

Aufgabe 1: Multiple Choice ($8 \times 2 = 16$ Punkte)

Bitte kreuzen Sie alle wahren Aussagen an.

- Es ist bei allen Teilaufgaben mindestens eine Aussage wahr, aber auch mindestens eine Aussage falsch.
- Ein gesetztes Kreuz kann durch ein leeres \circ -Symbol links neben der Aussage aufgehoben werden.
- Es werden nur für vollständig richtige Teilaufgaben Punkte vergeben.

(a) Welche Wachstumsordnung beschreibt die Laufzeit der folgenden Methode $f()$ am genauesten?

```
public static int f(int N) {
    if (N <= 1)
        return 0;
    return 2 * f(N/2);
}
```

<input checked="" type="radio"/>	$O(\log(N))$
<input type="radio"/>	$O(N)$
<input type="radio"/>	$O(2N)$
<input type="radio"/>	$O(N \log(N))$
<input type="radio"/>	$O(N^2)$
<input type="radio"/>	$O(2^N)$

(b) Welche Wachstumsordnung hat die Laufzeit des Dijkstra Algorithmus für einen Graphen $G = (V, E)$ mit nicht-negativen Kantengewichten, wenn er mit einer indizierten Vorrangwarteschlange (*index PQ*) implementiert wird?

<input type="radio"/>	$O(V + E)$
<input type="radio"/>	$O(VE)$
<input type="radio"/>	$O(V \log E)$
<input type="radio"/>	$O(V^2 \log E)$
<input checked="" type="radio"/>	$O(E \log V)$

(c) Für einen Spannbaum T eines zusammenhängenden Graphen $G = (V, E)$ gilt:

<input type="radio"/>	T hat V Kanten.
<input checked="" type="radio"/>	T ist zusammenhängend und das Entfernen einer beliebigen Kante macht ihn unzusammenhängend.
<input type="radio"/>	T hat genau einen Zyklus.
<input type="radio"/>	T benutzt jede Kante des Graphen G genau einmal.
<input checked="" type="radio"/>	Zwei beliebige Knoten von G werden durch einen einfachen Pfad der in T liegt verbunden. Ein einfacher Pfad ist ein Pfad, in dem keine Knoten doppelt vorkommen.
<input type="radio"/>	T ist eindeutig, wenn G gerichtet ist.



(d) Welche Aussagen treffen auf den Algorithmus von Kruskal zur Bestimmung eines minimalen Spannbaums zu?

<input checked="" type="radio"/>	Es wird eine UnionFind Struktur benutzt, um effizient zu überprüfen, ob das Hinzufügen einer neuen Kante einen Zyklus erzeugen würde.
<input type="radio"/>	Es wird eine UnionFind Struktur benutzt, um effizient zu überprüfen, ob das Weglassen einer neuen Kante den Graphen unzusammenhängend machen würde.
<input type="radio"/>	Der Algorithmus hat große Ähnlichkeit mit dem Dijkstra-Algorithmus.
<input type="radio"/>	Der Baum wird schrittweise ausgehend von einem Startknoten s aufgebaut.
<input type="radio"/>	Der Algorithmus ist nicht effizient, da er Rekursion verwendet.
<input checked="" type="radio"/>	Die Kanten werden nach Gewicht sortiert durchlaufen, angefangen bei einer Kante mit minimalem Gewicht.

(e) Der A Algorithmus ...

<input checked="" type="radio"/>	benutzt eine Heuristik, um die kürzesten Pfade in Richtung eines gegebenen Zielknotens schneller zu finden.
<input type="radio"/>	hat eine Laufzeit in $\mathcal{O}(V + E)$, wenn die Heuristik konsistent ist.
<input checked="" type="radio"/>	relaxiert jede Kante nur einmal, wenn die Heuristik konsistent ist.
<input type="radio"/>	löst Kollisionen durch sternförmige Sondierung auf.
<input type="radio"/>	ist ein approximativer Algorithmus.
<input type="radio"/>	findet kürzere Pfade als der Dijkstra Algorithmus, wenn die Heuristik zulässig ist.

