Prof. Dr. Ir. Joost-Pieter Katoen

Christian Dehnert, Friedrich Gretz, Benjamin Kaminski, Thomas Ströder

Allgemeine Hinweise:

- Die **Hausaufgaben** sollen in Gruppen von je **2-3 Studierenden** aus der **gleichen Kleingruppenübung (Tutorium)** bearbeitet werden. **Namen und Matrikelnummern** der Studierenden sind auf jedes Blatt der Abgabe zu schreiben. **Heften bzw. tackern Sie die Blätter!**
- Die **Nummer der Übungsgruppe** muss **links oben** auf das **erste Blatt** der Abgabe geschrieben werden. Notieren Sie die Gruppennummer gut sichtbar, damit wir besser sortieren können.
- Die Lösungen müssen **bis Mittwoch, den 01.07.2015 um 12:15 Uhr** in den entsprechenden Übungskasten eingeworfen werden. Sie finden die Kästen am Eingang Halifaxstr. des Informatikzentrums (Ahornstr. 55). Alternativ können Sie die Lösungen auch vor der Abgabefrist direkt bei Ihrer Tutorin/Ihrem Tutor abgeben.
- In Aufgaben, bei denen Sie Algorithmen implementieren sollen, dürfen Sie Ihre Lösung als Pseudo-Code abgeben. Abgaben in verbreiteten imperativen Sprachen wie Java oder C++ sind ebenfalls erlaubt.

Tutoraufgabe 1 (Bellman-Ford Algorithmus):

a) Passen Sie die Implementierung des Bellman-Ford-Algorithmus, die Sie in der Vorlesung kennengelernt haben (Vorlesung 17, Folie 13), so an, dass der kürzeste Pfad zwischen zwei Knoten i und j ausgegeben (nicht zurückgegeben!) wird. Sie dürfen davon ausgehen, dass der übergebene Graph keine negativen Zykel besitzt.

Ihre Funktion soll die folgende Signatur haben:

```
void bellFord(List adjLst[n], int n, int i, int j)
```

Zur Ausgabe können Sie annehmen, dass eine Funktion `ausgabe` mit folgender Signatur existiert:

```
void ausgabe(String text)
```

Außerdem können Sie annehmen, dass `int`-Werte automatisch nach `String` konvertiert werden können.

b) Einem Studenten gefällt die Laufzeit des Bellman-Ford-Algorithmus nicht und er schlägt das folgende alternative Verfahren vor, um Zyklen mit negativem Gesamtgewicht zu erkennen:

Wir benutzen Sharirs Algorithmus, um alle starken Zusammenhangskomponenten zu finden. Dies kostet $\mathcal{O}(|V| + |E|)$. Da $|E| \in \mathcal{O}(|V|^2)$ sind die Kosten damit in $\mathcal{O}(|V| + |V|^2) = \mathcal{O}(|V|^2)$. Für jede starke Zusammenhangskomponente berechnen wir dann die Summe der Gewichte ihrer Kanten. Dies kostet insgesamt $\mathcal{O}(|E|)$ und ist also wiederum in $\mathcal{O}(|V|^2)$. Ist eine dieser Summen negativ, geben wir TRUE aus, sonst FALSE.

Wo liegt der Fehler, den der Student begangen hat?

Tutoraufgabe 2 (Dijkstra Algorithmus):

a) Beweisen oder widerlegen Sie die folgenden Aussagen für einen gewichteten, zusammenhängenden, ungerichteten Graphen G:

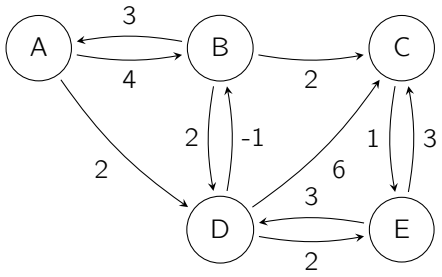
i) Der vom Dijkstra-Algorithmus berechnete SSSP-Baum ist ein Spannbaum.

ii) Der vom Dijkstra-Algorithmus berechnete SSSP-Baum ist ein minimaler Spannbaum.

b) Arbeitet der Dijkstra-Algorithmus immer korrekt, wenn man ihn auf einen Graphen mit negativen Gewichten anwendet, der keine Zyklen mit negativem Gesamtgewicht enthält? Begründen Sie Ihre Antwort!

