DAP2 UB12

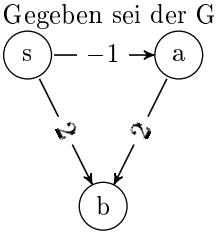
Anja Rey, Gr.23 , Briefkasten 22

Max Springenberg, 177792

13.1 Negative Kanten und Bellman-Ford

13.1.1

Gegeben sei der Graph G:



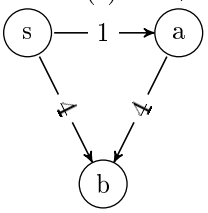
Wir suchen nach dem kuerzestem Weg von s nach b.

Nach Bobs Verfahren waehlen wir also nun ein c, so dass

$$\forall e \in E.c + w(e) > 0, c \in \mathbb{N}$$

,mit E als der Menge aller Kanten aus G, gilt.

Ein solches c waere hier beispielsweise c=2. Daraus wuerde dann ueber die veraenderte Gewichtsfunktion $w'(e) = c + w(e)$, der folgenede veraenderte Graph G' resultieren.



Es ist ersichtlich, dass der kuerzeste Weg aus G von s ueber a nach b mit den Kanten $K = \{(s,a),(a,b)\}$ verlaueft.
Wuerde man nun aber den Dijkstra-Algorithmus auf G' anwenden waere die kuerzeste Verbindung von s nach b mit den Kanten $K' = \{(s,b)\}$. Es gilt $K \neq K'$ damit ist Bobs Verfahren nicht korrekt.

13.1.2

Bellman-Ford Algorithmus:
d_i steht je fuer den i-ten durchlauf.

d_i	s	a	b	c	d
d_0	0	∞	∞	∞	∞
d_1	0	5	4	8	∞
d_2	0	2	4	8	3
d_3	0	2	4	8	0
d_4	0	2	4	6	0

Negative Zyklen:
ein Graph enthaelt beispielsweise dann einen negativen Zyklus, wenn sich nach $n = |V| - 1$ Iterationen noch ein Wert in M veraendert.

d_i	s	a	b	c	d
d_4	0	2	4	6	0
d_5	0	2	3	6	0

Aufgrund eines Weges ueber die Kanten $\{(a,d),(d,c),(c,b),(b,a)\}$ veraendern sich auch nach n Iterationen noch Werte. Damit existiert ein negativer Zyklus.

13.2 Graphalgorithmen

gegeben:

(i) Menge von Schmetterlingen V, der typisierten Mengen A, B
(ii) Ferner definieren wir eine Menge $T \subseteq V \times V$ von Tupeln von Schmetterlingen,
die der selben typisierten Menge entstammen.
(iii) Teilmengen $E \subseteq V, F \subseteq V$ mit den Regeln:
1. $(u, v) \in E \Rightarrow (u, v) \in T$
2. $(u, v) \in F \Rightarrow (u, v) \notin T$
3. $\forall (u, v) \in V \times V.((u, v) \notin E \wedge (u, v) \notin F) \vee ((u, v) \in E \Leftrightarrow (u, v) \notin F)$

13.2.1

Ziel des Algorithmus soll sein, auszugeben, ob die Regeln aus (iii) auf Mengen V, E, F gelten.
Fuer eine wahre Ausgabe muss gewaehrleistet sein, dass:
alle Schmetterlingpaare aus E den selben Typ haben. (iii) 1.
alle Schmetterlingpaare aus F nicht den selben Typ haben. (iii) 2.
Schmetterlingpaare duerfen in keiner beider Mengen sein, aber nicht in beiden (iii) 3.
Nach (iii) 3. reicht es Paare aus E und F zu betrachten.

Wir nehmen an, das die Knoten alle als ids, bzw. als int Wert / ganzahlige Zahl zur Verfuegung stehen und man deshalb Arrays anhand von ihnen befuellen kann. Weiter nehmen wir an, dass die Methode type(x) den Typen des Knoten x in $O(1)$ ausgibt. Der Typ sei mit anderen Typen vergleichbar.
Zu Beginn initialisieren wir einen Array dessen Laenge gleich der Anzahl von Knoten des Graphen ist. In diesen Array wird gespeichert, ob ein Knoten markiert ist oder nicht. Zu Beginn sind alle Knoten unmarkiert.
Zunaechst wird durch E gierig durchgelaufen und geguckt, ob das jeweilige Paar vom gleichen Typen ist. Ist dies der Fall, so wird jeder Knoten des Paares makiert um zu signalisieren, dass er in E enthalten ist.
Danach wird durch F gierig durchgelaufen und geguckt, ob das jeweilige Paar nicht vom gleichen typen ist und nicht beide Knoten des Paares markiert wurden, bzw. in E enthalten sind.

```
KONSISTENT(V, E, F):
1       colors ←new Array[1...|V|]
2       for i ←1 to |V| do
3           colors[i] ←white
4       for each (s_0, s_1) ∈ E do
5           if type(s_0) = type(s_1) then
6               colors[s_0] ←black
7               colors[s_1] ←black
8           else
9               return False
10      for each (s_0, s_1) ∈ F do
11          if (colors[s_0] = black and colors[s_1] = black) or (type(s_0) = type(s_1)) then
12              return False
13      return True
```

