Fakultät IV

Klausur MPGI 2
27. Juli 2012

Brock
Eppner / Höfer / Putz / Rouiller

Name: .....

Vorname: .....

Matr.-Nr.: .....

Bearbeitungszeit: 75 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, gib unbedingt an, zu welcher Aufgabe die Lösung gehört!
- Schreib 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.
- Gib Nebenrechnungen an, um Deinen Lösungsweg zu verdeutlichen.
- Trage *jetzt* (vor Beginn der Bearbeitungszeit) auf *allen* Blättern Deinen Namen und Deine Matrikelnummer ein.

	Punkte	Erreichte Punkte
1	12	
2	11	
3	12	
4	10+6	
5	12	
6	8	
Σ	65+6	

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.

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.

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

g_1:

g_2:

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

A:

B:

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.)

-

-

-

2. Aufgabe (11 Punkte): Sortieren

Gegeben sei folgender Algorithmus:

```
class Algorithm {
  public static int[] doSomething(int[] numbers) {
    int minValue = getMin(numbers);

    int range = getMax(numbers) − minValue + 1;
    int histogram[] = new int[range];

    for (int i = 0; i < range; i++)
      histogram[i] = 0;

    for (int x : numbers)
      histogram[x − minValue]++;

    // — hierauf bezieht sich Frage 2.1.

    int total = 0;
    for (int i = 0; i < range; i++) {
      int tmp = histogram[i];
      histogram[i] = total;
      total += tmp;
    }

    int result [] = new int[numbers.length];
    for (int x : numbers) {
      result [histogram[x − minValue]] = x;
      histogram[x − minValue]++;
    }
    return result;
  }

  public static int getMax(int[] set) {
    int currentMax = set[0];
    for (int x : set)
      currentMax = Math.max(currentMax, x);

    return currentMax;
  }

  public static int getMin(int[] set) {
    int currentMin = set[0];
    for (int x : set)
      currentMin = Math.min(currentMin, x);

    return currentMin;
  }
}
```

2.1. Algorithmus verstehen (4 Punkte) Angenommen der Algorithmus wird folgendermaßen aufgerufen:

```
Algorithm.doSomething(new int[]{5, 2, 8, 4});
```

Welchen Wert hat dann die Variable `histogram` unmittelbar vor der Kommentarzeile "`// --- hierauf bezieht sich Frage 2.1.`"?

`histogram` = { }

Wie lautet der Rückgabewert des Aufrufes?

`Algorithm.doSomething(new int[]{5, 2, 8, 4})` = { }

2.2. Algorithmus identifizieren (2 Punkte) Um welchen Algorithmus handelt es sich?

2.3. Aufwand (2 Punkte) Sei $n = numbers.length$ und $m = getMax(numbers) - getMin(numbers)$. Wie lautet die kleinste obere Schranke für die Laufzeit von `doSomething(int[] numbers)` in Abhängigkeit von n und m? (O-Notation)

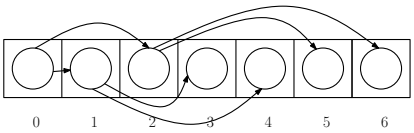
2.4. Stabile Sortieralgorithmen (3 Punkte) Wann ist ein Sortieralgorithmus *stabil*? Antworte in einem Satz.

Handelt es sich bei `doSomething(int[] numbers)` um einen stabilen Sortieralgorithmus?

☐ Ja
☐ Nein, `doSomething(int[] numbers)` sortiert zwar die Eingabemenge, ist jedoch nicht stabil.
☐ `doSomething(int[] numbers)` ist kein Sortieralgorithmus.

3. Aufgabe (12 Punkte): Bäume

3.1. Baumrepräsentation (4 Punkte) Wir stellen einen beliebigen Binärbaum mit Hilfe eines Arrays folgendermaßen dar:



Mit Hilfe der Indexfunktionen $left(i)$, $right(i)$ bzw. $parent(i)$ kann man das rechte Kind, linke Kind, sowie den Elternknoten des i-ten Knotens im Array erreichen. Wie lauten deren Formeln?

$$left(i) \quad =$$

$$right(i) \quad =$$

$$parent(i) \quad =$$

Gegeben sei das folgende Array:

Index	0	1	2	3	4	5	6	7	8	9
Element	17	10	21	`null`	14	19	23	`null`	`null`	12

Der Wert `null` stellt einen fehlenden Knoten an der entsprechenden Position im zugehörigen Baum dar. Ab Index 10 enthält das Array nur noch den Wert `null`.

