DAP2 – Heimübung 13

Ausgabedatum: 7. 7. 17 — Abgabedatum: Fr. 14. 7. 17 (Mo. 17. 7. für Gruppen 27–32) 12 Uhr

Schreiben Sie unbedingt immer Ihren **vollständigen Namen**, **Ihre Matrikelnummer** und **Ihre Gruppennummer** auf Ihre Abgaben!

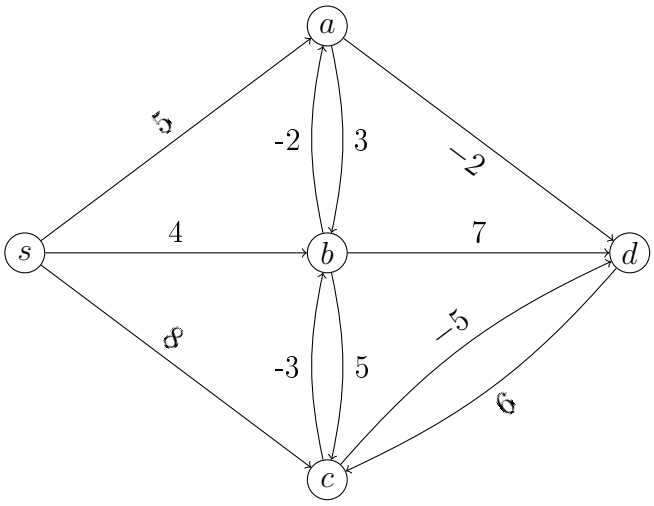
Aufgabe 13.1 (5 Punkte): (Negative Kanten und Bellman-Ford)

a) (2 Punkte) Bob schlägt für gerichtete Graphen $G = (V, E)$ mit Gewichtsfunktion w, die negative Kantengewichte, aber keine negativen Zyklen haben, die folgende Methode zur Berechnung kürzester Wege von einem Startknoten s vor:

- Wähle eine Konstante c so, dass für alle Kanten $e \in E$ gilt: $c + w(e) > 0$.
- Wähle als neue Gewichtsfunktion $w' : E \to \mathbb{R}$ mit $w'(e) = c + w(e)$.
- Berechne mit Hilfe des Dijkstra-Algorithmus die kürzesten Wege von s für den Graphen G mit der Gewichtsfunktion w'.

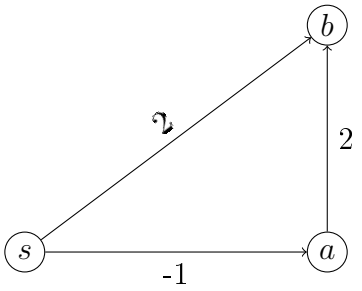
Warum stellt Bobs Methode kein korrektes Verfahren dar? Geben Sie ein Gegenbeispiel an und erläutern Sie dieses.

b) (3 Punkte) Führen Sie den Algorithmus von Bellman-Ford auf dem folgenden gerichteten Graphen $G = (V, E)$ aus. Als Startknoten dient der Knoten s. Geben Sie dabei nach der Initialisierung und nach jedem Durchlauf i der äußeren *for*-Schleife für jeden Knoten $v \in V$ den Wert $M[i][j]$ an. Ermitteln Sie mit Hilfe einer aus der Vorlesung bekannten Methode, ob der gegebene Graph G negative Zyklen enthält. Erläutern Sie die dazu nötigen zusätzlichen Schritte und Ihre Antwort gesondert.



Lösung:

a) Betrachte hierzu den folgenden gerichteten Graphen:



Der kürzeste Weg von s nach b geht über a und hat das Gewicht 1. Nach Bobs Methode würde zu jedem Kantengewicht ein Wert $c = 1 + e > 1$ für $e > 0$ addiert werden. Damit würde die Kante (s, b) das Gewicht $2 + e$ haben, der Weg (s, a, b) hätte dann das Gewicht $(-1 + 1 + e) + (2 + 1 + e) = 3 + 2e$. Der Dijkstra-Algorithmus würde somit den Weg (s, b) als kürzesten Weg von s nach b berechnen.

b) Wir erstellen die Tabelle mit den Werten für $M[i][v]$:

	s	a	b	c	d
0	0	∞	∞	∞	∞
1	0	5	4	8	∞
2	0	2	4	8	3
3	0	2	4	8	0
4	0	2	4	6	0
5	0	2	3	6	0

Es gilt:

G hat keinen negativen Zyklus genau dann, wenn $M[|V|][v] = M[|V| - 1][v]$ für alle Knoten $v \in V$. Da für den gegebenen Graphen G $M[4][v] \neq M[5][v]$ für Knoten $v \in \{b\}$ gilt, folgt, dass G einen negativen Zyklus besitzt.

