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.

Lösung:

```
public Hotelroom [][] rooms;
    // constructor
        public PyramidHotel (int floor) {
            this.rooms = new Hotelroom[floor][];
            for (int i = 0; i < floor; i++) {
              Hotelroom[] floorRs = new Hotelroom[floor - i];
              for (int k = 0; k < (floor - i); k++){
                  floorRs[k] = new Hotelroom(4,true);
              }
              this.rooms[i] = floorRs;
            }
        }
```

Richtlinien zur Bewertung:

- *0.5 Punkte für korrekte Deklaration*
- *3 Punkte für absteigende Arraygrösse und richtige Initialisierung*
- *je 1 Punkt für jede korrekte for-Schleife*
- *1.5 Punkte für korrekte Initialisierung der Zimmer (new, vier Betten und frei)*
- *-1 Punkt je Java-Fehler (keine Syntaxfehler)*

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.

Lösung:

```
public Hotelroom searchRoom(int numberOfBeds) {
    for (int floor = 0; floor < this.rooms.length; floor++) {
        for (int nr = 0; nr < this.rooms[floor].length; nr++) {
            Hotelroom r = this.rooms[floor][nr];
            if (r.free && r.beds == numberOfBeds) {
                return r;
            }
        }
    }
    return null;
}
```

Richtlinien zur Bewertung:

- *1 Punkt für korrekte Deklaration (die Funktion muss ein Gaestezimmer liefern und bekommt genau einen int-Parameter)*
- *1 Punkt für korrekt verschachtelte Schleifen zur Iteration über die Arrays*
- *1 Punkt für korrekte Bedingung und Rückgabe eines geeigneten Zimmers (null, genau dann, falls kein Zimmer das Kriterium erfüllt - alle anderen Rückgaben sind in diesem Fall nicht sinnvoll)*
- *-0.5 Punkte je Java-Fehler (keine Syntaxfehler)*

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;
}
```

Lösung:

```
public class GenericGraph<EdgeWeight extends Comparable<EdgeWeight>> {
        EdgeWeight[][] aM;
}
```

Richtlinien zur Bewertung:

- *1 Punkt für Comparable < EdgeWeight>*
- *1 Punkt für EdgeWeight extends Comparable*

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);
    }
}
```

Lösung:

```
        public void addEdge(int start, int end, EdgeWeight w) throws GraphException {
                if(start < 0 || end < 0 ||
                                start >= aM.length || end >= aM.length) {
                        throw new GraphException("start or end node not in Graph");
                }
                this.aM[start][end] = w;
        }
```

Richtlinien zur Bewertung:

- *0.5 Punkte für throws GraphException*
- *0.5 Punkte für kleiner* `length`
- *0.5 Punkte für größer 0*
- *0.5 Punkte für korrekte Parameterübergabe*
- *1 Punkt für korrektes Einfügen des Knotens*
- *1 Punkt für korrektes Werfen der Exception*

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:

$$\begin{aligned} h(k) &= k \bmod 8 \\ g(k,j) &= (j+3) \bmod 8 \end{aligned}$$

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.
- Geben Sie die endgültige Hashtabelle an.

Lösung:

Schlüssel[k]	*1. Versuch*	*2. V.*	*3. V.*	*#Kollisionen*
10	*2*	-	-	*0*
18	*2*	*5*	-	*1*
21	*5*	*0*	-	*1*

Einfügen des Schlüsselwerts 10: $h(10) = 2$, *keine Kollision: also Einfügen unter Position* 2.
Einfügen des Schlüsselwerts 18: $h(18) = 2$, *Kollision:* $g(18,2) = 2+3 \ mod \ 8 = 5$, *keine Kollision: also Einfügen unter Position* 5.
Einfügen des Schlüsselwerts 21: $h(21) = 5$, *Kollision:* $g(21,5) = 5+3 \ mod \ 8 = 0$, *keine Kollision: also Einfügen unter Position* 0.
Endgültige Hashtabelle: $[21, -, 10, -, -, 18, -, -]$

Richtlinien zur Bewertung:

- *Für jede richtig berechnete Position: 0.5 Punkte.*
- *Für richtige Anwendung der Funktion g und richtige Berechnung der endgültigen Position: 0.5 Punkte.*
- *Für die richtige endgültige Hashtabelle: 0.5 Punkte.*

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.