Zeichne den daraus resultierenden Baum!

3.2. Traversierung (4 Punkte) Angenommen eine Traversierung des o.g. Binärbaumes besucht dessen Knoten in folgender Reihenfolge:
17, 10, 21, 14, 19, 23, 12.

Welcher Traversierungsart entspricht dies?

☐ Pre-Order
☐ Post-Order
☐ In-Order
☐ keiner der oben genannten

Schreibe einen Algorithmus, der die Knoten in dieser Reihenfolge traversiert und auf der Standardausgabe ausgibt. Du kannst eine einzelne for-Schleife verwenden.

```
public void traverseTree(Integer[] tree) {




}
```

3.3. Binärer Suchbaum (4 Punkte) Zeichne einen binären *Such*baum, der durch das Einfügen der Zahlen **12, 10, 8, 7, 4** (in genau dieser Reihenfolge) entsteht!

Wir möchten die vorherige Array-Implementierung eines Baumes nutzen, um einen binären Suchbaum zu realisieren. Wie groß muss das Array im schlimmsten Fall sein, wenn wir n Elemente einfügen wollen? (Kleinste obere Schranke in O-Notation genügt.)

4. Aufgabe (10+6 Punkte): Graphen

Tim ist neu in der Stadt. An der Universität hat er bereits ein paar Freunde kennengelernt, die ihn in ein soziales Netzwerk eingeladen haben. Ein Ausschnitt dieses Netzwerks ist in dem Graphen in Abbildung 1 gezeigt. Jeder Knoten des Freundesnetzwerks stellt ein Mitglied dar. Kanten zwischen Mitgliedern stehen für *direkte* Freundschaften. Beantworte die folgenden Fragen anhand dieses Graphen:

