berlin

Fakultät IV

Übungstest MPGI 2
24./25. Juni 2013

Alexa
Kyprianidis / Lessig
Herholz / Hostettler / Putz

Name: .....

Vorname: .....

Matr.-Nr.: .....

Bearbeitungszeit: 45 Minuten

- Es ist ein auf beiden Seiten beschriebenes DIN-A4 Blatt als Hilfsmittel zugelassen, wenn es deinen Namen und deine Matrikelnummer enthält. Nicht zugelassen sind elektronische Hilfsmittel, wie z. B. Taschenrechner, Handys oder Laptops.
- Benutze für die Lösung der Aufgaben nur das mit diesem Deckblatt ausgeteilte Papier. **Lösungen, die auf anderem Papier geschrieben werden, können nicht gewertet werden!**
- Schreibe deine Lösungen auf das Aufgabenblatt der jeweiligen Aufgabe. Verwende auch die Rückseiten. **Schreibe keine Lösungen einer Aufgabe auf ein Blatt, das nicht zu dieser Aufgabe gehört!** Wenn du zusätzliche, von uns ausgegebene Blätter verwendest, gebe unbedingt an, zu welcher Aufgabe die Lösung gehört!
- Schreibe deutlich! Doppelte, unleserliche oder mehrdeutige Lösungen werden nicht gewertet! Streiche gegebenenfalls eine Lösung durch!
- Schreibe nur in **blau** oder **schwarz**. Lösungen, die mit Bleistift geschrieben sind, können nicht gewertet werden!
- Erscheint dir eine Aufgabe mehrdeutig, wende dich an die Betreuer.
- Du kannst die Aufgaben in einer beliebigen Reihenfolge bearbeiten.
- Gebe Nebenrechnungen an, um deinen Lösungsweg zu verdeutlichen.
- Ein Prüfungsabschnitt oder eine Prüfung kann ganz oder teilweise als nicht bestanden erklärt werden, wenn du eine Täuschungshandlung versuchst oder begehst oder durch schuldhaftes Verhalten einen ordnungsgemäßen Ablauf der Prüfung unmöglich gemacht hast.
- Trage *jetzt* (vor Beginn der Bearbeitungszeit) auf *allen* Blättern deinen Namen und deine Matrikelnummer ein.

	Punkte	Erreichte Punkte
1	12	
2	5	
3	11	
4	12	
Σ	40	

Name: .....

Matr.-Nr. .....

A1

1. Aufgabe (12 Punkte): Hashing

Wir haben zwei Hashtabellen, in welche die Schlüssel **3, 10, 18** (in dieser Reihenfolge) eingefügt wurden:

Index	0	1	2	3	4	5	6
Schlüssel				18		3	10

Hash-Tabelle A

Index	0	1	2	3	4	5	6
Schlüssel	10				3	18	

Hash-Tabelle B

Es stehen uns zwei Hashverfahren zur Verfügung. Diese sind durch die Hashfunktionen h_1 und h_2, und durch die jeweiligen Sondierungsfunktionen g_1 und g_2 definiert:

$$\begin{aligned} h_1(k) &= (k{+}2) \bmod 7 \\ g_1(k,j) &= (j + (k \bmod 5) + 1) \bmod 7 \end{aligned}$$

$$\begin{aligned} h_2(k) &= (k{+}1) \bmod 7 \\ g_2(k,j) &= (j + 3) \bmod 7 \end{aligned}$$

Hierbei ist k der einzufügende Schlüssel und j die Stelle in der Tabelle, an der gerade versucht wurde einen Wert einzufügen.

1.1. Hashverfahren (2 Punkte) Für eine Hash-Tabelle wurden h_1 und g_1, für die andere h_2 und g_2 verwendet. Welche Paare wurden zum Befüllen welcher Tabelle verwendet?

⊗ h_1 und g_1 wurden für A verwendet, h_2 und g_2 für B.
□ h_1 und g_1 wurden für B verwendet, h_2 und g_2 für A.

Richtlinien zur Bewertung:
2 Punkte für richtiges Kreuz

1.2. Kollisionen (3 Punkte) Wie viele Kollisionen sind beim Einfügen der Schlüssel mit den beiden Sondierungsfunktionen aufgetreten?

Beim Einfügen mittels h_1 und g_1 sind _____ Kollisionen aufgetreten.

Beim Einfügen mittels h_2 und g_2 sind _____ Kollisionen aufgetreten.

Lösung:
$h_1 = 2$ $h_2 = 1$

Richtlinien zur Bewertung:
1.5 Punkte pro richtiger Anzahl.

