Klausur MPGI 2
23.7.2008

Alexa
Hahne / Rohloff / Tetzlaff / Yilmaz

Name: .....

Vorname: .....

Matr.-Nr.: .....

Bearbeitungszeit: 90 Minuten

- Es ist eine doppelseitig beschriebene DIN-A4 Seite als Hilfsmittel zugelassen, wenn sie Ihren Namen und Ihre Matrikelnummer enthält. Nicht zugelassen sind elektronische Hilfsmittel, wie z. B. Taschenrechner, Handys oder Laptops.
- Benutzen Sie für die Lösung der Aufgaben nur das mit diesem Deckblatt ausgeteilte Papier. **Lösungen, die auf anderem Papier geschrieben werden, können nicht gewertet werden!**
- Schreiben Sie Ihre Lösungen auf das Aufgabenblatt der jeweiligen Aufgabe. Verwenden Sie auch die Rückseiten. **Schreiben Sie keine Lösungen einer Aufgabe auf ein Blatt, das nicht zu dieser Aufgabe gehört!** Wenn Sie zusätzliche, von uns ausgegebene Blätter verwenden, geben Sie unbedingt an, zu welcher Aufgabe die Lösung gehört!
- Schreiben Sie deutlich! Doppelte, unleserliche oder mehrdeutige Lösungen werden nicht gewertet! Streichen Sie gegebenenfalls eine Lösung durch!
- Schreiben Sie nur in **blau** oder **schwarz**. Lösungen, die mit Bleistift geschrieben sind, können nicht gewertet werden!
- Erscheint Ihnen eine Aufgabe mehrdeutig, wenden Sie sich an die Betreuer.
- Sie können die Aufgaben in einer beliebigen Reihenfolge bearbeiten.
- Tragen Sie jetzt (vor Beginn der Bearbeitungszeit) auf *allen* Blättern Ihren Namen und Ihre Matrikelnummer ein.

Aufgabe	Punkte	erreicht
1	10	
2	6	
3	5	
4	13	
5	10	
6	7	
7	9	

1. Aufgabe (10 Punkte): Java-Programmierung (Arrays)

Die Hotelkette Pyramid baut nur Hotels in Pyramidenform. Durch diese Form nimmt die Anzahl der Zimmer von Stockwerk zu Stockwerk um eins ab. Hat also eines dieser Hotels **zehn** Stockwerke, dann sind im **Erdgeschoss** (dem 0. Stockwerk) **zehn** Zimmer, im **1.Stock neun** usw. und im **9. Stock** nur **ein** Zimmer. Die Hotelkette möchte eine neue Zimmerverwaltungssoftware für ihre Hotels.

Die Zimmer sind in dem 2-dimensionalen Array `rooms` erfasst (1. Dimension Stockwerke, 2. Dimension Zimmer pro Stockwerk)

Für die Zimmer ist die Klasse `Hotelroom` gegeben.

```
public class Hotelroom {
    public int beds;
    public boolean free;

    public Hotelroom(int beds, boolean free){
        this.beds = beds;
        this.free = free;
    }
}
```

Das Attribut `beds` gibt an, wie viele Betten in dem Zimmer vorhanden sind. Das Attribut `free` gibt an, ob das Zimmer bereits belegt ist.

Erweitern Sie in den folgenden Aufgaben die Klasse `PyramidHotel`.

```
public class PyramidHotel {
    public Hotelroom [][] rooms;
    // constructor

    // methods
}
```

1.1. Konstruktor (7 Punkte) Schreiben Sie einen Konstruktor für die Klasse `PyramidHotel`. Achten Sie darauf, dass die Anzahl der Zimmer in jedem Stockwerk abnimmt und nicht unnötiger Speicherplatz belegt wird. Die Anzahl der Stockwerke soll als Parameter übergeben werden. Die Zimmer sollen am Anfang alle **nicht** belegt sein und haben alle **vier** Betten.

1.2. Freie Zimmer (3 Punkte) Implementieren Sie die Methode `searchRoom`, die ein freies Hotelzimmer zurückliefert, in dem es eine bestimmte Anzahl Betten gibt. Die Anzahl der freien Betten soll als Parameter übergeben werden. Wählen Sie einen geeigneten Rückgabewert für den Fall, dass kein Zimmer diese Bedingung erfüllt.