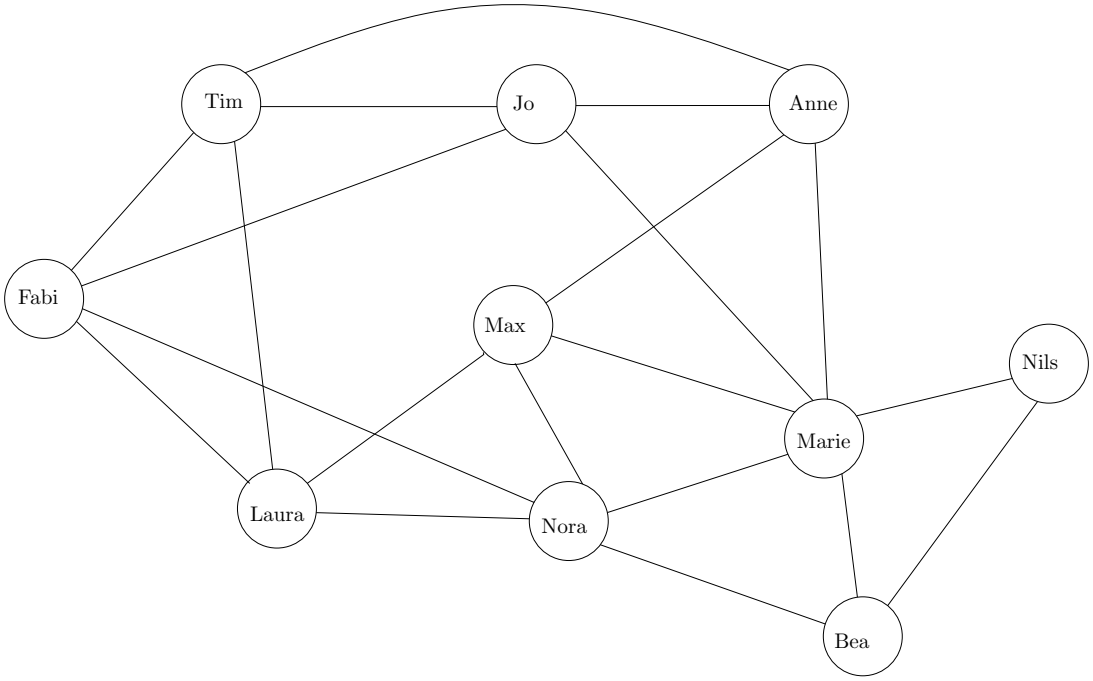


Abbildung 1: Freundesnetzwerk

4.1. Erweiterter Freundeskreis (4 Punkte) Tim möchte seinen Freundeskreis erweitern und plant dafür ein gemeinsames Abendessen. Neben seinen direkten Freunden lädt er zusätzlich noch deren direkte Freunde ein. Markiere alle Personen, die er einlädt, mit einem *Stern* im Graphen. Wie viele Personen sind das?

Beschreibe in nicht mehr als drei Sätzen, *welchen* Graphenalgorithmus man *wie* modifizieren müsste, um dieses Problem zu lösen? Dabei soll nur die *unbedingt notwendige* Anzahl an Knoten betrachtet werden. (Name des Algorithmus genügt nicht.)

4.2. BONUSAUFGABE: Tischaufteilung (6 Punkte) Tim möchte die eingeladenen Personen nun so auf Tische verteilen, dass jeder *exakt zwei* andere Personen pro Tisch *direkt* kennt. Dabei sollen

- *direkte* Freunde auf gegenüberliegenden Seiten des Tisches sitzen,
- Personen, die sich *nur über andere* kennen, *nebeneinander* sitzen,
- nur die *Längsseiten* der Tische besetzt werden.

Tim selbst sitzt natürlich auch an einem der Tische.

Wie viele Tische braucht er dafür?

☐ 2 Tische
☐ 3 Tische
☐ 4 Tische

Wer sitzt zusammen? Zeichne die benötigten Tische als Rechtecke und trage ein, wer wo sitzt. Beachte dabei die oben genannten Einschränkungen!

Betrachtet man jeden Tisch als Teilgraphen des gesamten Freundesnetzwerks, welche *zwei* Grapheigenschaften besitzt dann jeder Tisch?

☐ Er bildet eine Clique im gesamten Freundesnetzwerk.
☐ Er ist bipartit.
☐ Er ist vollständig.
☐ Er enthält einen Eulerkreis.

4.3. Kürzeste Wege (6 Punkte) Tim hat ein Date mit Marie ausgemacht und überlegt, wie er am *schnellsten* zu ihr kommt. Da er etwas schüchtern ist, will er unterwegs noch einem seiner *direkten* Freunde einen Besuch abstatten. Die Gewichte auf den Kanten im Freundesnetzwerk in Abbildung 2 entsprechen den Entfernungen in Minuten zwischen zwei Freunden.