(f) Welche Aussagen zum Thema Hashing sind zutreffend?

<input type="radio"/>	Bei Hashing mit separater Verkettung sollte der Belegungsfaktor (load factor) unter $1/2$ liegen.
<input checked="" type="radio"/>	Kollisionen können durch universelles Hashing vermieden werden.
<input checked="" type="radio"/>	Gute Hashfunktionen bilden die Schlüssel möglichst gleichmäßig auf die Hashadressen ab.
<input checked="" type="radio"/>	Beim Hashing kann man beliebige Objekte als Schlüssel verwenden, nicht nur Integerzahlen.
<input type="radio"/>	Hashtabellen benötigen in den meisten Fällen weniger Speicher, als normale verkettete Listen.
<input type="radio"/>	Der Zugriff über Hashtabellen ist bei Schlüsseln vom Typ Integer schneller als bei einem normalen Array.



- (g) Die Busverbindungen einer Region seien als Graph modelliert. Dabei stellen die Knoten Haltestellen dar und die Kanten stehen für eine Fahrt von Haltestelle zu Haltestelle. Für jede Kante ist die Streckenlänge bekannt. Es soll die Verbindung mit den wenigsten Zwischenstopps (also Haltestellen auf dem Weg) gefunden werden. Start- und Zielhaltestelle sind dabei gegeben. Welcher der folgenden Algorithmen eignet sich dafür? Bei unterschiedlichen Möglichkeiten soll derjenige mit der besten Laufzeit-Garantie angekreuzt werden.

<input checked="" type="radio"/>	Breitensuche
<input type="radio"/>	Tiefensuche
<input type="radio"/>	Topologisches Sortieren
<input type="radio"/>	Dijkstra-Algorithmus
<input type="radio"/>	Kruskal-Algorithmus
<input type="radio"/>	Edmonds-Karp

- (h) Welche der folgenden Aussagen treffen den Kern des Beweises, der zeigt, dass sich das *Traveling Salesman Problem* (unter der Annahme $P \neq NP$) nicht in polynomieller Zeit approximieren lässt?

<input type="radio"/>	Zu einem vorgegebenen $\varepsilon > 0$ konstruiert man einen Algorithmus, dessen Kosten größer als $(1 + \varepsilon)$ mal den Kosten der optimalen TSP Tour sind.
<input checked="" type="radio"/>	Der approximative Algorithmus wird genutzt, um das NP-vollständige Problem der Existenz eines Hamilton-Zyklus in polynomieller Zeit zu lösen.
<input type="radio"/>	Man fängt mit einer hohen Kostenschranke an und halbiert diese solange, bis man die Approximationsgrenze unterschreitet.
<input type="radio"/>	Man startet mit dem minimalen Spannbaum des Graphen und baut diesen zu einem Zyklus um, dessen Kosten maximal das doppelte der Kosten der optimalen TSP Tour sind.



Aufgabe 2: Hashing (1 + 4 + 6 = 11 Punkte)

Benutzen Sie die Hashfunktion: $h((x, y)) = (7x - 2y) \pmod 7$

(a) Berechnen und notieren Sie die restlichen Hashwerte in der folgenden Tabelle:

Schlüssel	Hashcode	Hashvalue
A	(1, 1)	5
B	(3, 6)	2
C	(2, 1)	5
D	(3, 3)	1
E	(4, 5)	4
F	(4, 3)	1
G	(4, 2)	3
H	(3, 2)	3
I	(2, 3)	1

(b) Fügen Sie die Schlüssel A-G in alphabetischer Reihenfolge in die Hashtabelle ein. Verwenden Sie dabei zur Kollisionsauflösung lineares Sondieren mit Inkrement 1 (d.h. $s(n) = n$).

0	1	2	3	4	5	6
C	D	B	F	E	A	C

(c) Entscheiden Sie welche der folgenden Szenarien durch das Einfügen der Schlüssel A-G aus a) in beliebiger Reihenfolge entstehen können und kreuzen Sie diese an. Hierbei wird wie oben zur Kollisionsauflösung lineares Sondieren mit Inkrement 1 genutzt.