2. Aufgabe (6 Punkte): Java-Programmierung (Generics und Fehlerbehandlung)

Im folgenden soll ein generischer, gerichteter Graph implementiert werden. Der Typparameter soll dabei für die Kantengewichte stehen. Der Typ der Kantengewichte soll das Interface Comparable implementieren.

Verwenden Sie zum Speichern Ihres Graphen die Adjazenzmatrix `aM`.

2.1. Klassendefinition (2 Punkte) Ergänzt die folgende Klassendefinition für die Klasse `GenericGraph`.

```
public class GenericGraph _____________________________________________{
        EdgeWeight[][] aM;
}
```

2.2. Kante hinzufügen mit Fehlerbehandlung (4 Punkte) Fügt eurer Klasse eine Methode `addEdge` hinzu. Der Startknotenindex, der Zielknotenindex und das Kantengewicht sollen als Parameter übergeben werden. Diese Methode kann nur Kanten für existierende Knoten hinzufügen. Achten Sie darauf, dass mögliche Fehler behandelt werden.

Bei einem Fehler soll eine `GraphException` ausgelöst werden:

```
class GraphException extends Exception {
    public GraphException(String s)
    {
        super(s);
    }
}
```

3. Aufgabe (5 Punkte): Hashing

3.1. Hashtabellen (3 Punkte) Fügen Sie die drei Schlüssel 10, 18, 21 in der angegebenen Reihenfolge in eine anfangs leere Hashtabelle ein. Die folgende Hashfunktion h und Sondierungsfunktion g sind zu verwenden:

$$h(k) = k \bmod 8$$
$$g(k,j) = (j+3) \bmod 8$$

Hierbei ist k der einzufügende Schlüssel und j der zuletzt getestete Speicherplatz. Tragen Sie für jeden Schlüssel in die Tabelle ein:

- welche Positionen getestet werden,
- wo und wieviele Kollisionen auftreten,
- und wo der Schlüssel letztendlich gespeichert wird.

Schlüssel[k]	1. Versuch	2. V.	3. V.	#Kollisionen
10				
18				
21				

- Geben Sie die endgültige Hashtabelle an.

3.2. Weitere Fragen (2 Punkte) Beantworten Sie die folgenden Fragen:

- Wie nennt man das in der vorherigen Aufgabe verwendete Sondierungsverfahren?

- Wie groß ist der Füllfaktor der resultierenden Hashtabelle aus der vorherigen Aufgabe?

- Tritt in der vorherigen Aufgabe eine sekundäre Häufung auf? Wenn ja, wann?

- Nennen Sie ein Sondierungsverfahren, welches sekundäre Häufungen vermeidet.

4. Aufgabe (13 Punkte): Graphen

4.1. Darstellung eines Graphen (3 Punkte) Nachfolgend ist ein Graph in Form einer Adjazenzliste in reiner Array-Realisierung gegeben. Zeichnen Sie diesen Graphen.

`firstedge`

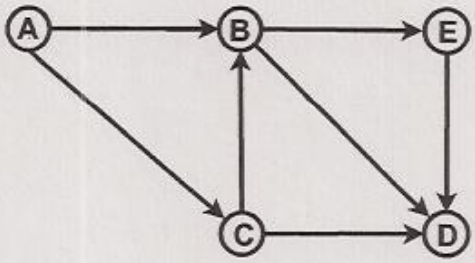
0
1
4
5
6
8

`endnode`

1
2
3
4
4
2
0
3

Zeichnung:

4.2. Topologische Sortierung (1 Punkt) Geben Sie eine topologische Sortierung der Knoten des folgenden Graphen an. Eine gerichtete Kante **A** → **B** bedeutet, dass *B* von *A* abhängig ist.

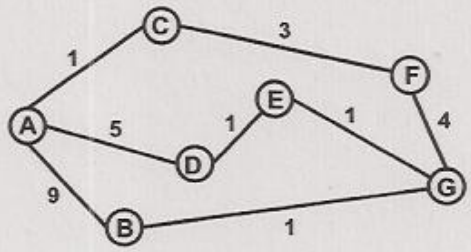


Sortierung:

4.3. Topologische Sortierbarkeit (1 Punkt) Welche **Bedingungen** muss ein Graph erfüllen, damit er topologisch sortierbar ist?