13.2.2

Es folgen die Operationen je nach Zeile fuer die O-Notation:

1	$\in O(1)$
2-3	$\in O(1 + \|V\|) = O(\|V\|)$
4-9	$\in O(1 + 3 * \|E\|) = O(\|E\|)$
10-12	$\in O(1 + 2 * \|F\|) = O(\|F\|)$
13	$\in O(1)$

Die Zeilen 2-3, 4-9, 10-12 laufen in $O(|V|)$, $O(|E|)$ und $O(|F|)$ da fuer sie jeweils die Mengen einmal gierig durchgegangen werden muesen.
Die Zeilen 1, 13 sind nicht an Schleifen gebunden und in Konstanter Laufzeit.
Fuer die gesamte Laufzeit in O-Notation ergibt sich:
$O(1) + O(|V|) + O(|E|) + O(|F|) = O(1 + |V| + |E| + |F|) = O(|V| + |E| + |F|)$

Damit liegt unserer Algorithmus in der gewuenschten Laufzeit $O(|V| + |E| + |F|)$.

13.2.3

Wie bereits erwaehnt muessen folgende Regeln gelten:

(1) alle Schmetterlingpaare aus E den selben Typ haben.
(2) alle Schmetterlingpaare aus F nicht den selben Typ haben.
(3) Schmetterlingpaare duerfen in keiner beider Mengen sein, aber nicht in beiden.

Wir nehmen an ein Paar (s_0, s_1) existiert, fuer dass eine der 3 Regeln, nach dem Ablauf unseres Algorithmus und einer Ausgabe True oder False, nicht gilt.
Wir betrachten jede moegliche Verletzung einer Regel einzelnt.

I) Es gilt nicht: (1) alle Schmetterlingpaare aus E den selben Typ haben.
In diesem Fall sei also (s_0, s_1) in E und nicht vom selben Typ und es wird True ausgegeben.
Dies steht im Widerspruch zu den Zeilen 8-9, die fuer den Fall, dass ein Paar aus E nicht den selben Typ hat direkt False ausgeben.

II) Es gilt nicht: (2) alle Schmetterlingpaare aus F nicht den selben Typ haben.
In diesem Fall sei also (s_0, s_1) in F und vom selben Typ und es wird True ausgegeben.
Dies steht im Widerspruch zu den Zeilen 11-12, die fuer den Fall, dass ein Paar aus F den selben Typ hat direkt False ausgeben.

III) Es gilt nicht: (3) Schmetterlingpaare duerfen in keiner beider Mengen sein, aber nicht in beiden
In diesem Fall sei also (s_0, s_1) in E und F und es wird True ausgegeben.
Dies steht im Widerspruch zu den Zeilen 5-7 und 11-12, die dafuer zustandig sind alle Schmetterlinge in E zu makieren und nachdem alle Schmetterlinge aus E makiert wurden in F ueberpruefen, ob beie Schmetterlinge des Paares makiert wurden. Nur dann waere auch das Paar in E. In diesem Fall wird direkt False ausgegeben.

Da Keiner der Faelle moeglich ist, gilt die Annahme nicht und durch Kontraposition wurde gezeigt, dass der Algorithmus korrekte Ausgaben gibt.

13.3 Bonus: Dynamische Programmierung

13.3.1

13.3.2

```
RUN(S):
    E ←new Array[0...length(S)][0...lengthS[0]]
    for i ←0 to length(E) do
        E[i][0] ←∞
    for j ←1 to length(E[0]) do
        E[0][j] ←0
    for j ←1 to length(E) do
        E[1][j] ←E[1][j-1] + S[1][j]
    for i ←2 to length(E) do
        for j ←1 to length(E[0]) do
            E[i][j] ←min{S(i − 1, j), S(i, j)} + E[i][j − 1]
    result ←∞
    for i ←1 to length(E) do
        result ←min{result, E[i][length(E[0])]}
    return result
```