Lösung:
lineare Sondierung
3/8 oder bei fehlerhaften Ergebnissen 3/N.
Ja, die 21 trifft auf die bereits sondierte 18.
quadratisches, inkrementelles oder schlüsselbasiertes Sondieren (double hashing).

Richtlinien zur Bewertung:
0.5 Punkte für jede richtige Antwort.

4. Aufgabe (13 Punkte): Graphen

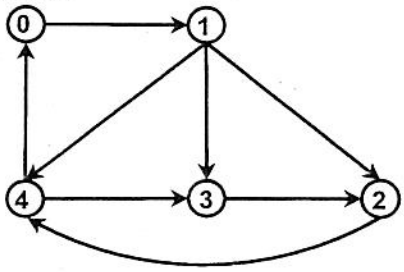
4.1. Darstellung eines Graphen (3 Punkte) Nachfolgend ist ein Graph in Form einer Adjazenzliste in reiner Array-Realisierung gegeben. Zeichnen Sie diesen Graphen.

firstedge
0
1
4
5
6
8

endnode
1
2
3
4
4
2
0
3

Zeichnung:

Lösung:



Richtlinien zur Bewertung:

1 Punkt wenn die Anzahl der Knoten gleich 5 ist

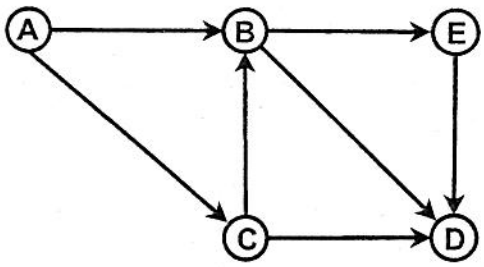
1 Punkt wenn die Anzahl der Kanten gleich 8 ist

1 Punkt bei korrekter Lösung (alle 8 Kanten richtig)

-0.5 Punkte für jede falsche Kante

mind. 0 Punkte für diese Aufgabe

4.2. Topologische Sortierung (1 Punkt) Geben Sie eine topologische Sortierung der Knoten des folgenden Graphen an. Eine gerichtete Kante $\mathbf{A} \rightarrow \mathbf{B}$ bedeutet, dass B von A abhängig ist.



Sortierung:

Lösung:

Die einzig mögliche Sortierung ist: A, C, B, E, D

Richtlinien zur Bewertung:

-0.5 Punkte für jede verletzte Abhängigkeit / pro Vertauschung

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

Lösung:

Gerichtet und zyklenfrei.

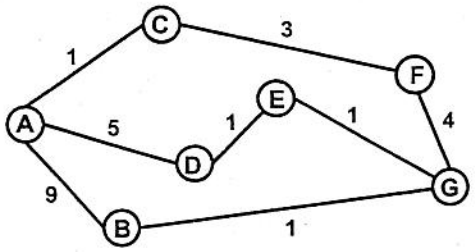
Richtlinien zur Bewertung:

1 Punkt für die richtige vollständige Antwort

0.5 Punkte für einen richtigen Begriff

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							

Lösung:

Iteration	*A*	*B*	*C*	*D*	*E*	*F*	*G*
0	*0*	∞	∞	∞	∞	∞	∞
1	*0*	*9*	*1*	*5*	∞	∞	∞
2		*9*	*1*	*5*	∞	*4*	∞
3		*9*		*5*	∞	*4*	*8*
4		*9*		*5*	*6*		*8*
5		*9*			*6*		*7*
6		*8*					*7*
7							

Richtlinien zur Bewertung:

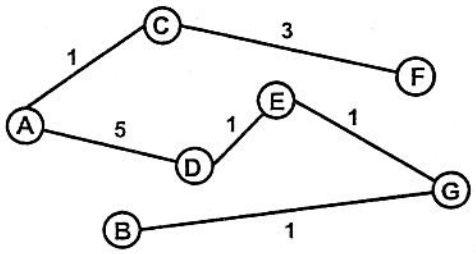
4 Punkte für die korrekte Lösung.

Für jeden falschen Eintrag 0.5 Punkte Abzug (Folgefehler führen nicht zu Punktabzug).

0.5 Punkte Abzug falls der Wert zu früh nicht geschrieben wird.

Für die Aufgabe gibt es mindestens 0 Punkte.

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

Richtlinien zur Bewertung:

0.5 Punkte für jedes richtige Kreuz, also 2 Punkte insgesamt

-0.5 Punkte für jedes falsche Kreuz

-0.5 Punkte, wenn in Tabelle Werte zu früh nicht mehr eingetragen

Für die Aufgabe gibt es mindestens 0 Punkte.

