Lösungen von Übungsblatt 5

Algorithmen (WS 2018, Ulrike von Luxburg)

Lösungen zu Aufgabe 1

a) Bezeichne G den gemäß Aufgabenstellung konstruierten Graphen.

```
1:  function AttackMCP(G)
2:      (V_1, ..., V_N) ← SCC(G)        ▷ Finde alle starken Zusammenhangskomponenten von G.
3:      E ← ∅        ▷ Initialisiere eine leere Kantenmenge für "Meta-Kanten" zwischen den V_i's.
4:      for i, j = 1 ... N do
5:          if ∃u ∈ V_i, ∃v ∈ V_j : u → v then      ▷ Wenn irgendeine Kante aus V_i nach V_j führt.
6:              E ← E ∪ {V_i → V_j}                 ▷ Füge E eine Meta-Kante von V_i nach V_j hinzu.
7:          end if
8:      end for
9:      I ← ∅                                       ▷ (minimale) Menge zu infiltrierender Module.
10:     for i = 1 ... N do
11:         if InDegree(E, V_i) = 0 then            ▷ Wenn keine Meta-Kante nach V_i führt.
12:             I ← I ∪ ChooseAny(V_i)              ▷ Füge I irgendeinen Knoten aus V_i hinzu.
13:         end if
14:     end for
15:     InfiltrateAndEliminate(I)        ▷ Infiltriere die Module aus I und eliminiere MCP.
16: end function
```

Die Zeilen 1-8 konstruieren einen neuen "Meta-Graphen" $\mathcal{G} = (\{V_1, \ldots, V_n\}, \mathcal{E})$, dessen Knoten die starken Zusammenhangskomponenten von G sind. Eine Kante $V_i \to V_j$ wird erstellt, wenn irgendein Knoten in V_j direkt abhängig von irgendeinem Knoten in V_i ist (und somit *jeder* Knoten in V_j direkt oder indirekt abhängig von *jedem* Knoten in V_i ist). Die Zeilen 9-14 bestimmen nun all diejenigen Zusammenhangskomponenten, die unabhängig von allen anderen sind, und wählt aus diesen jeweils ein beliebiges Modul aus. Zeile 15 führt schließlich die Infiltrierung aller Module aus I aus, sowie die abschließende Eliminierung des MCP.

b) Jedes Modul $m \in V$ liegt in irgendeiner Zusammenhangskomponente V_i. Falls V_i in $\mathcal{G}$ keine eingehende Kante hat, so wird in Zeilen 11-12 irgendein Modul aus V_i zur Infiltrierung ausgewählt, wodurch aufgrund des starken Zusammenhangs in V_i auch m direkt oder indirekt eliminiert wird. Falls hingegen $V_k \to V_i$ existiert, so wird m direkt oder indirekt von allen Modulen aus V_k heraus erreicht. Induktiv lässt sich dies bis auf eine unabhängige Zusammenhangskomponente zurückführen, welche wieder dem ersten Fall genügt und infiltriert ist. Also wird letztlich *jedes* Modul aus G eliminiert.

c) Um alle Module auszuschalten, *muss* aus jeder Zusammenhangskomponente, die nicht von einer anderen abhängt, mindestens ein Modul zur Infiltrierung ausgewählt werden. Andernfalls würde keines der Module dieser Zusammenhangskomponente infiltriert, und somit das MCP nicht komplett eliminiert. Da $|I|$ der Anzahl solcher unabhängigen Zusammenhangskomponenten gleicht, ist $|I|$ minimal.

d) Siehe Uploads im Moodle. Falls man keinen zufälligen Graphen benutzt, sondern den gegebenen kleineren Graphen, ist die minimale Anzahl der zu infiltrierenden Module 3.

Lösungen zu Aufgabe 2

a) Ja, dieses Problem ist wohldefiniert. Es gibt nur endlich viele einfache Pfade in G, einer von diesen muss minimale Länge haben.

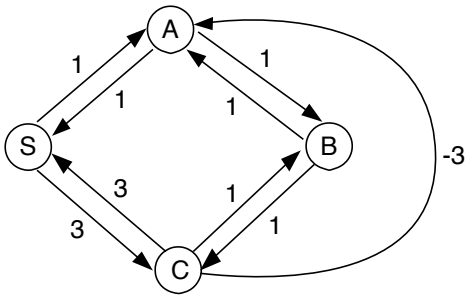
b) Zunächst grundsätzliches zum Problem:

- Falls es in einem Graphen Zyklen (oder in einem ungerichteten Graphen auch nur einzelne Kanten) mit insgesamt negativem Gewicht gibt, so gibt es keinen kürzesten Pfad im Graphen, sehr wohl aber einen kürzesten *einfachen* Pfad (siehe (a)). Der Bellman-Ford-Algorithmus in seiner ursprünglichen Form sucht nicht gezielt nach einfachen Pfaden, im Falle von Zyklen mit negativem Gewicht gibt er False zurück.

- Der Bellman-Ford-Algorithmus berechnet nur Pfade, die von einem festen Startknoten S aus zu allen anderen Knoten gehen. Wir beschreiben im folgenden daher das Problem, die Länge eines kürzesten einfachen und von einem festen Knoten S ausgehenden Pfades zu bestimmen. Eine Lösung des allgemeinen Problems ergibt sich unmittelbar durch Lösung dieses Problems für jede mögliche Wahl von S (mit anderen Worten: jeder Knoten wird einmal als S gewählt).

