

TECHNISCHE UNIVERSITÄT BERLIN

Fakultät IV – Elektrotechnik und Informatik
Fachgebiet Neurotechnologie (MAR 4-3)
Prof. Dr. Benjamin Blankertz
Röhr / Schlegel



Algorithmen und Datenstrukturen, SoSe 24

Bitte füllen Sie alle folgenden Felder aus:

Vorname:

Nachname:

TUB-Kontoname:

Matrikelnummer:

Studiengang:

Hochschule:

Durch meine Unterschrift bestätige ich die Korrektheit obiger Angaben sowie meine Prüfungsfähigkeit und die Anmeldung zur Prüfung!

Ort, Datum

Unterschrift

Beachten Sie die folgenden Hinweise!

- Sie brauchen Ihren Namen **nur** auf das Deckblatt zu schreiben. Die restlichen Blätter können über die Klausur-ID zugeordnet werden.
- Diese Klausur besteht mit diesem Deckblatt aus den (nummerierten) Seiten **1-16**.
- Am Ende der Klausur befinden sich zwei leere Seiten, die Sie für Notizen verwenden können. Sollten Sie mehr Papier benötigen, können Sie dies von der Aufsicht bekommen. Notieren Sie in diesem Fall die Klausur-ID auf dem Zusatzblatt.
- Notieren Sie Ihre Antworten nur auf dem Blatt (inklusive Rückseite), auf dem die zugehörige Aufgabe steht, da die Aufgaben getrennt korrigiert werden.
- Falls Sie eine Antwort auf ein Zusatzblatt schreiben, markieren Sie dies klar bei der zugehörigen Aufgabe und auf dem Zusatzblatt.
- Geben Sie nur eine Lösung pro Aufgabe ab, streichen Sie alle alternativen Lösungsansätze auf Schmier-/Notizblättern durch.
- Schreiben Sie **nicht** mit roter Farbe, grüner Farbe (Korrekturfarben) oder Bleistift. Diese Lösungen werden nicht bewertet!
- Insgesamt können in der Klausur **100 Punkte** erreicht werden.

Zusatzblätter:



Punktetabelle

Aufgabe	Punkte		
1	/11		
2	/11		
3	/12		
4	/14		
5	/14		
6	/9		
7	/11		
8	/12		
9	/6		
Σ	/ 100		

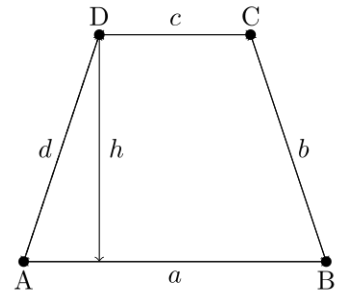


- (b) Die Klasse Trapezoid repräsentiert ein Trapez, wobei die Kanten des Trapezes als Edge gespeichert sind.

Ihr Coding Assistent hat Ihnen die Methoden zur Berechnung des Flächeninhalts `area` und des Umfangs `perimeter` vorgeschlagen. Sie können davon ausgehen, dass alle Funktionsaufrufe innerhalb der Methoden korrekt sind und funktionieren. `length()` gibt hierbei Länge einer Kante zurück und hat eine konstante Laufzeit.

Hinweis: Der Flächeninhalt eines Trapezes kann mit folgender Formel berechnet werden:

$A = \frac{1}{2} \cdot h \cdot (a + c)$ mit der Höhe h und den Längen a, c der parallelen Seiten.



```

1 public class Trapezoid {
2     private Edge paralleleEdgeA;
3     private Edge edgeB;
4     private Edge paralleleEdgeC;
5     private Edge edgeD;
6     private double height;
7
8     public Trapezoid(Edge paralleleEdgeA, Edge edgeB, Edge paralleleEdgeC,
9         Edge edgeD, double height) {
10        this.paralleleEdgeA = paralleleEdgeA;
11        this.edgeB = edgeB;
12        this.paralleleEdgeC = paralleleEdgeC;
13        this.edgeD = edgeD;
14        this.height = height;
15    }
16    //to evaluate:
17    public double area() {
18        double sideA = paralleleEdgeA.length();
19        double sideC = paralleleEdgeC.length();
20        return 0.5 * (sideA + sideC) * height;
21    }
22    //to evaluate:
23    public static double perimeter(Edge[] edges) {
24        double perimeter = 0.0;
25        for (Edge edge : edges) {
26            for (int i = 0; i < edges.length; i++) {
27                perimeter += edge.length();
28            }
29        }
30        return perimeter;
31    }