1.3. Sondierungsverfahren (3 Punkte) Wie heißen die beiden Sondierungsverfahren g_1 und g_2?

Lösung:

g_1: *Schlüsselabhängiges Sondieren (Double Hashing)*

g_2: *Lineares Sondieren*

Richtlinien zur Bewertung:

1.5 Punkte pro richtigem Namen.

1.4. Füllfaktoren (4 Punkte) Wie groß sind die Füllfaktoren der beiden angegebenen Hashtabellen A und B?

Lösung:

$A = B = 3/7 \approx 0.43$

Richtlinien zur Bewertung:

0.5 Punkte pro richtigem Füllfaktor.

Angenommen, der Füllfaktor der Tabelle A beträgt 1.

Was müssen wir tun, damit wir weitere Schlüssel in der Hash-Tabelle speichern können? (Gefragt ist nach einer kurzen Erläuterung der Schritte des Verfahrens.)

Lösung:

Rehashing:

Tabelle vergrößern

Hash-Funktion und gegebenfalls Sondierungsfunktion anpassen (z.B. Term hinter modulo erweitern)

Alle Schlüssel in der Tabelle neu einfügen bzw. neu hashen.

Richtlinien zur Bewertung:

1 Punkte für Nennung pro Bestandteil von Rehashing

Falls genannte Bestandteile weniger als 1 Punkt ergeben, gibt es noch 1 Punkt, wenn der Begriff "Rehashing" genannt wurde

Name: .....

Matr.-Nr. .....

A2

2. Aufgabe (5 Punkte): Graphen I - Scheduling

Fluglotsen weisen einfliegenden Flugzeugen ein Zeitfenster von wenigen Minuten zur Landung zu. Die Landereihenfolge der Flugzeuge hängt dabei von vielen Faktoren ab. Kriterien für eine Priorisierung sind beispielsweise wartende Anschlussflüge bei Verspätung.

Stell Dir vor, Du bist ein angehender Fluglotse und bekommst in einer Prüfung die folgenden beiden Landeanflugsituationen in den Graphen A und B vorgelegt. Die Abhängigkeiten der Flugzeuge (Knoten) untereinander sind durch die gerichteten Pfeile dargestellt.

Hinweis: Der Pfeil von $EJ \rightarrow BA$ bedeutet hier, daß Flugzeug EJ **vor** BA landen muß.

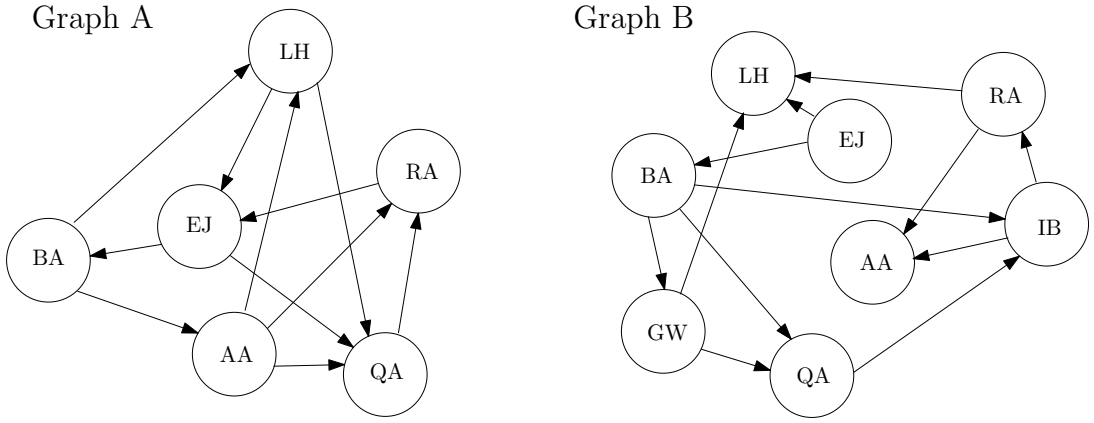


Abbildung 1: Zwei verschiedene Landeanflugssituationen

2.1. Scheduling (2 Punkte) Du musst nun in kürzester Zeit entscheiden, ob es für die beiden Situationen eine Landereihenfolge gibt. Falls ja, gib diese für den jeweiligen Graphen an. Falls nein, begründe kurz, warum es nicht möglich ist.