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.

Lösung:

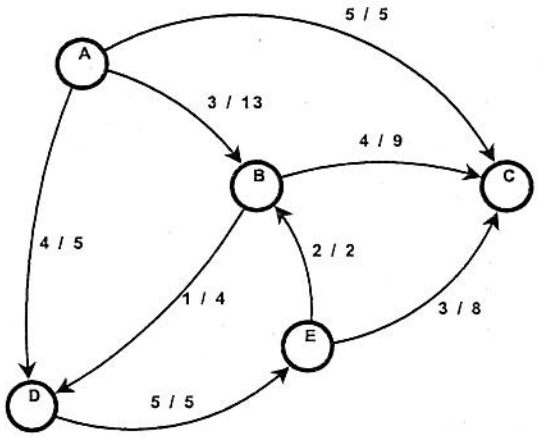
Der Aufbau des Heaps der Warteschlange kostet $O(\|V\|)$. *Jede* `dequeue` *Operation der Prioritätenwarteschlange benötigt* $O(\log \|V\|)$. *Insgesamt werden* $\|V\|$ *dieser Operationen benötigt. Die Modifikation der Distanzwerte (relax) impliziert ein Umordnen des Heaps, das in* $O(\log \|V\|)$ *Schritten durchgeführt werden kann (*`Q.decrease`*). Höchstens* $\|E\|$ *dieser Operationen sind erforderlich. Wir erhalten also* $O(\|V\| \log \|V\| + \|E\| \log \|V\|) = O((\|V\| + \|E\|) \log \|V\|) = O(\|E\| \log \|V\|)$.

Richtlinien zur Bewertung:

0.5 Punkte für den Heap-Aufbau
0.5 Punkte für den log beim Einfügen und Löschen aus der PrioQueue
0.5 Punkte für $O(\|V\| \log \|V\| + \|E\| \log \|V\|)$
0.5 Punkte für das Kürzen auf $O(\|E\| \log \|V\|)$

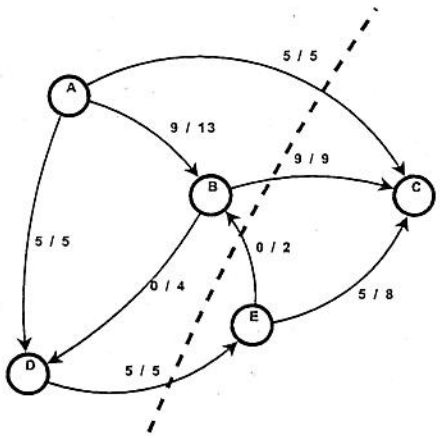
5. Aufgabe (10 Punkte): Flüsse in Netzwerken

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.

Lösung:
Die Quelle ist A, die Senke C. Es müssen die folgenden Kanten geändert werden, um den Fluss zu maximieren.



Richtlinien zur Bewertung:

Positivbewertung:

jeweils 0.5 Punkte für Quelle und Senke erkannt und benannt.

1 Punkt, wenn der minimale Schnitt korrekt eingezeichnet wurde.

+ 2.0 Punkte, wenn der Fluss maximal und keine Eigenschaft des Flussgraphen verletzt ist. oder nur + 1 Punkt, wenn der Fluss korrekt erhöht wurde, aber nicht maximal ist. oder nur + 1 Punkt, wenn der Fluss "eigentlich" maximal ist, aber ein Fehler unterlaufen ist
(eine Eigenschaft des Flussgraphen verletzt ist).

bzw. Negativbewertung:

1 Punkt Abzug, wenn der Fluss nicht maximal ist.

1 Punkt Abzug, wenn eine Eigenschaft des Flussgraphen verletzt wird.

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.

Lösung:

In jedem Schritt wird ein Kantenzug (Pfad/Weg auch noch ok) im Restflussgraph von der Quelle zur Senke gesucht. Dann wird die minimale Restflusskapazität des Kantenzuges ermittelt. Der Fluss wird entlang des Pfades um dieses Minimum erhöht. Der Algorithmus bricht ab, wenn kein Pfad mehr gefunden werden kann.