Tutoraufgabe 3 (Bellman-Ford Algorithmus):

Betrachten Sie den folgenden Graphen:



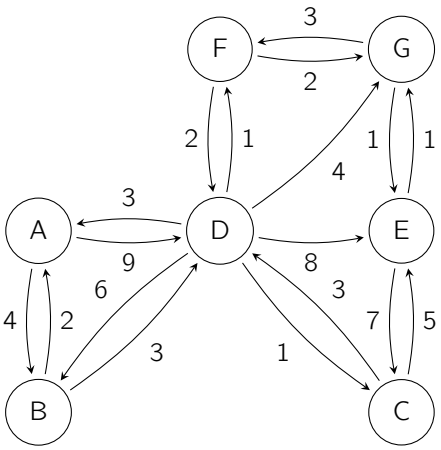
Führen Sie den *Bellman-Ford-Algorithmus* auf diesem Graphen mit dem *Startknoten A* aus. Sie dürfen den Algorithmus vorzeitig abbrechen, wenn sich keine weiteren Änderungen mehr ergeben. Außerdem dürfen Sie Zellen, deren Werte von Zeile zu Zeile gleich bleiben, leer lassen. Füllen Sie dazu die nachfolgende Tabelle aus:

Aktueller Knoten / Entfernung	A	B	C	D	E
–	0	∞	∞	∞	∞

Geben Sie außerdem an, ob der Algorithmus einen Zyklus mit negativem Gesamtgewicht gefunden hat.

Tutoraufgabe 4 (Dijkstra Algorithmus):

Betrachten Sie den folgenden Graphen:

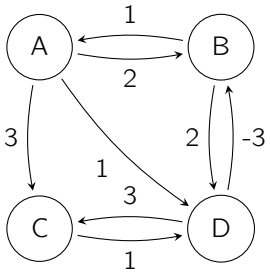


Führen Sie den *Dijkstra* Algorithmus auf diesem Graphen mit dem *Startknoten A* aus. Füllen Sie dazu die nachfolgende Tabelle aus:

Knoten	A					
B						
C						
D						
E						
F						
G						

Aufgabe 5 (Bellman-Ford Algorithmus): (4 Punkte)

Betrachten Sie den folgenden Graphen:



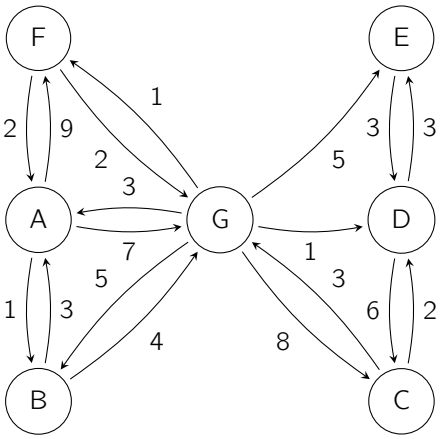
Führen Sie den *Bellman-Ford-Algorithmus* auf diesem Graphen mit dem *Startknoten A* aus. Sie dürfen den Algorithmus vorzeitig abbrechen, wenn sich keine weiteren Änderungen mehr ergeben. Außerdem dürfen Sie Zellen, deren Werte von Zeile zu Zeile gleich bleiben, leer lassen. Füllen Sie dazu die nachfolgende Tabelle aus:

Aktueller Knoten / Entfernung	A	B	C	D
–	0	∞	∞	∞

Geben Sie außerdem an, ob der Algorithmus einen Zyklus mit negativem Gesamtgewicht gefunden hat.

Aufgabe 6 (Dijkstra Algorithmus): (3 Punkte)

Betrachten Sie den folgenden Graphen:



Führen Sie den *Dijkstra* Algorithmus auf diesem Graphen mit dem *Startknoten A* aus. Füllen Sie dazu die nachfolgende Tabelle aus:

Knoten	A					
B						
C						
D						
E						
F						
G						

Aufgabe 7 (Azyklische Graphen): (3 + 5 Punkte)

Sei $G = (V, E, W)$ ein azyklischer, gerichteter, gewichteter Graph.

a) Entwerfen Sie einen Algorithmus, der den längsten Pfad (bezüglich der Gewichte) in G in Zeit $\mathcal{O}(|V| + |E|)$ berechnet.

b) Entwerfen Sie einen Algorithmus, der die Anzahl der Pfade in G in Zeit $\mathcal{O}(|V| + |E|)$ berechnet. Beachten Sie hierbei auch **leere Pfade**.