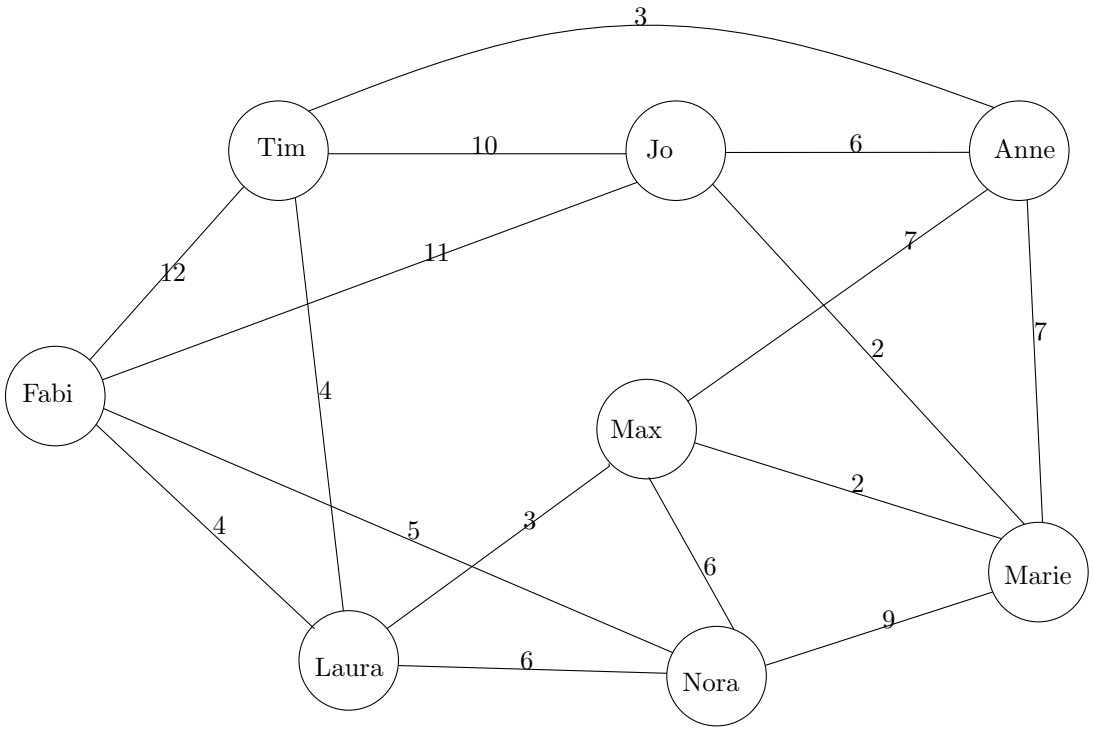


Abbildung 2: Freundesnetzwerk mit Entfernungen

Über welche anderen Mitglieder des Netzwerks führt der kürzeste Weg von Tim zu Marie?

Wieviel Zeit muss Tim für diese Wegstrecke einplanen?

Welcher Algorithmus berechnet diesen kürzesten Weg am effizientesten? Begründe Deine Wahl in einem kurzen Satz!

☐ Floyd-Warshall
☐ Dijkstra
☐ Ford-Fulkerson
☐ Bellman-Ford

Begründung:

5. Aufgabe (12 Punkte): Graphen II

Ein paar Leute aus Tims Freundeskreis benutzen Twitter. Bei Twitter kannst man kurze Textnachrichten versenden (*twittern*). Deine Nachrichten erhalten alle Leute, die einem folgen – die *Follower*. Man kann sowohl twittern als auch Follower von anderen Leuten sein.

Wenn man eine Nachricht, die man erhält, an alle seine Follower schickt, wird das *Retweeten* genannt.

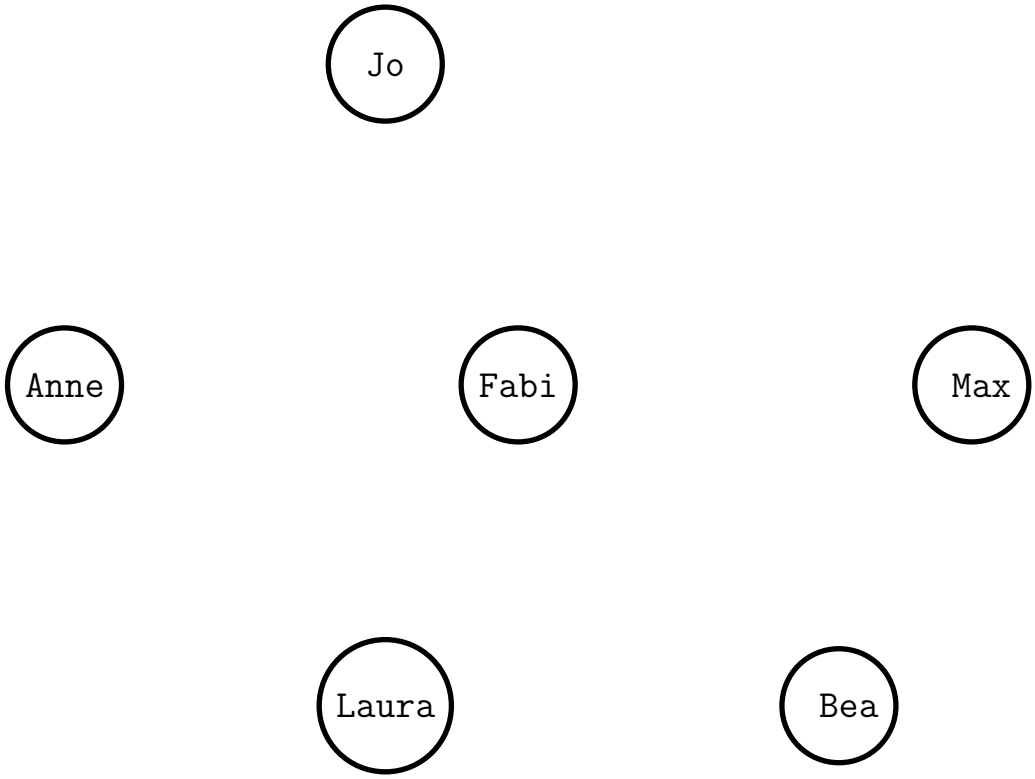
5.1. Graph erstellen (4 Punkte) Wir modellieren in einem Graphen, wer von Tims Freunden Follower von wem ist.