Aufgabe 13.2 (5 Punkte): (Graphenalgorithmen)

Der Biologe Bob beobachtet und klassifiziert die Schmetterlinge, die einem der zwei Typen A und B gehört. Sei die Menge der beobachteten Schmetterlinge mit V bezeichnet. Dabei hat er seine Beobachtungen in zwei Mengen E und F gesammelt, so dass für jedes Paar der Schmetterlinge $(u, v) \in V \times V$ genau eine der folgenden Möglichkeiten gilt:

- das Paar (u, v) ist in E, was bedeutet dass u und v von gleichem Typ sind;
- das Paar (u, v) ist in F, was bedeutet dass u und v von unterschiedlichem Typ sind;
- das Paar (u, v) ist weder in E noch in F.

Jeder Schmetterling wurde mindestens einmal mit einem anderen Schmetterling verglichen (d. h. es wurde beobachtet, ob das Knotenpaar in E oder F liegt). Bob möchte jetzt wissen, ob seine Beobachtungen konsistent sind, d. h., ob es keine widersprüchlichen Aussagen über die Klassifizierung der Schmetterlinge in zwei Gruppen gibt.

a) Entwerfen Sie einen Algorithmus `Konsistent`(V, E, F), der entscheidet, ob die Menge der Beobachtungen konsistent ist und dementsprechend `TRUE` bzw. `FALSE` zurückgibt. Beschreiben Sie den Algorithmus zunächst mit eigenen Worten. Setzen Sie den Algorithmus dann in Pseudocode um. Für die volle Punktzahl wird ein Algorithmus erwartet, dessen Laufzeit durch $\mathcal{O}(|V| + |E| + |F|)$ beschränkt ist.

b) Analysieren Sie die Laufzeit Ihres Algorithmus.

c) Beweisen Sie die Korrektheit Ihres Algorithmus.

Lösung:

a) Wir betrachten die Schmetterlinge als die Knoten eines Graphens, und die Menge der Beobachtungen (E bzw. F) als die Kanten. Wir passen für Testen aller Kanten den Tiefensuche-Algorithmus an, sodass für alle Knoten $u \in V$ die Adjazenzknoten v noch einen anderen Parameter haben, den wir als $Typ[v]$ bezeichnen, und zwar mit dem Wert 1 für gleiche Farbe wie u (E-Kanten), und mit dem Wert 2 für unterschiedliche Farbe als u (F-Kanten).

Die Knoten werden drei möglichen Farben haben: weiß, solange sie noch nicht entdeckt wurden, blau und rot (die für die entdeckten Knoten genutzt werden, und zwar konsistent aber abhängig vom Typ der Kante die wir zurzeit betrachten). Es wird `FALSE` zurückgegeben, wenn es ein Paar der Knoten über einer Kante entdeckt wird, sodass die Kante vom Typ 1 ist, und die Knoten sind rot und blau, bzw. die Kante vom Typ 2 ist, und die beiden Knoten sind gleichgefärbt (in blau oder rot).

Der Algorithmus `Konsistent` spielt die Rolle des Algorithmus `DFS` (aus der Vorlesung), während der Algorithmus `SchmetterlingDFS-Visit` ist der angepasste `DFS-Visit` Algorithmus.

Konsistent(V, E, F):

```
1  for u ∈ V do
2      Adj[u] ← nullptr                          /* Gewichtete Adjazenzliste erstellen */
3  for (u, v) ∈ E do
4      Adj[u] ← Adj[u] ∪ (v, 1), Adj[v] ← Adj[v] ∪ (u, 1)
5  for (u, v) ∈ F do
6      Adj[u] ← Adj[u] ∪ (v, 2), Adj[v] ← Adj[v] ∪ (u, 2)
7  foreach u ∈ V do
8      color[u] ← weiß                                   /* DFS Initialisieren */
9  foreach u ∈ V do
10     if color[u] = weiß then
11         if SchmetterlingDFS-Visit((V, Adj), u, blau)=FALSE then
12             return FALSE
13 return TRUE
```

Die rekursive Teilaufrufe erfolgen durch folgenden Algorithmus. Die Farbe c ist die Farbe die als nächste zum Nutzen ist (am Anfang sei sie blau).