```

Geben Sie die Größenordnung der Laufzeit von den Methoden `area` und `perimeter` mit den richtigen abhängigen Größen an. Geben Sie außerdem an, ob die jeweilige Methode funktioniert, also die Fläche bzw. den Umfang korrekt bestimmt. Begründen Sie kurz, falls eine Methode fehlerhaft ist.



Aufgabe 2: Java-Programmierung (2 + 7 + 2 = 11 Punkte)

Sie haben hier die Klasse `MatrixImage`, welche das Interface `Image` implementiert. Die Klasse soll eine Bildrepräsentation als Integer-Array implementieren.

```
1 public interface Image {
2
3     int sizeX();
4     int sizeY();
5
6 }
7
8 public class MatrixImage implements Image {
9     public int[][] field;
10
11     /**
12      * A Constructor for the MatrixImage class
13      * @param sx the size on the x-axis
14      * @param sy the size on the y-axis
15      */
16     public MatrixImage(int sx, int sy) {
17         //TODO a)
18     }
19
20     /**
21      * A deep copy constructor for a given MatrixImage that
22      * @param that the to be copied MatrixImage
23      */
24     public MatrixImage(MatrixImage that) {
25         //TODO b)
26     }
27
28     /**
29      * @return the size on the x-axis
30      */
31     @Override
32     public int sizeX() {
33         return field.length;
34     }
35
36     /**
37      * @return the size on the y-axis
38      */
39     @Override
40     public int sizeY() {
41         return field[0].length;
42     }
43 }
```



Aufgabe 3: Greedy Raumplanung (4 + 4 + 4 = 12 Punkte)

Zur Bestimmung der Belegung eines Konferenzraums sollen aus einer Liste von Raumbuchungsanfragen **möglichst viele** ausgewählt werden, die sich nicht überlappen. Dabei kann die Endzeit einer Buchung mit der Anfangszeit der folgenden Buchung übereinstimmen. Für die Auswahl wird der optimale Greedy Algorithmus für *Interval Scheduling* aus der Vorlesung genutzt.

Die ersten beiden Klassen sind **fehlerfrei**. Die Klasse `Interval` repräsentiert die Zeitintervalle der Buchungsanfragen. Die Klasse `IntervalCompareFinishTime` vergleicht Objekte der Klasse `Interval` auf Basis der Endzeitpunkte.

```

1 public class Interval {
2     private double s; //start
3     private double f; //finish
4     public Interval(double s, double f)
5     {
6         this.s = s;
7         this.f = f;
8     }
9     public double start() { return s; }
10    public double finish() { return f; }
11 }

```

```

1 public class
  IntervalCompareFinishTime
  implements Comparator<Interval>
  {
2     @Override
3     public int compare(Interval i1,
4                       Interval i2) {
5         return Double.compare(i1.
6                               finish(), i2.finish());
7     }
8 }

```

Durch Fehler in der Klasse `IntervalScheduling` zeigt der Greedy Ansatz nicht das erwartete Verhalten. Die Importe wurden in allen Klassen dieser Aufgabe weggelassen. Sie können davon ausgehen, dass diese korrekt sind.

```

1 public abstract class IntervalScheduling {
2     public static Iterable<Interval> intervalScheduling(Iterable<Interval>
3     intervals) {
4         PriorityQueue<Interval> pq =
5             new PriorityQueue<>(new IntervalCompareFinishTime());
6         for (Interval i : intervals) {
7             pq.add(i);
8         }
9         Queue<Interval> queue = new LinkedList<>();
10        double finishTime = 0;
11        while (!pq.isEmpty()) {
12            Interval iv = pq.peek();
13            if (iv.start() >= finishTime) {
14                queue.add(iv);
15            }
16        }
17        return queue;
18    }
19    public static void main(String[] args) {
20        Queue<Interval> intervals = new LinkedList<>();
21        intervals.add(new Interval(1, 2));
22        intervals.add(new Interval(2, 4));
23        intervals.add(new Interval(3, 5));
24        intervals.add(new Interval(4, 7));
25        intervals.add(new Interval(5, 6));
26        Iterable<Interval> jobs = intervalScheduling(intervals);
27    }
28 }