Hinweis: Es ist mindestens eine Antwort richtig und eine Antwort falsch. Für jedes richtig gesetzte und jedes richtig nicht gesetzte Kreuz bekommen Sie 2 Punkte.

<input type="radio"/>	0	1	2	3	4	5	6	kein einziger Schlüssel ist am Hash-Bucket ohne Kollision => unmöglich
	F	A	D	B	G	E	C	
<input type="radio"/>	0	1	2	3	4	5	6	keine gültige Reihenfolge. nach B, G kann der nächste Schlüssel nicht platziert werden.
	C	A	B	G	F	E	D	
<input checked="" type="radio"/>	0	1	2	3	4	5	6	F, D, B, G, E, C, A funktioniert.
	A	F	D	B	G	E	C	



Aufgabe 3: Laufzeit (4 + 6 = 10 Punkte)

Eine Variante zum Finden minimaler Spannbäume (die nicht in der Vorlesung besprochen wurde) beruht auf der folgenden Eigenschaft:

Sei ein beliebiger Zyklus in einem Graphen gegeben, und sei e eine der darin enthaltenen Kanten mit dem höchsten Gewicht. Dann gibt es einen minimalen Spannbaum, der die Kante e nicht enthält.

Der Pseudocode für diesen MST-Algorithmus sieht wie folgt aus:

```

1 // gegeben ein gewichteter Graph  $G = (V, E)$ 
2 sortiere die Kanten absteigend nach ihrem Gewicht
3 for jede Kante  $e \in E$  in dieser Reihenfolge:
4   if  $e$  ist Teil eines Zyklus von  $G$ :
5     entferne  $e$  aus  $G$ 
6   end
7 end
8 return  $G$  //  $G$  ist der MST
```

- (a) Als Teil des Algorithmus muss für eine gegebene Kante e geprüft werden, ob e Teil eines Zyklus in G ist. Beschreiben Sie einen Algorithmus, der dies in linearer Zeit erledigt. Dabei können Sie Verfahren benutzen, die in der Vorlesung besprochen wurden.

Sei $G=(V, E)$ der Graph und $e=(v,w) \in E$ die zu prüfende Kante.
 e ist Teil eines Zyklus, genau dann wenn in $G'=(V, E \setminus e)$ ein Weg von v nach w existiert.

Dies kann mit einer simplen DFS überprüft werden. Man startet bei v und überprüft, ob w erreichbar ist.

Die Laufzeit der DFS ist $O(|V| + |E|)$.

- (b) Wie ist die Laufzeit des gegebenen Algorithmus? Die Laufzeit muss für jeden Teil des Pseudocodes angegeben und daraus die Gesamtlaufzeit folgerichtig abgeleitet werden.

Z. 3: Die Schleife wird $|E|$ -Mal ausgeführt.

Z. 4: Die Zyklensuche ist wie in 3.a) festgestellt in $O(|V| + |E|)$.

Z. 5: Das Entfernen einer Kante ist in $O(1)$ möglich.

Z. 8: Das Zurückgeben des Resultats ist in $O(1)$ möglich.

Da Z. 4-6 $|E|$ -mal ausgeführt werden, ist die Gesamtlaufzeit in
 $O(|E| \cdot (|V| + |E| + 1)) = O(|E| \cdot |V| + |E|^2 + |E|) = O(|E| \cdot |V| + |E|^2)$



Aufgabe 4: Objektorientierte Programmierung (4 + 4 + 2 + 1 = 11 Punkte)

Im Folgenden ist unvollständiger Code vorgegeben.