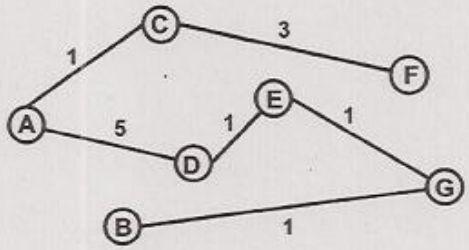
4.4. Dijkstra Algorithmus (4 Punkte) Gegeben ist nachfolgend ein ungerichteter Graph G.
Bestimmen Sie mit Hilfe des Dijkstra-Algorithmus die kürzesten Wege vom Startknoten **A** zu allen anderen Knoten. Verwenden Sie dazu die folgende Tabelle, um dort für jeden Knoten die aktuellen Kosten (d.h. die Länge des jeweils aktuell gefundenen Weges vom Startknoten **A**) anzugeben.

Hinweis: Die erste Zeile (Iteration 0) ist vorgegeben. Bei der 1. Iteration wird also der Knoten **A** betrachtet. Kann sich eine eingetragene Weg-Länge nicht mehr verbessern, brauchen Sie ab der nächsten Iteration diesen Wert in der entsprechenden Spalte nicht mehr einzutragen.



Iteration	A	B	C	D	E	F	G
0	0	∞	∞	∞	∞	∞	∞
1							
2							
3							
4							
5							
6							
7							

4.5. Weiterführende Fragen (2 Punkte) Geben Sie an, ob die Behauptungen zu der nachfolgenden Darstellung in Bezug auf den Graphen G aus der vorigen Teilaufgabe richtig oder falsch sind. Falsche Antworten führen zu Punktabzug.



- Es ist ein aufspannender Untergraph dargestellt.
  ☐ A. richtig
  ☐ B. falsch

- Es ist der kürzeste Weg vom Knoten **A** zu allen anderen Knoten dargestellt.
  ☐ A. richtig
  ☐ B. falsch

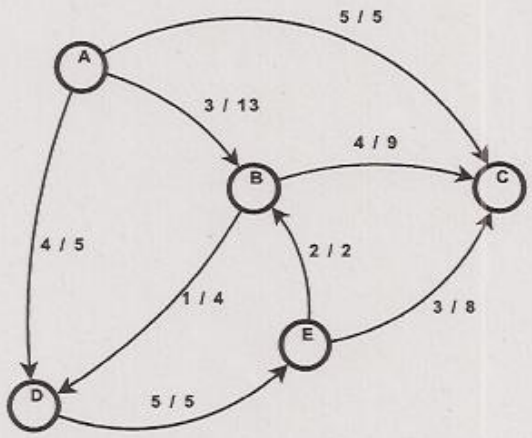
- Es ist ein Euler-Pfad dargestellt.
  ☐ A. richtig
  ☐ B. falsch

- Es ist ein minimaler Spannbaum dargestellt.
  ☐ A. richtig
  ☐ B. falsch

4.6. Aufwand von Dijsktra (2 Punkte) Schätzen Sie die ideale Laufzeit des Algorithmus von Dijkstra ab. Begründen Sie jeden Schritt Ihrer Berechnung und benennen Sie eine geeignete Datenstruktur.

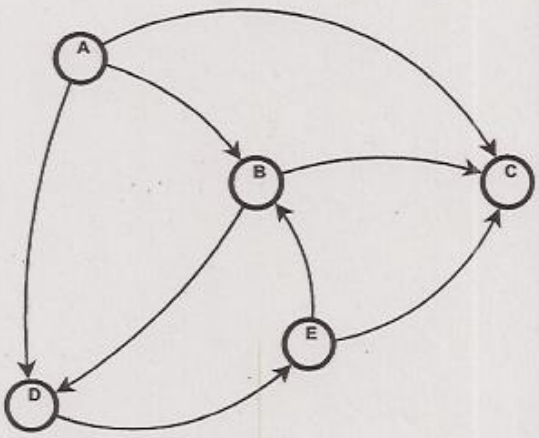
5. Aufgabe (10 Punkte): Flüsse in Netzwerken

5.1. Maximaler Fluss - Minimaler Schnitt (4 Punkte) Betrachten Sie den folgenden gültigen Flussgraphen. Der eingezeichnete Fluss ist nicht maximal.



- Ändern Sie den Fluss im Graphen so, dass der Fluss maximal ist,
- zeichnen Sie den Minimalen Schnitt ein,
- und benennen Sie die Quelle und die Senke.