```



- (a) Korrigieren Sie die Zeilen 8-16 der Klasse `IntervalScheduling`, sodass der Greedy Ansatz die richtigen Ergebnisse liefert. Sie brauchen dabei nur die Zeilen aufzuschreiben, die berichtigt oder hinzugefügt werden müssen. Geben Sie dabei die Zeilennummern an. Wenn Sie zur Berichtigung neue Zeilen einfügen, geben Sie an, zwischen welchen Zeilen diese eingefügt werden müssen.
Hinweis: Es sind keine reinen Syntaxfehler im Code.

<i>Zeile(n) des Fehlers</i>	<i>Berichtigung</i>

- (b) Führen Sie den korrigierten Algorithmus mit dem in der `main` gegebenen Beispiel aus. Listen Sie die gewählten Intervalle auf!
Hinweis: Die `PriorityQueue` sortiert basierend auf der Klasse `IntervalCompareFinishTime`.

- (c) Nun stehen genügend Räume zur Verfügung und es sollen alle Buchungsanfragen erfüllt werden. Durch den folgenden Pseudocode wird immer dann ein neuer Raum hinzugenommen, wenn es keinen Raum gibt, der aktuell die `if`-Abfrage in Zeile 12 erfüllen kann.

```

1 if Intervall iv ist kompatibel zu einem Raum R[k] für ein 0 ≤ k < K
2   füge iv zu R[k] hinzu, k minimal gewählt
3 else
4   initialisiere neuen Raum R[K]
5   füge iv zu R[K] hinzu
6   K++

```

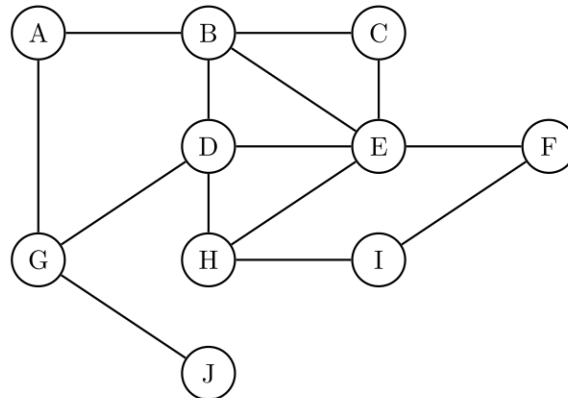
Wie werden dadurch die Buchungsanfragen in der `main` auf die Räume aufgeteilt werden?

<i>Raum</i>	<i>Intervalle</i>



Aufgabe 4: Tiefensuche (9 + 2 + 2 + 1 = 14 Punkte)

Gegeben ist der folgende ungerichtete und ungewichtete Graph G :



- (a) Führen Sie die Tiefensuche auf dem Graphen G aus. Beginnen Sie die Tiefensuche im **Knoten A**. Notieren Sie alle Knoten in der Reihenfolge, in der sie von der Tiefensuche gefunden werden (**Nebenreihenfolge/preorder**) und in der sie abgearbeitet werden, d.h. die **Hauptreihenfolge** bzw. **postorder**.

Gehen Sie dabei davon aus, dass bei Wahlmöglichkeit (Tiebreak) die benachbarten Knoten in **alphabetischer Reihenfolge** abgearbeitet werden.

Nebenreihenfolge: **A**

Hauptreihenfolge: **J**

- (b) Welche Kante müsste aus dem oben dargestellten Graphen G entfernt werden, damit ein anderer Knoten als **J** als erstes in der Hauptreihenfolge steht? Dabei soll der Graph weiterhin zusammenhängend bleiben.
- (c) Erklären Sie, wie die reine Tiefensuche ergänzt werden muss, um einen **Pfad** vom Knoten s aus zu Knoten v zu bestimmen, wenn Knoten v vom Knoten s aus erreichbar ist. Geben Sie ebenfalls an, wie der Pfad dann bestimmt wird.