SchmetterlingDFS-Visit(Graph (V, Adj), Knoten u, Farbe c):

```
1  color[u] ← c
2  foreach v ∈ Adj[u] do
3      if color[v] = weiß then
                                                  /* Erster Besuch eines Knotens */
4          if Typ[v] = 1 then
5              Antwort← SchmetterlingDFS-Visit((V, Adj), v, c)
6          else
7              if c = blau then
8                  Antwort← SchmetterlingDFS-Visit((V, Adj), v, rot)
9              else
10                 Antwort← SchmetterlingDFS-Visit((V, Adj), v, blau)
11         if Antwort=FALSE then
12             return FALSE
13     else
                                                  /* Wiederholter Besuch eines Knotens */
14         if (Typ[v] = 1 and color[v] ≠ c) or (Typ[v] = 2 and color[v] = c) then
15             return FALSE
16 return TRUE
```

b) Der Algorithmus `Konsistent` benötigt die Laufzeit $\mathcal{O}(|V|+|E|+|F|)$ für die Erstellung der Adjazenzliste (Zeilen 1 bis 6), sowie die Initialisierung der Farben der Knoten (Zeilen 7 und 8). Die Schleife in den Zeilen 9 bis 12 wird einmal auf jeder Zusammenhangskomponente aufgerufen, und sie ruft den Algorithmus `SchmetterlingDFS-Visit` auf.

Der Algorithmus `SchmetterlingDFS-Visit` wird jede Kante der Zusammenhangskomponente genau einmal besuchen, und für diesen Besuch die konstante Zeit benötigen. Deswegen braucht die Schleife in den Zeilen 9-12 des Hauptalgorithmus insgesamt $\mathcal{O}(|V| + |E| + |F|)$ Zeit. Die Rückgabe benötigt konstante Zeit, somit braucht der gesamte Ablauf $\mathcal{O}(|V| + |E| + |F|)$ Zeit.

c) Wir sollen die Korrektheit der Rückgabe zeigen, d.h. dass es `TRUE` genau dann zurückgegeben wird, wenn die Eingabe konsistent ist. Da wir die Tiefensuche angepasst haben, wird ganzer Graph besucht und jede Kante genau einmal genutzt, und jeder Knoten genau einmal von weiß in blau oder rot gefärbt. Die Korrektheit dieses Prozesses folgt aus der Korrektheit der Tiefensuche.

`TRUE` Nehmen wir an, es wird `FALSE` zurückgegeben, obwohl die Eingabe konsistent ist. D.h. es wird beim Aufruf `SchmetterlingDFS-Visit` bei einer Zusammenhangskomponente startend mit Knoten u `FALSE` zurückgegeben.

Die Rekursion bricht ab, entweder wenn man in den Zeilen 14 und 15 die Bedingung erfüllt ist und `FALSE` zurückgegeben wird, oder wenn alle Möglichkeiten ausprobiert sind, und wird `TRUE` zurückgegeben. In diesem Fall heißt es, dass für die Kante (u, v), der Knoten u hat die Farbe c (blau oder rot), und die Bedingung "($Typ[v] = 1$ and color$[v] \neq c$) or ($Typ[v] = 2$ and color$[v] = c$)" wahr ist.

Falls $Typ[v] = 1$ ist, bedeutet das dass u und v gleiche Farbe haben, d.h. color$[v] = c$. Dies ist Widerspruch mit color$[v] \neq c$. Analog, falls $Typ[v] = 2$ ist, bedeutet das dass u und v unterschiedliche Farben haben, d.h. falls u ist rot, v muss blau sein (und umgekehrt). Dies widerspricht color$[u] = c$ und color$[v] = c$.