Aufgabe 8 (Matrjoschka-Kartons): (3 + 3 + 5 Punkte)

Betrachten wir einen k-dimensionalen Karton A mit den Abmessungen $(a_1, \ldots, a_k)$. Wir wollen nun einen Matrjoschka-Karton (ineinander verschachtelte Kartons) aus k-dimensionalen Kartons bauen (siehe `http://de.wikipedia.org/wiki/Matrjoschka`). Ein Karton A passt in einen anderen k-dimensionalen Karton B mit Abmessungen $(b_1, \ldots, b_k)$, wenn eine Permutation π von $\{1, \ldots, k\}$ existiert, sodass

$$a_{\pi(1)} < b_1, a_{\pi(2)} < b_2, \ldots, a_{\pi(k)} < b_k.$$

Passt A in B, so schreiben wir $A \prec B$. (Im Fall von 3 Dimensionen stimmt diese Definition mit der "intuitiven" überein, wobei die Permutation die Möglichkeit des Drehens der Kiste widerspiegelt.)

a) Zeigen Sie, dass wenn $A \prec B$ und $B \prec C$, dann auch $A \prec C$.

b) Entwerfen Sie einen Algorithmus, der in $\mathcal{O}(k^2)$ prüft, ob $A \prec B$.

c) Es seien nun n k-dimensionale Kartons $\{A_1, \ldots, A_n\}$ gegeben. Da wir den größtmöglichen Matrjoschka-Karton (hinsichtlich der Anzahl der Kartons) bauen wollen, sind wir an der maximalen Schachtelungstiefe interessiert. Geben Sie dazu einen Algorithmus an, der die Länge der längsten Folge $\langle A_{i_1}, A_{i_2}, \ldots, A_{i_\ell} \rangle$ von Kartons berechnet, sodass $A_{i_j} \prec A_{i_{j+1}}$ für alle $1 \leq j < \ell$. Die asymptotische Worst-Case-Laufzeit $W(n)$ Ihres Algorithmus soll in $\mathcal{O}(n^i k^j)$ liegen für geeignete Wahlen von $i, j \in \mathbb{N}$. Geben Sie i und j an so dass kein $i' \in \mathbb{N}$ mit $i' < i$ existiert so dass $W(n) \in \mathcal{O}(n^{i'} k^j)$ und so dass kein $j' \in \mathbb{N}$ mit $j' < j$ existiert so dass $W(n) \in \mathcal{O}(n^i k^{j'})$. Begründen Sie, wieso der von Ihnen entwickelte Algorithmus diese Laufzeitschranke einhält.

Aufgabe 9 ($\mathcal{O}$-Notation): (5 Minuten, 3 Punkte)

Beweisen oder widerlegen Sie:

$$\Pi_{i=1}^{n}(i + n) \in \mathcal{O}(n^{2n})$$

Aufgabe 10 (Rekursionsgleichungen): (10 Minuten, 2 + 2 + 2 = 6 Punkte)

a) Geben Sie für das Programm

```
int max(int a[], int low, int high) {
    if (low >= high) {
        return a[low];
    } else {
        int mid = (low + high) / 2;
        int leftmax = max(a, low, mid);
        int rightmax = max(a, mid + 1, high);
        if (leftmax > rightmax) {
            return leftmax;
        } else {
            return rightmax;
        }
    }
}
```

eine Rekursionsgleichung für die asymptotische Laufzeit des Aufrufes `max(a, 0, n - 1)` an, wobei Sie annehmen dürfen, dass das Array `a` die Länge `n` hat. Die elementaren, also die für die asymptotische Laufzeit relevanten, Operationen sind alle arithmetischen Operationen sowie Vergleiche. Sie brauchen die Basisfälle der Rekursionsgleichung *nicht* anzugeben.

Lösen Sie anschließend die Rekursionsgleichung, indem Sie die asymptotische Komplexitätsklasse (Θ) in geschlossener Form angeben.

b) Geben Sie für das Programm

```
int rev(int n, int m) {
    if (n == 0) {
        return m;
    } else {
        return rev(n - 1, n + 1 + m);
    }
}
```

eine Rekursionsgleichung für die asymptotische Laufzeit des Aufrufes `rev(n, m)` an. Die elementaren, also die für die asymptotische Laufzeit relevanten, Operationen sind alle arithmetischen Operationen sowie Vergleiche. Sie brauchen die Basisfälle der Rekursionsgleichung *nicht* anzugeben.

Lösen Sie anschließend die Rekursionsgleichung, indem Sie die asymptotische Komplexitätsklasse (Θ) in geschlossener Form angeben.

c) Bestimmen Sie für die Rekursionsgleichung

$$T(n) = 9 \cdot T\left(\frac{n}{3}\right) + n^2 + \sqrt{n} + 3 \cdot n + 3$$

die Komplexitätsklasse Θ mit Hilfe des Master-Theorems. Begründen Sie Ihre Antwort.