Richtlinien zur Bewertung:

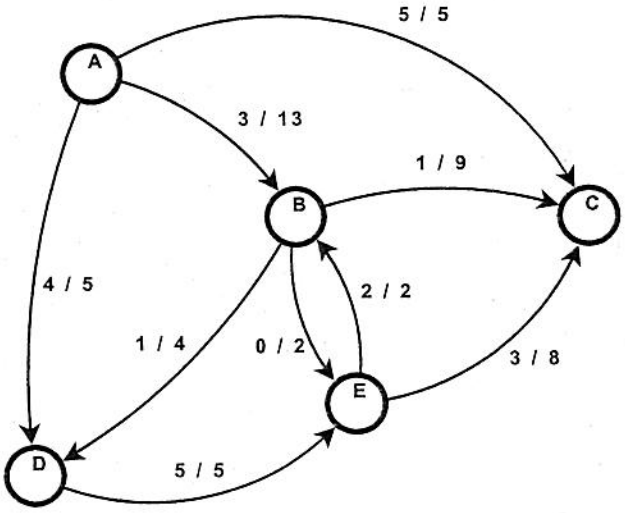
0.5 Punkte für Suche nach einem Kantenzug/Weg/Pfad im Restflussgraphen.

0.5 Punkte für minimale Restkapazität ermitteln.

0.5 Punkte für Erhöhung um dieses Minimum.

0.5 Punkte für Abbruch, wenn kein Pfad mehr gefunden wird.

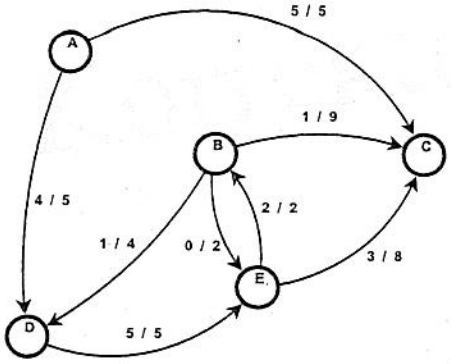
5.3. Restflussgraph (4 Punkte) Betrachten Sie den folgenden Flussgraphen.
Der eingezeichnete Fluss ist **nicht** gültig.



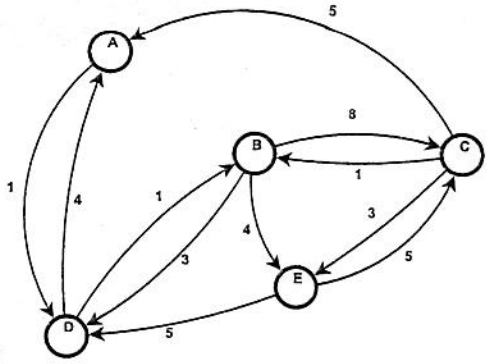
- Entfernen Sie **genau eine** Kante des Graphen, so dass der Fluss gültig wird
- und zeichnen Sie den zugehörigen Restflussgraphen nach dem Entfernen der Kante.

Lösung:

Es muss die folgende Kante entfernt werden, um den Fluss gültig zu machen.



Restflussgraph:



Richtlinien zur Bewertung:

Positivbewertung:

1.5 Punkte, wenn die richtige Kante entfernt wurde.

oder 0.5 Punkte bei Entfernen der falsche Kante, aber verändern der Flüsse, so dass Fluss gültig ist.

+2.5 Punkte, wenn der Restflussgraph richtig ist.

-0.5 Punkte für jede falsche Kante im Restflussgraphen.

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.**

Lösung:

```
1  public static double cosine(double x) {
2
3          double sum = 1.0;
4          double sign;
5          int n = 1;
6          long fac = 1;
7          double pot = 1.0;
8          double oldSum;
9          do {
10                 oldSum = sum;
11                 sign = Math.pow(-1.0,(double)(n));
12                 fac *= (2*n)*(2*n-1);
13                 pot *= x*x;
14                 sum += sign * (pot / fac);
15                 n++;
16         } while (Math.abs(sum - oldSum) >= 1e-4);
17         return sum;
18 }
```

Richtlinien zur Bewertung:

+3.5 Punkte für die korrekte Implementierung der Formel.

+1.5 Punkte für die korrekte Abbruchbedingung der Berechnung

-1 Punkt für kleine Fehler (Start bei 0, Vorzeichen falsch, Fakultät oder Potenz falsch)

-0.5 Punkte für Java-Fehler (keine Syntaxfehler)

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!