- (a) Vervollständigen Sie den Code (an den unterstrichenen Stellen) so, dass er kompiliert und dass die Klassen Car und Bicycle von der Klasse Vehicle erben.

```
1 abstract class _____ Vehicle {
2     protected int numberOfWheels;
3
4     abstract void refillAir(int num);
5
6     public Vehicle(int numberOfWheels) {
7         this.numberOfWheels = numberOfWheels;
8     }
9
10    public void repair() {
11        for (int i = 0; i < numberOfWheels; i++) {
12            refillAir(i);
13        }
14    };
15 }
```

```
1 class _____ Car extends Vehicle _____ {
2     public Car() {
3         super(4);
4     }
5
6     public void refillAir(int num)
7     {
8         System.out.print("c"+num);
9     }
10 }
```

```
1 class _____ Bicycle extends Vehicle _____ {
2     public Bicycle() {
3         super(2);
4     }
5
6     public void refillAir(int num)
7     {
8         System.out.print("b"+num);
9     }
10 }
```

PS: Wichtig sind hier das "abstract", da ansonsten Java sich beschwert das eine abstrakte Funktion in einer nicht-abstrakten Klasse ist, und das keine Klasse "public" ist, da Java fordert, dass der Dateiname zum Namen der öffentlichen Klasse identisch ist (was impliziert, dass höchstens eine öffentliche Klasse pro Datei existieren darf).



- (b) Betrachten Sie nun die folgende Methode:

```
1 public class Garage {
2     public static void main(String[] args) {
3         Vehicle [] allVehicles = new Vehicle[3];
4         Vehicle [] vehicles = new Vehicle[3];
5         Car c1 = new Car();
6         allVehicles[0]=c1;
7         Bicycle b1 = new Bicycle();
8         allVehicles[1]=b1;
9         Bicycle b2 = new Bicycle();
10        allVehicles[2]=b2;
11
12        try{
13            for (Vehicle v : allVehicles)
14                v.repair();
15        }
16        catch(Exception e){
17            System.out.println(e);
18        }
19    }
20 }
```

Was wird bei der Ausführung auf der Konsole ausgegeben?

c0c1c2c3b0b1b0b1

- (c) Betrachten Sie weiterhin die Klasse Garage. Wenn Sie den try-catch-Block (Zeile 12-18) mit dem folgenden ersetzen, was gibt das Programm aus?

```
1     try{
2         for (int i=0; i<=3; i++)
3             allVehicles[i].repair();
4     }
5     catch(Exception e){
6         System.out.println(e);
7     }
```

c0c1c2c3b0b1b0b1java.lang.ArrayIndexOutOfBoundsException: 3

- (d) Betrachten Sie weiterhin die Klasse Garage. Wenn Sie den try-catch-Block (Zeile 12-18) mit dem folgenden ersetzen, was gibt das Programm nun aus?

```
1     try{
2         for (int i=0; i<3; i++)
3             vehicles[i].repair();
4     }
5     catch(Exception e){
6         System.out.println(e);
7     }
```

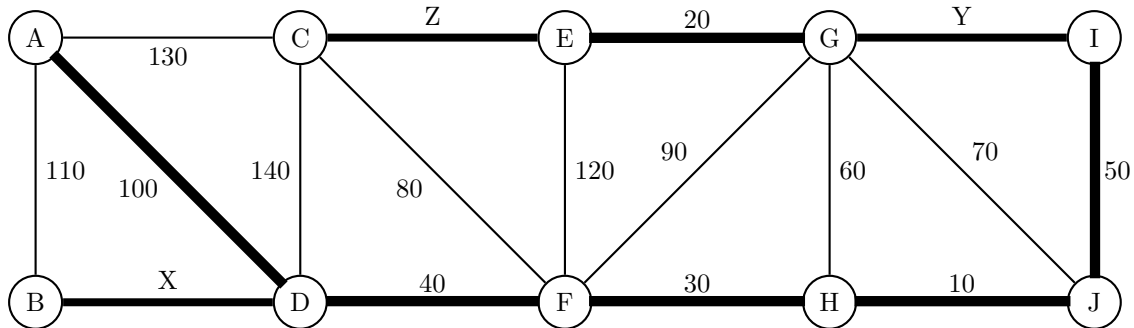
java.lang.NullPointerException

PS: Diese Ausgaben habe ich mit einer Online-IDE verifiziert.



Aufgabe 5: Minimum Spanning Tree (6 + 3 = 9 Punkte)

Nehmen Sie an, der Minimum Spanning Tree (MST) dieses Graphen enthalte die Kanten X, Y und Z.