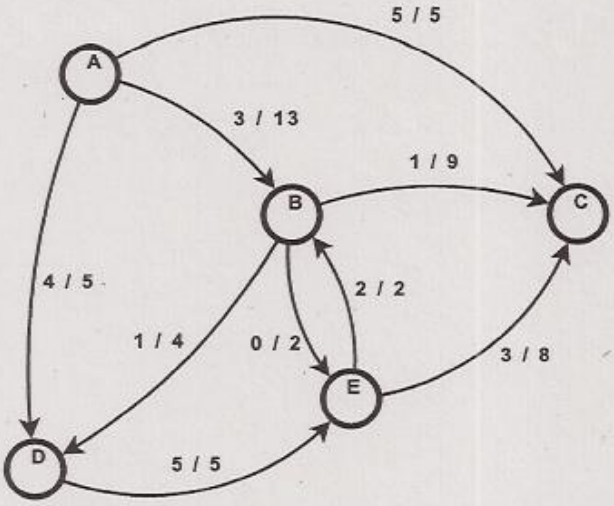
Benutzen Sie bitte die angegebene Zeichnung des Flussgraphen für Ihre Lösung:



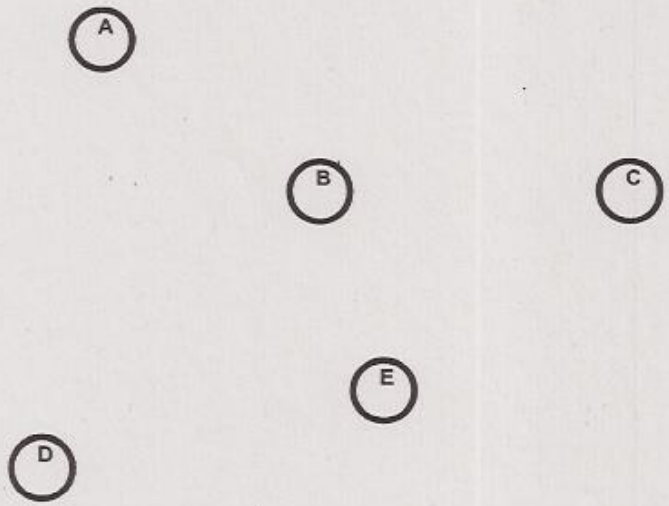
5.2. Ford-Fulkerson (2 Punkte) Beschreiben Sie den Algorithmus von Ford-Fulkerson in Worten.

Hinweis: Sie können den Begriff **Restflussgraph** verwenden ohne ihn zu erklären.

5.3. Restflussgraph (4 Punkte) Betrachten Sie den folgenden Flussgraphen.
Der eingezeichnete Fluss ist **nicht** gültig.



- Entfernen Sie **genau eine** Kante des Graphen, so dass der Fluss gültig wird
- und zeichnen Sie den zugehörigen Restflussgraphen nach dem Entfernen der Kante.



6. Aufgabe (7 Punkte): Numerik

6.1. Implementierung einer Reihenentwicklung (5 Punkte) Die trigonometrische Funktion $\cos x$ definieren wir aus numerischen Gründen folgendermaßen :

$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} \ldots + (-1)^n \frac{x^{2n}}{(2n)!} \pm \ldots = \sum_{n=0}^{\infty} (-1)^n \frac{x^{2n}}{(2n)!}$$

Implementieren Sie eine Methode `static double cosine(double x)`, welche die Kosinusfunktion anhand der oben angegebenen Formel berechnet. Der Algorithmus soll abbrechen, falls das aktuelle Ergebnis sich um weniger als 10^{-4} geändert hat.

Hinweis: Falls nötig, können die Methoden `Math.abs()` und `Math.pow()` verwendet werden. Andere Methoden aus der Klasse `Math` sind **nicht erlaubt.**

```
static double cosine(double x) {

}
```

6.2. Genauigkeit der Näherung (2 Punkte) Implementieren Sie die Methode

```
static boolean isNearMathCosine (double x, double epsilon),
```

die als Ergebnis einen booleschen Wert liefert, der angibt, ob Ihre Methode `double cosine(double x)` für das Argument `x` sich um weniger als `epsilon` von der Java-Implementierung `Math.cos(x)` unterscheidet.
Ihre Implementierung darf die Schlüsselwörter `true` und `false` **nicht** enthalten!