Es gelten folgende Follower-Beziehungen:

- Anne ist Follower von Jo und Laura
- Bea ist Follower von Fabi und Max
- Fabi ist Follower von Bea
- Jo ist Follower von Fabi und Max
- Laura ist Follower von Jo

Die Leute sind außerdem verschieden aktiv. Wir messen das in *Nachrichten pro Tag (NpT)*. Die Aktivitäten der Leute sind: Anne (0 NpT), Bea (2 NpT), Fabi (5 NpT), Jo (3 NpT), Laura (10 NpT) und Max (3 NpT).

Vervollständige den folgenden Graphen! Es soll eine Kante von X zu Y gezeichnet werden, wenn X ein Follower von Y ist. Das Gewicht der Kante X nach Y soll die Aktivität von Y in NpT sein.



5.2. Follower-Beziehungen trennen (6 Punkte) Angenommen, alle retweeten sofort jede Nachricht, die sie erhalten (egal, ob sie schon einmal retweetet wurde oder nicht).

Wer müsste aufhören, Follower von wem zu sein, damit Anne auf keinen Fall mehr retweetete Nachrichten von Max erhält – unter der Bedingung, dass so wenig Nachrichten pro Tag wie möglich verloren gehen?

(Das heißt z. B., dass die Follower-Beziehung zwischen Jo und Fabi mehr wert ist als zwischen Anne und Jo).

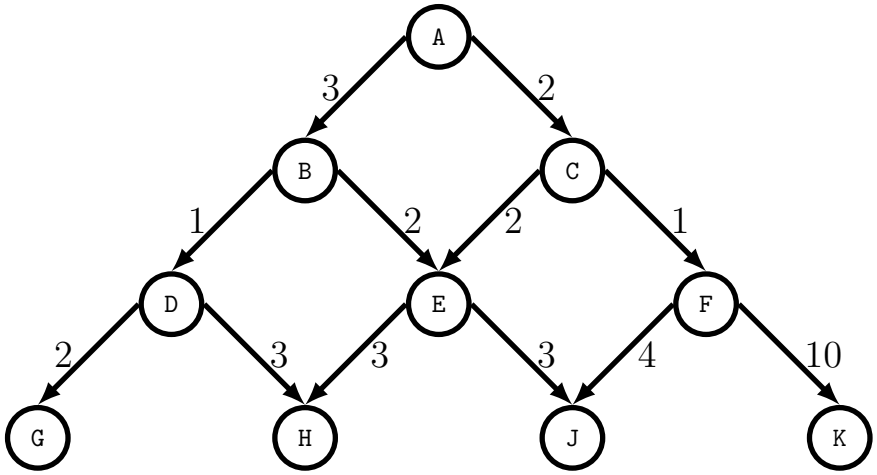
5.3. Abgemeldete Leute (2 Punkte) Was ist die *minimale* Anzahl an Leuten (ausgenommen Anne und Max selbst), die sich von Twitter abmelden muss, damit Anne keine retweeteten Nachrichten mehr von Max lesen muss?

(Die Nachrichten pro Tag können dabei ignoriert werden.)

☐ Einer, nämlich Jo
☐ Zwei Nutzer, nämlich Fabi und Jo, oder Jo und Laura
☐ Drei Nutzer, nämlich Fabi und Jo und Laura
☐ Alle außer Anne und Max

6. Aufgabe (8 Punkte): Algorithmische Prinzipien

Gegeben sei ein *gerichteter* Baum mit der Wurzel A, den Blättern G, H, J und K und gewichteten Kanten:



Wir suchen die Länge des kürzesten Pfades von der Wurzel bis zu einem der Blätter.