Lösung:

```
1  public boolean isNearMathCosine(double x, double eps){
2          return (Math.abs(cosine(x) - Math.cos(x)) < eps);
3  }
```

Richtlinien zur Bewertung:

2 Punkte für korrekte Implementierung (mit Absolutwertbetrachtung)

1 Punkt falls die Absolutwertbetrachtung fehlt, aber der Rest korrekt ist

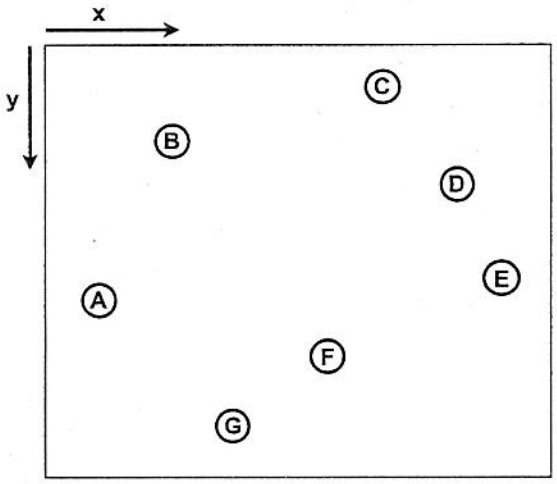
-1 Punkt falls `true` *oder* `false` *verwendet wurde.*

-0.5 Punkte für Java-Fehler (keine Syntaxfehler)

7. Aufgabe (9 Punkte): Bäume

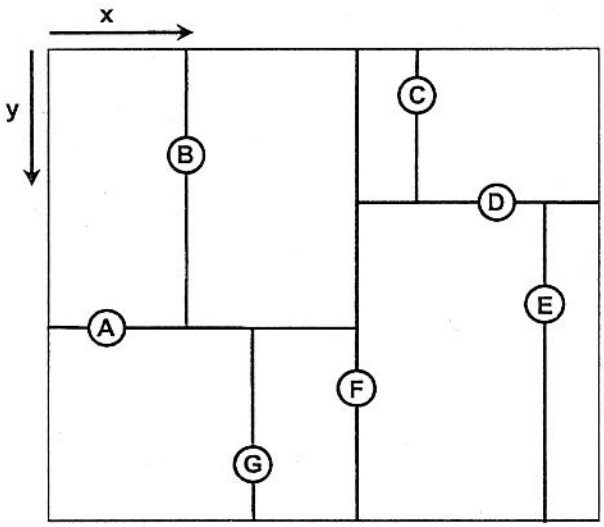
7.1. kD-Bäume (3 Punkte) Nachfolgend ist die Verteilung von Punkten in einem zweidimensionalen Raum dargestellt. Diese Punkte sollen in einen kD-Baum eingefügt werden. Beachten Sie dabei, dass der Koordinatenursprung in dieser Aufgabe links oben liegt.

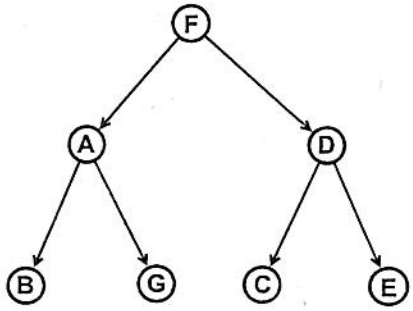
- Verwenden Sie zur Aufteilung des Raumes den Median und stellen Sie die von Ihnen gewählte Aufteilung durch Trennlinien dar. In Schicht 0 des Baumes wird nach den x-Koordinaten sortiert.
- Zeichnen Sie im nebenstehenden Feld anschließend die Sortierung der Punkte als kD-Baum.



Baum:

Lösung:





Richtlinien zur Bewertung:

0,5 Punkte wenn Teilung abwechselnd in x und y Richtung vorgenommen wurde.

0,5 Punkte wenn Teilung jeweils am Median erfolgte.