- (a) Notieren Sie die Gewichte der restlichen Kanten, d.h. alle außer X, Y und Z, die zum MST dieses Graphen gehören müssen unter der obigen Annahme.

10,20,30,40,50,100

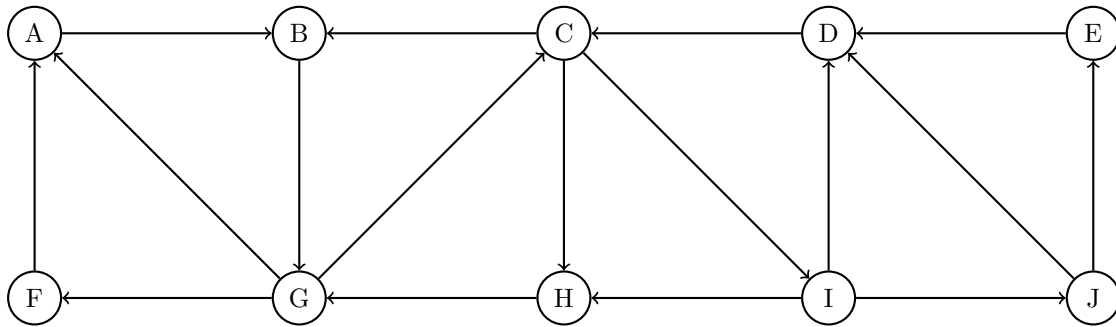
- (b) Geben Sie jeweils die größte obere Schranke für die Gewichte der Kanten X, Y und Z an, mit der garantiert ist, dass alle drei Kanten tatsächlich Teil des MST sind.
Hinweis: Geben Sie individuell für jedes Gewicht x , y , z eine Schranke an, wobei x , y und z die entsprechenden Gewichte zu den Kanten X, Y und Z sind.

$x < 110$

$y < 60$

$z < 80$



Aufgabe 6: Tiefensuche (6 + 6 = 12 Punkte)

Führen Sie eine Tiefensuche auf dem Graphen aus. Fangen Sie im Knoten A an.

Gehen Sie dabei davon aus, dass in jedem Knoten die benachbarten Knoten in alphabetischer Reihenfolge abgearbeitet werden. Z.B. würde die Kante I-D vor der Kante I-J bearbeitet werden.

- (a) Notieren Sie die Knoten in der Hauptreihenfolge (*pre-order*), d.h. in der Reihenfolge, in der sie entdeckt werden.

A, B, G, C, H, I, D, J, E, F

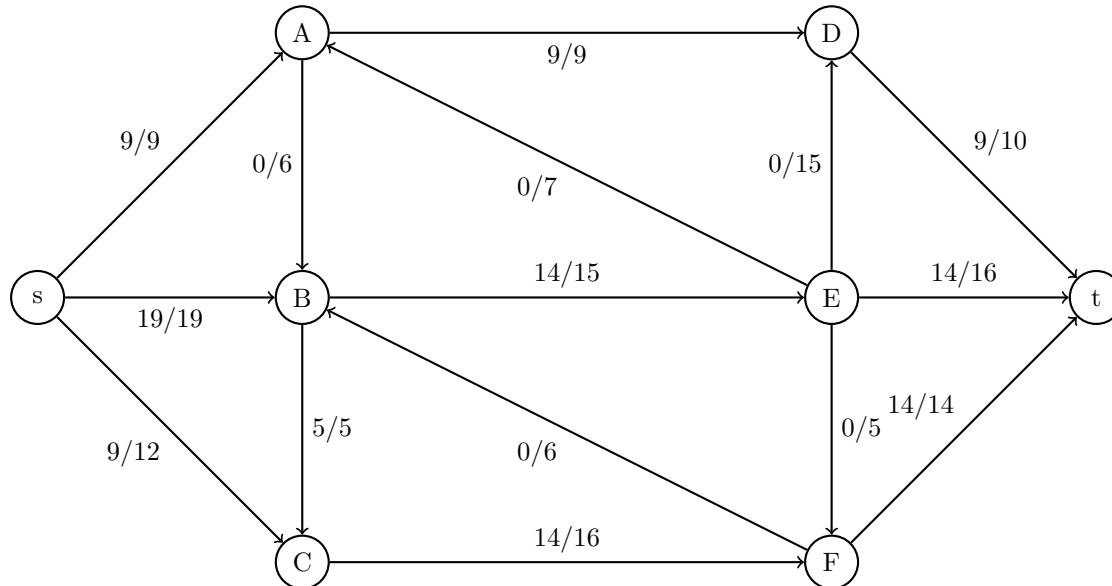
- (b) Notieren Sie die Knoten in der Nebenreihenfolge (*post-order*), d.h. in der Reihenfolge, in der sie fertig abgearbeitet sind. Ein Knoten ist abgearbeitet, wenn alle ausgehenden Nachbarn bereits bearbeitet wurden.