Im folgenden sind vier Beispielalgorithmen beschrieben. Gib für jeden Beispielalgorithmus an, welches algorithmische Prinzip ihm zugrunde liegt! (Es gibt jeweils genau eins.)

Randnotiz: In einem gerichteten Baum kann ein Knoten zwei Elternknoten haben (siehe E, H und J).

6.1. Algorithmus I (2 Punkte) Algorithmus I beginnt am Wurzelknoten A und wählt immer die ausgehende Kante mit dem geringeren Gewicht. Dies tut er so lange, bis er ein Blatt erreicht hat. Im obigen Beispiel ist das $(A \to C \to F \to J)$ mit Länge 7.

Hinweis: Der Algorithmus findet in *diesem* Beispiel *nicht* die richtige Lösung.

☐ Backtracking
☐ Branch-and-Bound
☐ Brute-Force (erschöpfende Suche)
☐ Dynamische Programmierung
☐ Divide-and-Conquer
☐ Greedy-Algorithmen
☐ Las-Vegas-Algorithmen
☐ Monte-Carlo-Algorithmen
☐ Keiner der oben genannten

6.2. Algorithmus II (2 Punkte) Algorithmus II geht von unten nach oben vor und berechnet für jeden Teilbaum den kürzesten Pfad: Erst für die Blätter G, H, J, K, dann für D, E, F etc. Eine höhere Ebene benutzt dabei die Teillösungen der tieferen Ebene.

Zum Beispiel berechnen wir für Teilbaum B:

$$\text{Länge kürzester Pfad im Teilbaum } \mathbf{B} = \min \begin{cases} \text{Länge kürzester Pfad im Teilbaum } \mathbf{D} + \text{Gewicht}(B \to D), \\ \text{Länge kürzester Pfad im Teilbaum } \mathbf{E} + \text{Gewicht}(B \to E) \end{cases}$$

Der Algorithmus endet an der Wurzel und findet die richtige Lösung 6.

☐ Backtracking
☐ Branch-and-Bound
☐ Brute-Force (erschöpfende Suche)
☐ Dynamische Programmierung
☐ Divide-and-Conquer
☐ Greedy-Algorithmen
☐ Las-Vegas-Algorithmen
☐ Monte-Carlo-Algorithmen
☐ Keiner der oben genannten

6.3. Algorithmus III (2 Punkte) Algorithmus III berechnet die Länge von *jedem* der acht Pfade von der Wurzel bis zu einem Blatt einzeln.

Danach gibt der Algorithmus die Länge des kürzesten Pfades als Lösung aus, nämlich $(A \to B \to D \to G)$ mit Länge 6.

☐ Backtracking
☐ Branch-and-Bound
☐ Brute-Force (erschöpfende Suche)
☐ Dynamische Programmierung
☐ Divide-and-Conquer
☐ Greedy-Algorithmen
☐ Las-Vegas-Algorithmen
☐ Monte-Carlo-Algorithmen
☐ Keiner der oben genannten

6.4. Algorithmus IV (2 Punkte) Algorithmus IV durchsucht den Baum, startend an der Wurzel. Er wählt den nächsten zu besuchenden Knoten ähnlich wie Dijkstra (kürzeste Distanz zur Wurzel). Der Unterschied ist, dass der Algorithmus die erwartete Länge des Pfades mittels einer unteren und einer oberen Schranke abschätzt. Die obere Schranke *überschätzt* die Distanz zu den Blättern, die untere Schranke *unterschätzt* sie.

Ein Knoten wird *nicht* besucht, wenn dessen untere Schranke *größer* als die obere Schranke irgend eines anderen Knotens ist.

Der Algorithmus findet die richtige Lösung $(A \to B \to D \to G)$ mit Länge 6.

☐ Backtracking
☐ Branch-and-Bound
☐ Brute-Force (erschöpfende Suche)
☐ Dynamische Programmierung
☐ Divide-and-Conquer
☐ Greedy-Algorithmen
☐ Las-Vegas-Algorithmen
☐ Monte-Carlo-Algorithmen
☐ Keiner der oben genannten