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?

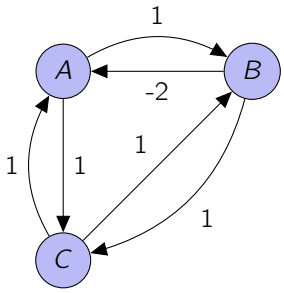
Lösung:

a) Der folgende Algorithmus gibt einen kürzesten Pfad von i nach j aus:

```
void bellFord(List adjLst[n], int n, int i, int j){
    int d[n] = +inf;
    d[i] = 0;

    int parent[n] = -1;

    for (int k = 1; k < n; k++) // n-1 Durchläufe
        for (int v = 0; v < n; v++) // alle Kanten
            foreach (edge in adjLst[v])
                if (d[edge.w] > d[v] + edge.weight) {
                    d[edge.w] = d[v] + edge.weight;
                    parent[edge.w] = v;
                }
    ausgabe("Ein kürzester Weg mit Länge " + d[j] + ":");
    printPfad(parent, j);
}

// printPfad gibt den Pfad rekursiv aus
void printPfad(int parent[n], int node){
    // gibt es weiter Vorgänger, so wird zuerst der Pfad bis zu diesem Knoten ausgegeben
    if(parent[node] != -1){
        printPfad(parent, parent[node]);
        ausgabe(" -> ");
    }
    // Zuletzt wird der Knoten selbst ausgegeben
```

```
        ausgabe(node);
    }
```

b) Eine starke Zusammenhangskomponente ist nicht dasselbe wie ein Zyklus. Betrachten wir den folgenden Graphen G_3:



Die einzige starke Zusammenhangskomponente ist der gesamte (vollständige) Graph. Ihr Gesamtgewicht ist 3. Somit würde das Verfahren des Studenten FALSE ausgeben. Es gibt jedoch einen Zyklus mit negativem Gewicht von A nach B und zurück.

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.

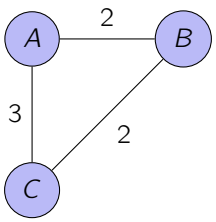
ii) Der vom Dijkstra-Algorithmus berechnete SSSP-Baum ist ein minimaler Spannbaum.

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!

Lösung:

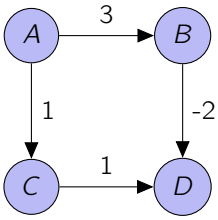
a) i) Da jede hinzugefügte Kante zu einem Knoten führt, der bisher nicht zum Baum gehört hat, kann kein Zyklus entstehen. Da außerdem jede dieser Kanten von einem Knoten ausgeht, der zum bisherigen Baum gehört, ist dieser zusammenhängend. Es handelt sich also tatsächlich um einen Baum. Da der Algorithmus erst terminiert, wenn es keine Randknoten mehr gibt, und der Graph zusammenhängend ist, werden alle Knoten hinzugefügt. Damit ist der SSSP-Baum ein Spannbaum. Die Aussage ist damit gezeigt.

ii) Betrachten wir folgenden Graphen G_4:



Der minimale Spannbaum ist $(\{A, B, C\}, \{\{A, B\}, \{B, C\}\})$ mit dem Gewicht 4. Der SSSP-Baum für den Startknoten A ist hingegen $(\{A, B, C\}, \{\{A, B\}, \{A, C\}\})$ mit dem Gewicht 5. Da $5 > 4$ gilt, ist der SSSP-Baum nicht minimal und die Aussage ist widerlegt.

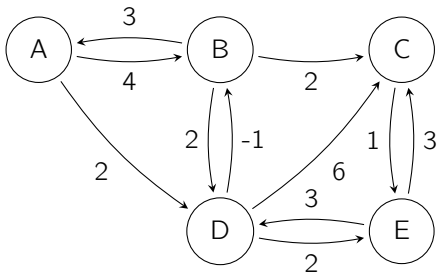
b) Betrachten wir den folgenden Graphen G_5:



Vom Startknoten A aus berechnet der Dijkstra Algorithmus die kürzeste Distanz von A zu D als 2. Dabei ist die tatsächliche kürzeste Distanz 1. Also arbeitet der Algorithmus nicht immer korrekt, wenn er auf Graphen mit negativen Gewichten angewendet wird.

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:

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

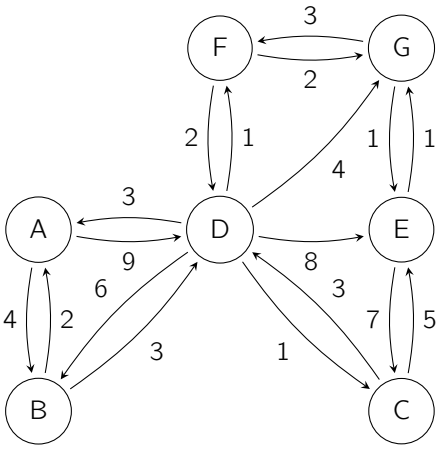
Lösung:

Aktueller Knoten / Entfernung	A	B	C	D	E
–	0	∞	∞	∞	∞
A		4		2	
B			6		
C					7
D		1			4
E					
A					
B			3		

Der Algorithmus hat keinen Zyklus mit negativem Gesamtgewicht gefunden.

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:

Lösung:

Knoten	A	B	D	C	F	G
B	4	–	–	–	–	–
C	∞	∞	8	–	–	–
D	9	7	–	–	–	–
E	∞	∞	15	13	13	11
F	∞	∞	8	8	–	–
G	∞	∞	11	11	10	–

Die grau unterlegten Zellen markieren, an welcher Stelle für welchen Knoten die minimale Distanz sicher berechnet worden ist.