H, D, E, J, I, C, F, G, B, A



Aufgabe 7: Edmonds-Karp Algorithmus (2 + 5 + 1 + 1 = 9 Punkte)

Betrachten Sie den folgenden Flussgraphen mit Quelle s und Senke t und dem eingetragenen Fluss:



- (a) Bestimmen Sie den Wert, den der Fluss in der Abbildung hat.

Flusswert = **37**

- (b) Führen Sie eine Iteration des Edmonds-Karp Algorithmus aus und notieren Sie die Knoten eines vergrößernden Pfades beginnend in s und endend in t .

$p = \{(s, C), (C, B), (B, E), (E, t)\}$

- (c) Notieren Sie den Wert des maximalen Flusses des oben gezeigten Flussgraphen.
Hinweis: Nach b) ist der Fluss maximal ist.

$f' = 38$

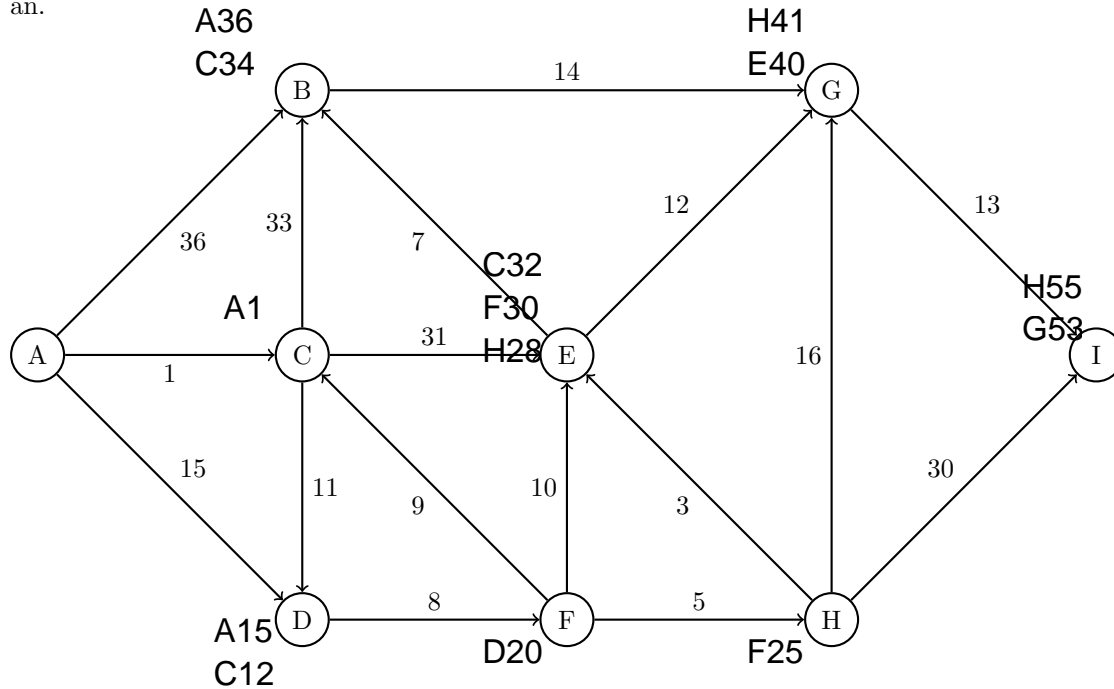
- (d) Notieren Sie die Kapazität des minimalen Schnittes in dem obigen Flussgraphen.