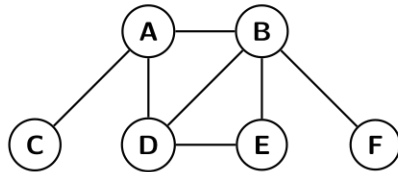
- (d) Wenn ein solcher Pfad per Tiefensuche gefunden wurde, hat dieser dann immer minimale Länge? Wenn ja, begründen Sie. Wenn nein, geben Sie ein Gegenbeispiel an.



Aufgabe 5: Radius eines Graphen (3 + 2 + 6 + 3 = 14 Punkte)

Zu einem zusammenhängenden, ungewichteten Graphen $G = (V, E)$ werden folgende Begriffe definiert: Die Anzahl der Kanten eines kürzesten Weges zwischen zwei Knoten v, w bezeichnen wir als Abstand $dist(v, w)$. Die *Exzentrizität* eines Knotens v ist als das Maximum der Abstände von v zu allen anderen Knoten des Graphens definiert $\epsilon(v) := \max\{dist(v, w) \mid w \in V\}$. Das Minimum der Exzentrizitäten aller Knoten in einem Graphen nennt man den *Radius* des Graphen $radius(G) := \min\{\epsilon(v) \mid v \in V\}$.

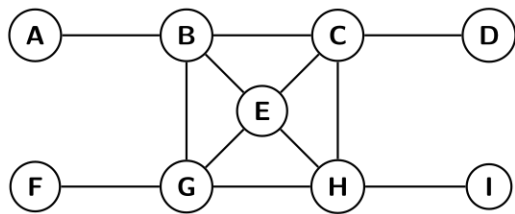
- (a) Bestimmen Sie im folgenden Graphen die Exzentrizität der Knoten **A** und **C**.



$\epsilon(\mathbf{A}) =$

$\epsilon(\mathbf{C}) =$

- (b) Markieren Sie den Knoten mit der niedrigsten Exzentrizität in dem folgenden Graphen.



- (c) Geben Sie eine exakte Beschreibung (bzw. Pseudocode) eines Verfahrens an, um den **Radius** eines gegebenen Graphen zu bestimmen. Sie dürfen dabei die aus der Vorlesung bekannten Verfahren benutzen (genauen Namen des Algorithmus angeben!). Ihre Lösung sollte eine *worst case* Laufzeit in $O(V(V + E))$ besitzen.

- (d) Begründen Sie die Laufzeit Ihres Algorithmus' aus (c). Verwenden Sie nach Möglichkeit die Voraussetzung, dass der Graph **zusammenhängend** ist, um eine bessere Abschätzung als $O(V(V + E))$ zu erreichen.



Aufgabe 7: Hashing (8 + 3 = 11 Punkte)

Es wird eine Hashtabelle der Größe 10 zur Speicherung von Integer-Werten betrachtet, d.h., die Schlüssel sind vom Typ Integer. Die Hashadresse eines Integer k wird durch

$$\text{hash}(k) = k \bmod 10$$

berechnet. In dieser Aufgabe geht es um die Anzahl von Schlüsselvergleichen bei Kollisionsauflösung durch Lineares Sondieren (Linear Probing) mit Inkrement 1. In der Hashtabelle unter (a) benötigt eine Suche nach dem Schlüssel **17** *einen* Schlüsselvergleich (mit 17) und eine Suche nach dem Schlüssel **7** *zwei* Schlüsselvergleiche (mit 17 und mit 7).

(a) Die folgenden vier Schlüssel sollen in die untenstehenden Hashtabellen eingetragen werden:

4, 12, 22, 23

Tragen Sie die Schlüssel bei der ersten Tabelle in einer Reihenfolge ein, die die Anzahl der Schlüssel (aus **4**, **12**, **22** und **23**) maximiert, bei denen die Suche jeweils mit nur einem Schlüsselvergleich auskommt.

