hat Buchstaben als Schlüssel und Zahl als Wert
  - Buchstaben im Baum wie im Suchbaum angeordnet (A < B, etc.)
  - Zahlen wie im Min-Heap angeordnet (parent <= children)
  - ohne Heapeigenschaft, dass nur unterste Ebene nicht gefüllt sein muss
  - Beim Einfügen wird zunächst gemäß des Buchstabens in den Suchbaum eingefügt und anschließend durch Rotation die Heapeigenschaft der Zahlen wieder hergestellt
- Was ist die Invariante der Rotation im Suchbaum?
- Einfügen des Knotens (D/5) in den Suchbaum

## Aufgabe 5: Suchbäume

- Löschen aus Suchbaum und Baum neu skizzieren (es waren dabei die drei Fälle: Blattknoten löschen,
  Elternknoten, das ein Kind hat, löschen und Elternknoten, das zwei Kinder hat, löschen)
- Java-Methode schreiben um den Knoten mit dem minimalen Wert zu finden

## Aufgabe 6: Heaps (MinHeap)

- Welche der Felder a bis d sind MinHeap, kennzeichne wo die Eigenschaft verletzt ist und begründe kurz
- Betrachten Sie Feld a - Wie viele Knoten haben nur ein Kindknoten?
- Wie viele solcher Knoten kann es höchstens in so einem Baum geben?
- In einem Heap mit n Einträgen, wie finde ich heraus, ob für den i-ten Knoten $a_i$ ein rechtes Kind existiert. Geben sie eine Formel (“formaler Audruck”) an.
  $hasrightchild(a_i) \Leftrightarrow 2 \cdot i + 1 \leq n$
- nennen Sie zwei Anwendungen des Heaps (= Priority Queue und Heapsort)

## Aufgabe 7: Hashing

- Hashtabelle der Größe m mit n Einträgen mit Separate Chaining und Entscheidung, ob sechs Aussagen wahr oder falsch
  1. Im schlechtesten Fall benötigt die Suche, ob ein Element in der Hashtabelle enthalten ist, O(n).
  2. Ist n > m, so hat es eine Kollision gegeben
  3. Ist n > m, so gibt es beim nächsten Einfügen eine Kollision
  4. Ist n > m, so gibt es beim nächsten Eintrag eine Kollision
  5. Existieren für einen Wert x an der Stelle h(x) in der Tabelle keine Einträge, so ist x nicht in der Tabelle enthalten.
  6. Existieren für einen Wert x an der Stelle h(x) in der Tabelle Einträge, so ist x in der Tabelle enthalten

## Aufgabe 8: Spannbäume dem Graphen zuordnen

- Graph gegeben, sowie 6 verschiedene Spannbäume
- für jeden Baum nach dessen Eigenschaft entscheiden (einfach Kreuz pro Baum, insgesamt sechs Bäume):
  - MST (minimaler Spannbaum nach Algorithmus von Prim)
  - SPT (shortest path tree nach Algorithmus von Dijkstra)
  - DFS (Spannbaum aus einer Tiefensuche ohne Gewichte)
  - BFS (Spannbaum aus einer Breitensuche ohne Gewichte)
  - oder "andere" sonst

## Aufgabe 9: Topologisches Sortieren

- zwei Graphen gegeben und entweder dessen topologisch sortierte Reihenfolge angeben oder begründen, warum dieser nicht sortiert werden kann
  - Graph 1 hatte einen Zyklus mit 5 -> 8 -> 6 -> 5 … und konnte daher nicht sortiert werden
  - Graph 2 konnte sortiert werden (Achtung, Reihenfolge dabei nicht eindeutig)
- Ergänzung des Codes, sodass dieser am Ende in System.out die topologisch sortierte Reihenfolge ausgibt

## Aufgabe 10: Gitter

- gegeben sind Abbildungen, die eventuell BFS, DFS, SPT (Dijkstra-Algorithmus) und MST (Prim-Algorithmus) entsprechen, wobei Knoten A zu Knoten B führt.
- Entscheiden Sie, für welche dieser Abbildungen (1) - (4)
  a) Dijkstra
  und
  b) A*
  gilt.

## Aufgabe 11: Dijkstra Implementierung

- unvollständige Dijkstra Implementierung gegeben und diese implementieren

## Aufgabe 12: Dynamische Programmierung (Rucksackproblem)

- vorgegeben war eine Tabelle, die dann noch für i = 7 (eine Zeile) ergänzt werden sollte, Rucksackkapazität war bei 10kg
- ergänze die Tabelle für i = 7 und gib an, wie groß der Gesamtwert maximal betragen

kann

- betrachte die Tabelle nun ohne i = 7 und gib mittels Backtracking an, welche Gewichte eingepackt werden und wie der Gesamtwert lautet

## Aufgabe 13:

wahre und falsche Aussagen

- Heapsort ist nicht stabil. //wahr
- Double Hashing maximiert die Anzahl der Kollisionen im Mittel. //falsch
- B-Bäume sind keine Binärbäume. //wahr
- Die Größe einer Hashtabelle muss eine Primzahl sein. //falsch
- Beim A* Algorithmus sind die geschätzten Kosten immer kleiner als die tatsächlichen Kosten. //falsch? habe ich schon anders gesehen
- Der Wert des maximalen Flusses in einem Graphen ist gleich den Kosten des minimalen Schnitts. //wahr
- Ein Array kann nicht zur Darstellung einer Queue verwendet werden, da lediglich der Zugriff auf das Ende in O(1) realisiert werden kann.
- Das Erstellen eines Heaps der Länge n hat im schlimmsten Fall den Aufwand O(n)
  // eventuell fehlen noch zwei Aussagen

## Zusatzaufgabe:

- AVL-Baum hat die Höhe h
- f(i) beschreibt die i-te Fibonacci-Zahl mit f(0) = 0, f(1) = 1 und f(i) = f(i-1)+f(i-2) für i > 1
- Beweise, dass der AVL-Baum mindestens immer f(h+2) - 1 Knoten besitzt