1 Punkt wenn Baum konsistent zu den gezeichneten Unterteilungen ist.

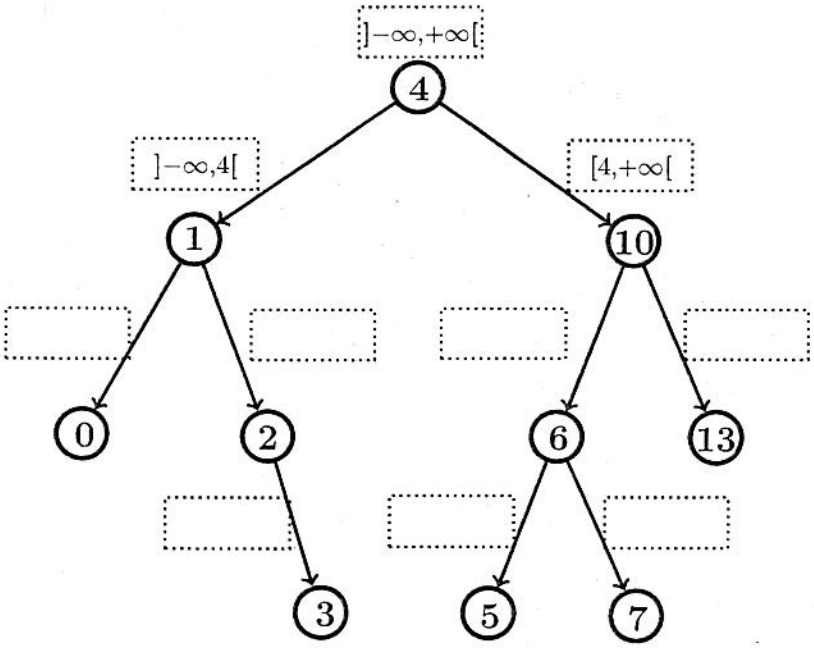
0.5 Punkte wenn Baum ein Suchbaum ist (Punkte mit kleinerer X bzw. Y Koordinate werden jeweils IMMER links oder IMMER rechts an ihren Vorgänger angehangen.)

0,5 Punkte wenn Baum ein Suchbaum nach gängigen Konventionen ist (Punkte mit kleinerer X bzw. Y Koordinate werden jeweils IMMER links an ihren Vorgänger angehangen.)

-0.5 Punkte wenn in y-Richtung begonnen wurde.

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.

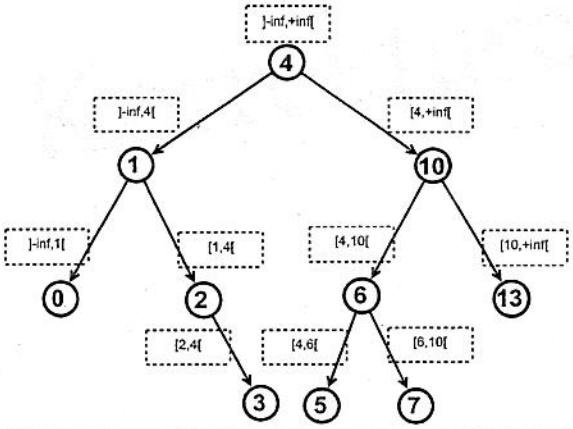
- Geben Sie für jeden Knoten in dem gegebenen Suchbaum an, welches Suchintervall für diesen Knoten gilt. Achten Sie auf die richtige Klammerung der Intervallgrenzen. Die Regel dafür wird durch die Beispielangaben impliziert.



- 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
-	(- , -)	∞	$]-\infty,+\infty[$

Lösung:



aktueller Knoten	*aktualisierte Lösung*	*aktuelles Bewertungskriterium*	*Suchintervall*
4	*(-,4)*	∞	$]-\infty,+\infty[$
10	*(4,10)*	*5*	*[4,14]*
6	*(6,10)*	*3*	*[6,12]*
7	*(7,10)*	*2*	*[7,11]*
13	*(7,10)*	*2*	*[7,11]*

Richtlinien zur Bewertung:

2 Punkte für korrekt ausgefüllten Suchbaum. (1 Punkt für korrekte Werte, 1 Punkt für korrekte Klammern)

-0.5 Punkte pro Fehler in dem Suchbaum, minimal 0 Punkte.

Je 1 Punkt pro richtiger Spalte, wobei nur Bewertungskriterium oder Suchintervall richtig sein muss, also insgesamt 5 Punkte

-0.5 Punkte pro falschen/nicht genannten Knoten

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.

Lösung:

Lösungsvariante 1: Branch & Bound, weil Teile des Suchbaums nicht betrachtet werden.

Lösungsvariante 2: Schrittweise Annäherung /Heuristische Suche, weil es eine Anfangslösung gibt, die verbessert wird.

Richtlinien zur Bewertung:

1 Punkt pro richtiger Antwort mit Begründung

0 Punkte wenn die Begründung fehlt oder falsch ist.