Die Kapazität des minimalen Schnitts ist 38.



Aufgabe 8: Dijkstra Algorithmus (8 + 2 = 10 Punkte)

Betrachten Sie den folgenden gerichteten Graphen. Die Zahlen an den Pfeilen geben die Kantengewichte an.

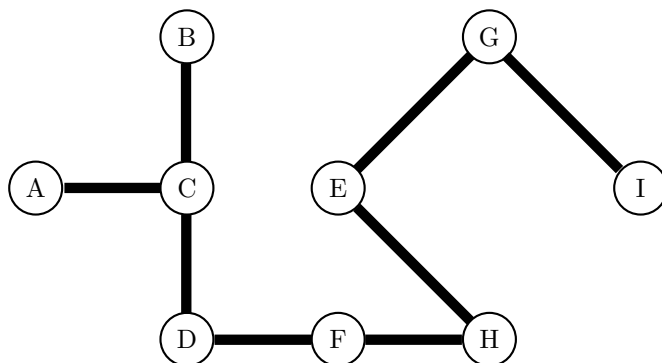


Führen Sie Dijkstras Algorithmus auf dem obigen Graphen ausgehend von dem Startknoten A aus.

- (a) Notieren Sie die Knoten in der Reihenfolge, in der sie aus der Priorityqueue entfernt werden. Geben Sie zudem zu jedem Knoten die Distanz zum Knoten A an, definiert über die Kantengewichte.

Knoten	A	C	D	F	H	E	B	G	I
Distanz	0	1	12	20	25	28	34	40	53

- (b) Zeichnen Sie alle kürzesten Pfade in den Graphen ein.



Aufgabe 9: Dynamisches Programmieren (3 + 7 + 2 = 12 Punkte)

Ein *Palindrom* ist eine Zeichenfolge die rückwärts gelesen dieselbe Folge ergibt wie vorwärts gelesen. Das Wort RENTNER ist also ein Palindrom. Eine *Palindrom-Teilfolge* ist eine Teilfolge einer Zeichenkette, die ein Palindrom ist. Dabei muss die Teilfolge nicht an einem Stück in der gegebenen Zeichenkette vorkommen.

Beispiel: DATENSTRUKTUREN enthält u.a. die Palindrom-Teilfolgen NRUKNR und ERUTURE. Auch UKU, NN und jeder einzelne Buchstabe sind (kurze) Palindrom-Teilfolgen der Zeichenkette DATENSTRUKTUREN.

Es soll ein Algorithmus entwickelt werden, der zu einer gegebenen Zeichenkette `str` die Länge der längsten Palindrom Teilfolge bestimmt.

- (a) Welche Laufzeit hat der *brute-force* Ansatz (alle Möglichkeiten durchprobieren)?

$O(n \cdot 2^n)$ (2^n 'Teilfolgen', die in je $n/2$ Operationen überprüft werden)

- (b) Geben Sie eine rekursive Definition der $\text{Opt}(i, j)$ an, die als Grundlage für dynamisches Programmieren verwendet werden kann.

$\text{Opt}(i, j)$ ist hierbei die Länge der längsten Palindrom-Teilfolge von `str[i:j]`.

$$\text{OPT}(i,j) = \begin{cases} 1 & , \text{ falls } i = j \\ 2 & , \text{ falls } \text{str}[i] = \text{str}[j] \text{ und } i = j \\ \text{OPT}(i+1, j-1) & , \text{ falls } \text{str}[i] = \text{str}[j] \\ \max(\text{OPT}(i+1, j), \text{OPT}(i, j-1)) & \text{ ansonsten} \end{cases}$$

- (c) Was ist die Laufzeit einer Implementation mit Hilfe der Opt Funktion und dynamischer Programmierung.

Hinweis: Sie können c) unabhängig von b) lösen.

$O(n^2)$ (es gibt n^2 Einträge in der Matrix, die Ermittlung eines einzelnen Eintrags der Matrix kann in konstanter Zeit geschehen)