`FALSE` Nehmen wir an, es wird fälschlicherweise `TRUE` zurückgegeben (d.h. nie vorher `FALSE` gegeben), obwohl es einen Knoten w gibt, der in Adjazenzlisten von Knoten u und v liegt, und der ihretwegen in zwei unterschiedlichen Farben gefärbt werden sollte. Da wir die Tiefensuche angepasst haben und dadurch "immer weiter" im Graphen gesucht haben, muss es um einen schon gefärbten Knoten zu treffen, einen Kreis $u_1, u_2, \ldots, u_k$ geben, bei dem zuerst der Knoten u_1 besucht wird und jeder Knoten u_i wird von seinem Vorgänger u_{i-1} in der Tiefensuche entdeckt. Sei u_1 blau durch Aufruf `SchmetterlingDFS-Visit`$((V, Adj), u_1,$ `blau`$)$.

Falls u_k ist gerade (im rekursiven Teilaufruf `SchmetterlingDFS-Visit`$((V, Adj), u_{k-1}, c)$, wo c von der Kante (u_{k-1}, u_k) abhängt) rot geworden und $(u_k, u_1) \in E$ (Typ 1), wird bei der For-Schleife (Zeile 2) für $u_1 \in Adj[u_k]$ gesehen, dass color$[u_1] \neq$ weiß, und dann in der Zeile 14 ist $Typ[u_1] = 1$ und color$[u_1] \neq$color$[u_k]$ wahr. Da wird `FALSE`, und rekursiv durch alle Aufrufe `FALSE` zurückgegeben. Widerspruch.

Analog, falls u_k ist gerade blau geworden, und die Kante $(u_k, u_1) \in F$ (Typ 2). Dann ist wieder $u_1 \in Adj[u_k]$ getroffen, color$[u_1] \neq$ weiß, und dann in der Zeile 14 ist $Typ[u_1] = 2$ und color$[u_1] =$color$[u_k]$ wahr. Da wird `FALSE`, und rekursiv durch alle Aufrufe `FALSE` zurückgegeben. Widerspruch.

Aufgabe 13.3 (5 Punkte): (**Bonusaufgabe:** Dynamische Programmierung)

Eine Laufbahn ist als ein zweidimensionales Feld mit $m \times n$ Blöcken dargestellt. In jedem Block des Feldes werden Strafpunkte angegeben: Im Block (i, j) sind $S(i, j)$ Strafpunkte für die Läufer anzurechnen, wobei $S(i, j) \geq 0$ gilt.

Emma möchte die Laufbahn durchlaufen, sodass sie die minimale Anzahl an Strafpunkten sammelt. Sie fängt in der ersten Spalte ($j = 1$) in einer beliebigen Zeile ($1 \leq i \leq m$) an und beendet ihren Lauf in der letzten Spalte ($j = n$). Die Regeln schreiben vor, dass sie von Block (i, j) ausgehend nur solche Blöcke betreten darf, die in der Spalte rechts von ihrem Block liegen und über eine Ecke oder über eine Seite an Block (i, j) angrenzen, d. h., die Blöcke, die oben rechts, rechts oder unten rechts von $(i.j)$ liegen.

Ein Beispiel ist unten gegeben. Von dem Block mit Punktewert 2 in Spalte $j = 1$ und Zeile 2 aus kann man nur die Werte 1 oder 2 (Zeilen 1, 2 oder 3 in Spalte 2) erreichen. Emma sollte die hervorgehobenen Blöcke besuchen, um so wenige Strafpunkte wie möglich zu sammeln. Die Pfeile kennzeichnen jeweils den Ein- und Ausgangsblock.

	$j = 1$			$j = n$	
	8	1	0	3	
$\rightarrow$	**2**	**1**	7	3	
	4	2	**1**	**1**	$\rightarrow$
	3	3	1	4	

a) Geben Sie eine Rekursionsgleichung für die minimale Anzahl der Strafpunkte $E(i,j)$ an, die Emma in ihrem Lauf bis einschließlich Block (i,j) gesammelt haben muss.

b) Geben Sie einen Algorithmus in Pseudocode an, der auf dem Prinzip der dynamischen Programmierung beruht und die minimale Anzahl der von Emma gesammelten Punkte zurückgibt.

c) Analysieren Sie die Laufzeit Ihres Algorithmus.

d) Beweisen Sie die Korrektheit Ihrer Rekursionsgleichung aus Teilaufgabe a) mittels Induktion.