Gibt es eine mögliche Landereihefolge für **Graph A**?
□ ja, nämlich:
⊗ nein, da:

Gibt es eine mögliche Landereihefolge für **Graph B**?
⊗ ja, nämlich:
□ nein, da:

Lösung:

Graph A: Nein, da der Graph zyklische Abhängigkeiten enthält.
Graph B: Ja, nämlich EJ, BA, GW, QA, IB, RA, AA, LH ist eine mögliche Lösung.
Alternativ geht auch: EJ, BA, GW, QA, IB, RA, LH, AA.

Richtlinien zur Bewertung:

Es gibt insgesamt 2 Punkte für die Teilaufgabe.

jeweils 1 Punkte für die korrekte Antwort und die korrekte Reihenfolge bzw. Begründung pro Graph

2.2. Algorithmus (1 Punkt) Welcher Algorithmus kommt hier zum Einsatz?

Lösung:

Topologische Sortierung

Richtlinien zur Bewertung:

1 Punkte für die korrekte Antwort topologische Sortierung

2.3. Abhängigkeiten auflösen (2 Punkte) Sollte es keine Landereihenfolge geben, welches Flugzeug aus welchem Graphen könntest Du auf einen anderen Flughafen umleiten, um doch noch eine Landereihenfolge zu ermöglichen? Durch die Umleitung werden dieses Flugzeug und seine Abhängigkeiten aus dem Graphen entfernt.

Gäbe es noch ein anderes Flugzeug, das in Frage käme?
□ ja, nämlich
⊗ nein, da

Lösung:

Graph A: Flugzeug EJ könnte umgeleitet werden,
die Landereihenfolge ist dann: BA, AA, LH, QA, RA.
Es gibt keine andere Lösung, da sonst immer noch eine zyklische Abhängigkeit bleibt.

Richtlinien zur Bewertung:

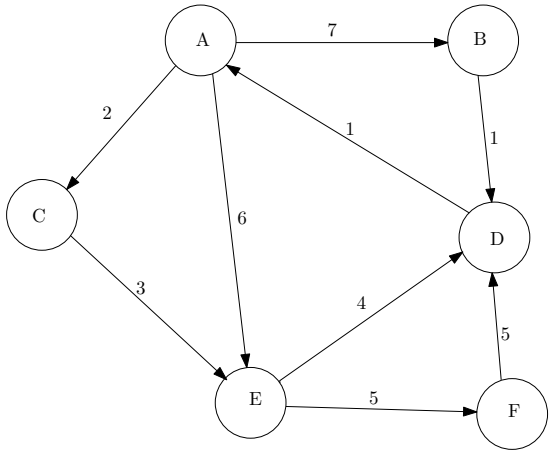
Es gibt insgesamt 2 Punkte für die Teilaufgabe.

jeweils 0.5 Punkte für korrekten Flugzeugnamen mit Graphen sowie korrekte Landereihenfolge

jeweils 0.5 Punkte für die korrekte Antwort mit Begründung.

3. Aufgabe (11 Punkte): Graphen II - Kürzeste Wege

Gegeben ist folgender gerichteter Graph. Die Gewichte auf den Kanten entsprechen den Entfernungen zwischen zwei Knoten.



3.1. Handsimulation (5 Punkte) Führe eine Handsimulation des **Dijkstra**-Algorithmus aus, um die kürzesten Wege von Knoten A zu allen anderen Knoten zu bestimmen. Nutze dazu die beiden gegebenen Tabellen.

Hinweis: A^0 bedeutet: A hat kürzeste Distanz 0 zu sich selbst.
Trage in der **zweiten** Tabelle für jeden Schritt **nur diejenigen** aktuell kürzesten Distanzen von Knoten ein, die sich **ändern**. Gib für diese Knoten auch jeweils den zugehörigen Vorgängerknoten (nach dem Komma) an.

Lösung:

Schritt	*akt. Knoten*	*Priority-Queue*
0		A^0
1	A^0	C^2, E^6, B^7
2	C^2	E^5, B^7
3	E^5	B^7, D^9, F^{10}
4	B^7	D^8, F^{10}
5	D^8	F^{10}
6	F^{10}	

	Knoten					
Schritt	*A*	*B*	*C*	*D*	*E*	*F*
0	*0, null*	∞, *null*	∞, *null*	∞, *null*	∞, *null*	∞, *null*
1		***7, A***	***2, A***		***6, A***	
2					***5, C***	
3				***9, E***		***10, E***
4				***8, B***		
5						
6						

Richtlinien zur Bewertung:

Es gibt insgesamt 5 Punkte für die Teilaufgabe.

jeweils 0.5 Punkte pro korrekte Zeile in beiden Tabellen (korrekte Änderungen in Tabelle 2)

1 Punkt für die korrekten beiden letzten Zeilen in beiden Tabellen (keine Änderungen mehr in Tabelle 2)

3.2. Kürzeste Wege Effizienz (2 Punkte) Berechnet **Dijkstra** diesen kürzesten Weg am effizientesten? Entscheide, ob die folgenden Algorithmen das Problem effizienter lösen. Begründe kurz Deine Entscheidung.