Index	0	1	2	3	4	5	6	7	8	9
Key								17	7	

Maximiere Anzahl der Schlüssel 4, 12, 22, 23, bei denen die Suche mit *einem* Schlüsselvergleich auskommt.

Bei der zweiten Tabelle soll eine Reihenfolge gewählt werden, die die *maximale* Anzahl der Schlüsselvergleiche bei der Suche nach den Schlüssel **4**, **12**, **22**, **23** *minimiert*.

Index	0	1	2	3	4	5	6	7	8	9
Key								17	7	

Minimiere *Worst Case* Anzahl von Schlüsselvergleichen bei der Suche nach 4, 12, 22, 23.

(b) Für diese Teilaufgabe ist die Hashtabelle mit anderen Schlüssel gefüllt worden. Es soll zunächst der Schlüssel **13** und danach der Schlüssel **8** gesucht werden. Schreiben Sie die Schlüssel aus der Hashtabelle auf, mit denen der gegebene Schlüssel bei der Suche verglichen wird.

Hinweis: Eine der beiden Suchen ist erfolglos, da der Schlüssel nicht in der Hashtabelle enthalten ist.

Index	0	1	2	3	4	5	6	7	8	9
Key	8	19		23	3	14		27	7	18

Schlüssel bei Suche nach **13**:

Schlüssel bei Suche nach **8**:



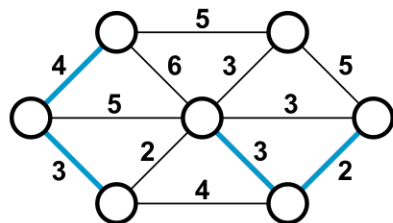
Aufgabe 8: Minimaler Spannbaum (6 + 6 = 12 Punkte)

In der folgenden Abbildung sehen Sie sechs mal den gleichen Graphen, wobei jedes mal unterschiedliche Kanten (**blau**) markiert sind. *Hinweis:* Lesen Sie zuerst die ganze Aufgabe.

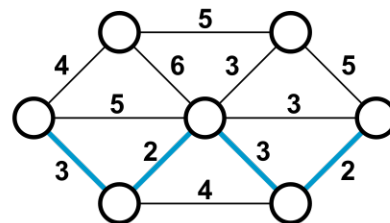
- (a) Kreuzen Sie für jeden der sechs Graphen an, ob die markierten Kanten nur durch den **Prim** Algorithmus, nur durch den **Kruskal** Algorithmus, durch **beide** oder durch **keinen** von beiden ausgewählt worden sein können. Dabei muss der jeweilige Algorithmus nicht bis zum Ende durchgelaufen sein. Es können also auch die Markierungen **während** der Berechnung eines MST abgebildet sein.
- (b) Markieren Sie außerdem in jedem Graphen folgendes:

gewählte Algorithmen	Markierung
<i>Prim</i>	einen Startknoten für den Prim Algorithmus, der zu der dargestellten Kantenauswahl führt
<i>Kruskal</i>	eine Kante, die Kruskal als nächstes auswählen könnte
<i>beide</i>	die Markierungen für <i>Prim</i> und <i>Kruskal</i> , siehe oben
<i>keiner</i>	eine der hervorgehobenen Kanten, die in einer partiellen Lösung von Kruskal nicht ausgewählt worden wäre

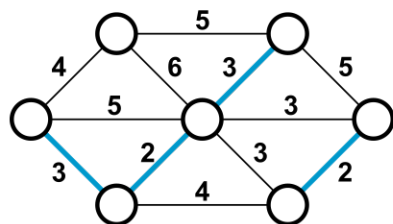
Markieren Sie Kanten, indem Sie das Gewicht umkreisen, und Knoten, indem Sie sie ausmalen. Wenn Sie eine andere Markierung wählen, z. B. weil Sie etwas verändern wollen, schreiben Sie eine lesbare und eindeutige Legende daneben.