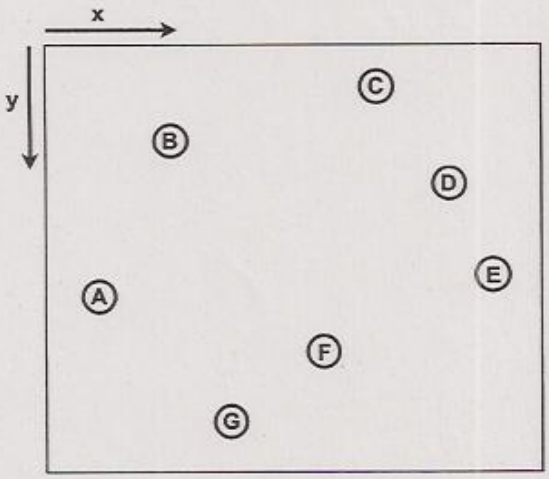
```
static boolean isNearMathCosine (double x, double epsilon) {

}
```

7. Aufgabe (9 Punkte): Bäume

7.1. kD-Bäume (3 Punkte) Nachfolgend ist die Verteilung von Punkten in einem zweidimensionalen Raum dargestellt. Diese Punkte sollen in einen kD-Baum eingefügt werden. Beachten Sie dabei, dass der Koordinatenursprung in dieser Aufgabe links oben liegt.

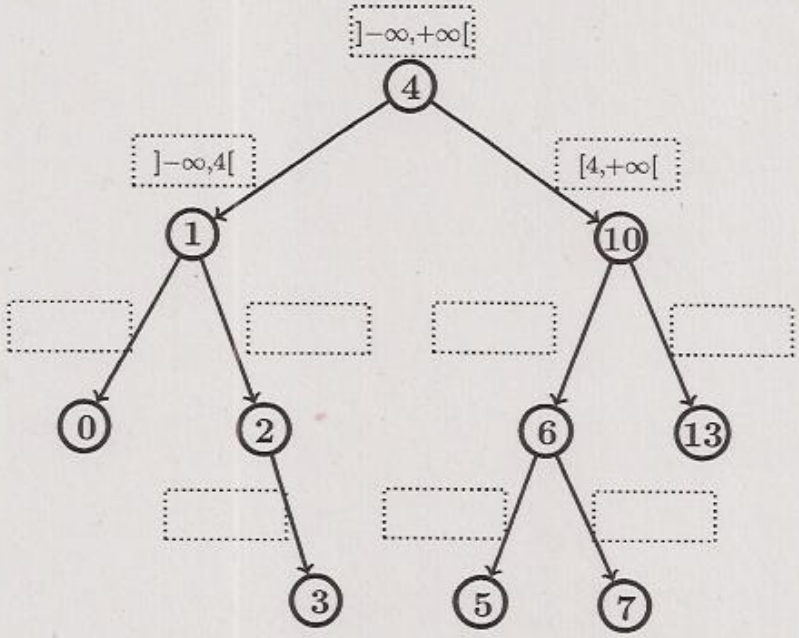
- Verwenden Sie zur Aufteilung des Raumes den Median und stellen Sie die von Ihnen gewählte Aufteilung durch Trennlinien dar. In Schicht 0 des Baumes wird nach den x-Koordinaten sortiert.
- Zeichnen Sie im nebenstehenden Feld anschließend die Sortierung der Punkte als kD-Baum.



Baum:

7.2. Suche k-nächster Nachbarn in Binärbäumen (5 Punkte) Nachfolgend ist ein binärer Suchbaum gegeben. Finden Sie in diesem Baum die **zwei** nächsten Nachbarn des Schlüssels **9**.
Verwenden Sie den (aus der Veranstaltung bekannten) algorithmischen Ansatz, der nur die Knoten betrachtet, die die aktuelle Lösung noch verbessern können.

- Geben Sie für jeden Knoten in dem gegebenen Suchbaum an, welches Suchintervall für diesen Knoten gilt. Achten Sie auf die richtige Klammerung der Intervallgrenzen. Die Regel dafür wird durch die Beispielangaben impliziert.



- Geben Sie in nachfolgender Tabelle die einzelnen Lösungsschritte bei der Anwendung des Algorithmus auf den gegebenen Suchbaum an. Geben Sie auch an, wie groß das Bewertungskriterium der aktuellen Lösung und das daraus resultierende Suchintervall ist.

aktueller Knoten	aktualisierte Lösung	aktuelles Bewertungskriterium	Suchintervall
-	(- , -)	∞	$]-\infty,+\infty[$

7.3. Algorithmenschema (1 Punkt) Um welches Algorithmenschema handelt es sich bei dem verwendeten Algorithmus zur k-nächsten Suche? Begründen Sie Ihre Antwort in einem Satz.