Floyd-Warshall löst das hier gegebene kürzeste Wege Problem

□ effizienter als Dijkstra, da

□ *genauso* effizient wie Dijkstra, da

⊗ *nicht* effizienter als Dijkstra, da

□ gar nicht, da

Bellmann-Ford löst das hier gegebene kürzeste Wege Problem

□ effizienter als Dijkstra, da

□ *genauso* effizient wie Dijkstra, da

⊗ *nicht* effizienter als Dijkstra, da

□ gar nicht, da

Lösung:

Dijkstra löst das gegebene Problem in $O(|V|log|V|)$ bzw. sogar in $O(|E|log|V|)$ bei Benutzung von Fibonnaci-Heaps zur Implementierung der Priority-Queue.

- *Floyd-Warshall löst das All-pairs-shortest-path problem in $O(|V|^3)$, damit aufwändiger als Dijkstra.*
- *Bellmann-Ford muss —V—-1 mal alle Kanten durchlaufen, damit ist er mit $O(|V| * |E|)$ weniger effizient als Dijkstra.*

Richtlinien zur Bewertung:

Es gibt insgesamt 2 Punkte für die Teilaufgabe.

jeweils 1 Punkte für korrekte Entscheidung und Begründung für die verschiedenen Algorithmen

3.3. Negative Kanten (2 Punkte) Wir setzen nun das Gewicht der Kante von Knoten F nach D auf den Wert -5. Warum funktioniert Dijkstra hier und auch im Allgemeinen nicht, wenn ein Graph negative Kantengewichte enthält? Begründe Deine Antwort kurz.

Lösung:

Dijkstra wählt in jedem Schritt den Knoten mit der aktuell kürzesten Distanz zum Startknoten aus, um die Distanzen zu seinen Nachbarknoten zu bestimmen (Greedy-Strategie). Die nötige Grundvoraussetzung ist, dass sich die kürzeste Entfernung zu diesem Knoten nie mehr ändert. Enthält ein Graph negative Kanten, kann ein Knoten im späteren Verlauf des Algorithmus über eine noch kürzere Entfernung erreicht werden. Damit ist aber die Grundvoraussetzung für das Funktionieren von Dijkstra verletzt.

Richtlinien zur Bewertung:

Es gibt insgesamt 2 Punkte für die Teilaufgabe.

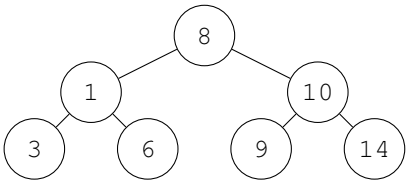
jeweils 1 Punkt für korrekte Voraussetzung mit entsprechender Begründung

Name: .....

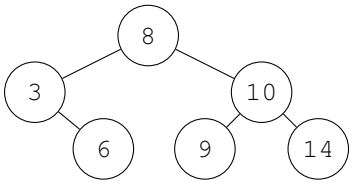
Matr.-Nr. .....

A4

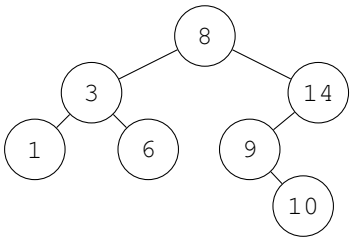
4. Aufgabe (12 Punkte): Bäume



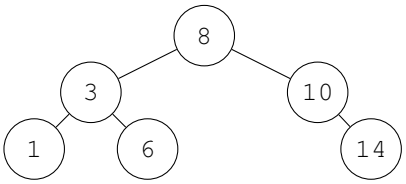
(a)



(b)



(c)



(d)

4.1. Definition binärer Bäume (4 Punkte)

Welche Bäume sind binäre Suchbäume?

□ Baum (a)
⊗ Baum (b)
⊗ Baum (c)
⊗ Baum (d)

Welche Bäume sind balanciert?

⊗ Baum (a)
⊗ Baum (b)
□ Baum (c)
⊗ Baum (d)

Welche Bäume sind linksvoll?

⊗ Baum (a)
□ Baum (b)
□ Baum (c)
□ Baum (d)

Welche Bäume sind rechtsvoll?