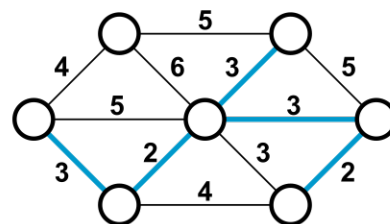
Prim beide
 Kruskal keiner



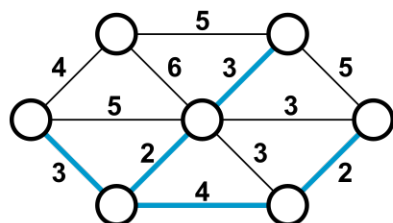
Prim beide
 Kruskal keiner



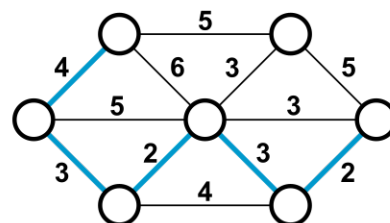
Prim beide
 Kruskal keiner



Prim beide
 Kruskal keiner



Prim beide
 Kruskal keiner



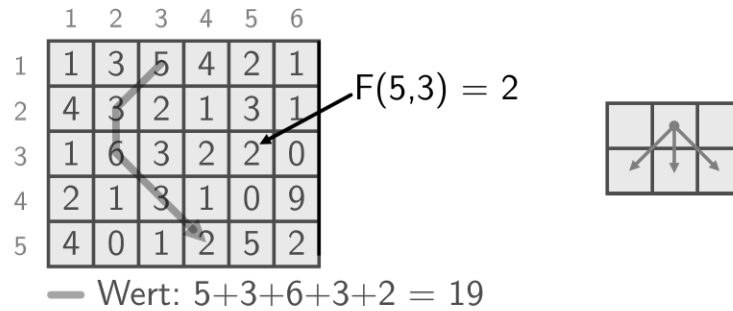
Prim beide
 Kruskal keiner



Aufgabe 9: Dynamische Programmierung (4 + 2 = 6 Punkte)

Es ist eine Matrix der Größe $M \times N$ mit nicht-negativen ganzen Zahlen gegeben. Mit Hilfe dynamischer Programmierung soll ein Weg von der obersten zu der untersten Zeile gefunden werden, der die maximale Summe 'einsammelt', siehe Abbildung. Der Weg kann dabei in einer beliebigen Zelle der ersten Zeile beginnen und in einer beliebigen Zelle der letzten Zeile enden.

Bei jedem Schritt verläuft der Weg von einer Zelle in eine der gerade oder diagonal darunter liegenden Zellen, also von einem Feld (m, n) zu einem der Felder $(m-1, n+1)$, $(m, n+1)$ oder $(m+1, n+1)$, sofern diese Felder innerhalb der Matrix liegen. Es gilt $M \geq 3$ und $N \geq 3$. Der Wert eines Feldes (m, n) wird als $F(m, n)$ bezeichnet.



Die Abbildung links zeigt ein Raster der Größe 6×5 in dem ein Weg von der obersten zur untersten Zeile eingezeichnet ist. Der Wert des Weges ist die Summe der durchlaufenen Felder, also $5 + 3 + 6 + 3 + 2 = 19$. Die Darstellung rechts zeigt, auf welche Art der Weg von einer Zeile in die nächste verlaufen kann.

- (a) Definieren Sie eine rekursive Funktion $\text{OPT}(m, n)$, die den größten Wert eines Weges von der obersten Zeile zu dem Feld (m, n) erzielt ($1 \leq m \leq M, 1 \leq n \leq N$). Verwenden Sie eine mathematische Schreibweise und keinen Java- oder Pseudo-Code.

- (b) Die Werte der OPT-Funktion werden in einer Matrix $D[m][n]$ für alle $1 \leq m \leq M$ und $1 \leq n \leq N$ gespeichert. Geben Sie ein Verfahren an, mit dem die Lösung (maximale Summe) aus dieser Matrix D bestimmt werden kann.

Hinweis: Sie brauchen zur Lösung dieser Teilaufgabe die Lösung von (a) nicht zu kennen. Die Beschreibung der OPT-Funktion aus der Aufgabenstellung reicht aus.



Diese Seite können Sie für Notizen verwenden. Bitte nur im Ausnahmefall für Lösungen verwenden!



Diese Seite können Sie für Notizen verwenden. Bitte nur im Ausnahmefall für Lösungen verwenden!