Lösung:

a) In ersten Spalte ($i = 1$) kann Emma nur die Strafpunkte haben, die in diesem Feld gegeben sind. Da Emma aus dem Feld $[i,j]$ nur die Felder $[i-1,j+1]$, $[i,j+1]$ und $[i+1,j+1]$ erreichen kann, bedeutet dies dass sie zum Feld $[i,j]$ nur von den Felder $[i-1,j-1]$, $[i,j-1]$ und $[i+1,j-1]$ kommen kann, solange diese Felder existieren (und Emma nicht am Rand der Laufbahn ist). Um Fallunterscheidung zu vermeiden, defininieren wir die Zeilen mit $i = 0$ und $i = m+1$, die den unmöglichen Fall representieren, und dementsprechend unendlich größen Wert enthalten. Deswegen haben wir folgenden rekursiven Form:

$$E[i,j] = \begin{cases} +\infty & \text{falls } i = 0 \text{ oder } i = m+1, \forall j \geq 1 \\ S[i,1] & \text{falls } j = 1 \\ S[i,j] + \min\{\begin{matrix} E[i-1,j-1] \\ E[i,j-1] \\ E[i+1,j-1] \end{matrix}\} & \text{falls } j > 1 \end{cases}$$

b) In Pseudocode erhalten wir folgenden Algorithmus. Wir geben den minimalen Wert in der Spalte $j = n$ zurück, da Emma in beliebiger Reihe ihren Lauf beenden kann.

```
Lauf (Array  S[1..m][1..n]):
1.  E ← new Array [0..m+1][1..n]
2.  for j =← 1 to n do
3.      E[0,j] =← ∞
4.      E[m+1,j] =← ∞
5.  for i =← 1 to m do
6.      E[i,1] =← S[i,1]
7.  for j ← 2 to m do
8.      for i ← 1 to n do
9.          E[i,j] ← S[i,j] + min{E[i-1,j-1], E[i,j-1], E[i+1,j-1]}
10. return min_{i∈{1..m}}{E[i,n]}
```

c) Die Laufzeit $T(m,n)$ des Algorithmus ist $O(mn)$, weil wir die Zeit $O(mn)$ für die Initialisierung des zweidimensionalen Arrays in der Zeile 1 brauchen, sowie für die zwei verschachtelten Schleifen in Zeilen 7-9. Die For-Schleife in den Zeilen 5-6 benötigt $O(m)$ Zeit. Die For-Schleife in den Zeilen 2-4, sowie Minimumberechnung für die Rückgabe in der Zeile 10 benötigen $O(m)$ Zeit.

d) Behauptung: $E[i,j]$ enthält minimale Strafpunktesumme, um das Feld $[i,j]$ zu erreichen, für alle $j \geq 1$ und alle $0 \leq i \leq m+1$.

I.A. Sei $j = 1$. Da dies das erste Feld ist, ist die einzige Möglichkeit auf diesem Feld den Lauf anzufangen. Dann ist für alle $1 \leq i \leq m$ (erlaubte Felder) die einzige – minimale Summe $E[i,1] = S[i,1]$. Wenn $i = 0$ bzw. $i = m+1$ (unmögliche Fälle) ist die minimale Summe $E[i,1] = +\infty$.

I.V. Seien $E[i,j-1]$ minimal für beliebiges aber festes j und alle $0 \leq i \leq m+1$.

I.S. Nehmen wir an, $E[i,j]$ ist für bestimmte i nicht optimal, mit $1 \leq i \leq m$ (da für $i = 0$ und $i = m+1$ es sich um unmöglichen Fälle handelt, mit $E[i,j] = +\infty$). Dann gäbe es eine $v < E[i,j] = S[i,j] + \min\{E[i-1,j-1], E[i,j-1], E[i+1,j-1]\}$, sodass Emma mit v Strafpunkte das Feld $[i,j]$ erreichen kann. Da in der Lösung v das Feld $[i,j]$ nur aus den Felder $[i-1,j-1]$, $[i,j-1]$ und $[i+1,j-1]$ erreicht werden kann, nehmen wir an, sei es $[i-1,j-1]$ gewesen (die andere zwei sind äquivalent). Es gilt dann $v < S[i,j] + E[i-1,j-1] \Leftrightarrow v - S[i,j] < E[i-1,j-1]$ und $v - S[i,j]$ wäre die Anzahl der Punkte, damit sie das Feld $[i-1,j-1]$ erreichen könnte. Das widerspricht der Induktionsvoraussetzung, weil $E[i-1,j-1]$ die minimale Punktesumme bis zum Feld $[i-1,j-1]$ ist.