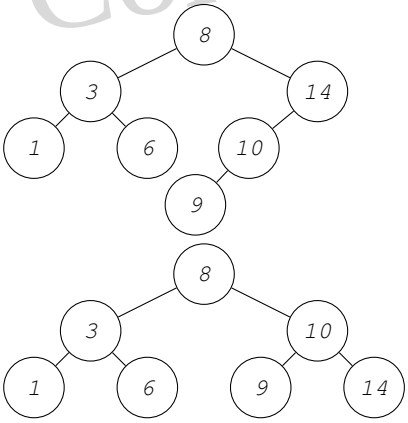
⊗ Baum (a)
⊗ Baum (b)
□ Baum (c)
□ Baum (d)

Richtlinien zur Bewertung:

1 Punkt für jedes korrektes frage, -0.5 Punkt pro Fehler. minumum ist 0 Punkte.

4.2. AVL-Bäume (2 Punkte) Angenommen, (c) ist ein AVL-Baum. Zeichne die Bäume, die als Zwischenergebnisse entstehen, wenn wir den Baum balancieren, sowie das Endresultat.

Lösung:



Richtlinien zur Bewertung:

1 Punkt fÃ¼r das richtiges balanced baeum, 1Punkt fÃ¼r das richtiges step.

4.3. Binärer Suchbaum (4 Punkte) Was ist der mittlere Zeitaufwand einer Suche in einem binären Suchbaum mit n Knoten?

□ $O(n)$
⊗ $O(log(n))$
□ $O(n\ log(n))$
□ $O(n^2)$

Richtlinien zur Bewertung:

1 Punkt für korrektes Kreuz.

Wie hoch ist der Aufwand im worst-case?

⊗ $O(n)$
□ $O(log(n))$
□ $O(n\ log(n))$
□ $O(n^2)$

Richtlinien zur Bewertung:

1 Punkt für korrektes Kreuz.

Im Folgenden ist eine Java-Implementierungen für einen binären Suchbaum angegeben, bei dem die Knoten Instanzen der Klasse `BSTNode` sind.

```
public class BinarySearchTree {

        private BSTNode root;

        /*
         * More declarations...
         */
}

class BSTNode{

    private BSTNode left;
    private BSTNode right;
    private int value;

    BinarySearchTreeNode getLeft() {
        return left;
    }
```

```
    BinarySearchTreeNode getRight() {
        return right;
    }

    int getValue() {
        return value;
    }
}
```

Implementiere die Methode `BSTNode find(int value)` der Klasse `BinarySearchTree` **ohne Rekursion zu verwenden**. Die Methode soll den Knoten mit gegebenen Wert zurückgeben oder `null`, falls ein solcher Knoten nicht existiert.

Lösung:

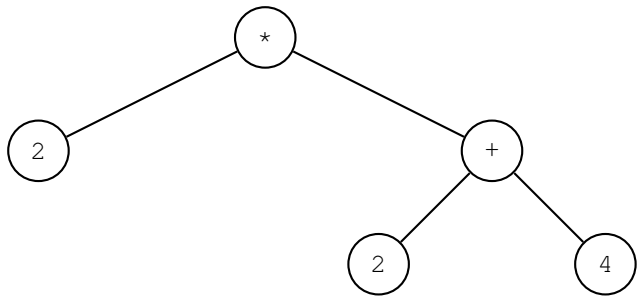
```
public BSTNode find(int value) {

        BSTNode current = root;

        while (current != null && current.getValue() != value) {
                if (value < current.getValue()) {
                        current = current.getLeft();
                } else {
                        current = current.getRight();
                }
        }

        return current;
}
```

Richtlinien zur Bewertung:

2 Punkt für korrekte Implementierung, -0.5 Punkt pro Fehler.

4.4. Expression Trees (2 Punkte) Bei der Implementierung von Compilern oder Taschenrechnern müssen algebraische Ausdrücke wie z. B. "$2 * (2 + 4)$" gespeichert und ausgewertet werden. Dafür werden sog. Ausdrucksbäume (*expression trees*) verwendet, bei denen Blätter numerische Werte und innere Knoten Operatoren repräsentieren.

Der Ausdruck "$2 * (2 + 4)$" würde beispielsweise durch folgenden Baum dargestellt:



Welcher Baumtraversierung würde die Elemente in der Reihenfolge "$2 * 2 + 4$" wiedergeben?

Lösung:

In-Order or infix traversal.

Welcher Baumtraversierung würde die Elemente in der Reihenfolge "$* \; 2 + 2 \;\; 4$" wiedergeben?

Lösung:

Pre-Order or prefix traversal.

Richtlinien zur Bewertung:

1 Punkt für jede korrekte Antwort.