Unsere Idee besteht darin, den Bellman-Ford-Algorithmus dahingehend zu adaptieren, dass wann immer er einem Knoten v einen neuen gegenwärtig kürzesten Pfad π zuweisen möchte, in dem v schon einmal vorkommt (abgesehen davon, dass v jetzt neuer Endknoten wird), dies unterbunden wird, denn dann wäre der Pfad ja nicht einfach. Und nochmal: Im normalen Bellman-Ford-Algorithmus würde das auf negative Kreise hindeuten, aber die sind ja in dieser Anwendung nicht schlimm.

Ganz so einfach ist es aber leider nicht, aus folgendem Grund: Für jeden Knoten hält der Algorithmus nur *einen* gegenwärtig kürzesten Pfad aufrecht. Falls der Algorithmus zum Knoten u einen Pfad π_1, der v enthält, gespeichert hat und der Algorithmus v den Pfad "zuerst π_1, dann die Kante von u nach v" zuweisen möchte, wird dies unterbunden, obwohl womöglich ein Pfad π_2 zu u existiert, der die gleiche Länge wie π_1 hat und v nicht enthält. Wir machen uns diese Situation an folgendem Beispiel bewusst:



Wir durchlaufen die Kanten in folgender Reihenfolge:

$$\text{(S,A)},\ \text{(A,S)},\ \text{(A,B)},\ \text{(B,A)},\ \text{(B,C)},\ \text{(C,B)},\ \text{(C,S)},\ \text{(S,C)},\ \text{(C,A)}$$

Dann passiert im ersten Durchlauf der äußeren for-Schleife (Zeile 2 im Pseudo-Code aus der Vorlesung) zunächst folgendes:

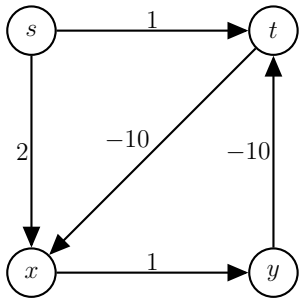
A.dist = 1 über S,A
B.dist = 2 über S,A,B
C.dist = 3 über S,A,B,C

Wann immer wir nun RELAX(C,A) aufrufen, so würde dies eigentlich zu einem kürzeren Pfad zu A führen. Da in diesem (zu C über S,A,B,C und dann die Kante von C nach A) aber A bereits vorkommt, würden wir dieses Update gemäß obiger Idee unterbinden. Nun gibt es aber zu C noch einen anderen Pfad der Länge 3, der A nicht enthält — das Update für A: "zuerst nach C, dann die Kante von C nach A" möchten wir haben!

Hinweis: Ab hier war die ursprüngliche Musterlösung falsch, aber ich lasse den Text stehen, um fruchtbare Diskussionen anzuregen.

Lösung des Problems: Wir speichern für jeden Knoten alle möglichen gegenwärtig kürzeste (und einfache) Pfade zu diesem Knoten. Bei Aufruf von RELAX für eine Kante (u, v) wenden wir obige Überlegung (d.h. Überprüfen, ob v nur als Endknoten vorkommt) auf jeden für u gegenwärtig gespeicherten Pfad an und speichern alle sich daraus ergebenden, gegenwärtig kürzeste und einfache Pfade für v. Auf diese Weise berechnen wir für alle Knoten t alle kürzesten einfachen Pfade mit Startknoten S und Endknoten t (ein formaler Korrektheitsbeweis verläuft völlig analog zum Korrektheitsbeweis des gewöhnlichen Bellman-Ford-Algorithmus) und finden damit auch einen kürzesten einfachen Pfad mit Startknoten S.

Was ist hier schief gelaufen? Die Idee "Wir speichern für jeden Knoten alle möglichen gegenwärtig **kürzeste und einfache** Pfade zu diesem Knoten" funktioniert leider nicht. Betrachten wir folgendes Beispiel.



Wir fügen die Kanten in folgender Reihenfolge hinzu:

$$(s,t),\ \ (t,x),\ \ (x,y),\ \ (y,t),\ \ (s,x).$$

Wenn wir nun den kürzesten einfachen Pfad von s nach t finden wollen, sehen wir schon, dass der Weg über x und y verläuft und eine Länge von -7 hat. Doch was macht der Bellman-Ford Algorithmus, wie wir ihn bisher beschrieben haben? Schon im ersten Durchlauf der Kanten wird für x ein kürzester einfacher Pfad über t mit Länge -9 abgespeichert. Der einfache Pfad von s nach x wird nicht abgespeichert, weil er Länge 2 hat, also länger ist, als der Weg über t. Wie gesagt: Wir wollten alle einfachen und kürzesten Pfade speichern. Darum verwerfen wir den direkten Weg von s nach x. Wenn nun der Algorithmus die Kante von y nach t betrachtet, stellt er fest, dass der bisherige Pfad schon über t lief, und somit nicht einfach wäre. Der einzige kürzeste einfache Pfad, der für t gespeichert wird, ist der direkt Weg von s nach t der Länge 1.

Der Kern des Problems liegt im Induktionsschritt des Korrektheitsbeweises. Er beruht auf der Tatsache das Teilpfade kürzester Pfade auch kürzeste Pfade sind. Stimmt das auch für einfache Pfade? Wie am obigen Beispiel zu sehen ist, nein!

Darum bleibt uns nichts anderes übrig als alle (einfachen!) Pfade zu speichern.

c) Nein, wir berechnen ja tatsächlich für alle Knoten t alle einfachen Pfade mit Startknoten S und Endknoten t.

Betrachte nun einen ungerichteten vollständigen Graphen G mit n Knoten, d.h. je zwei Knoten sind durch eine Kante miteinander verbunden. Einer dieser Knoten sei als S ausgezeichnet. Alle Kanten, die S enthalten, haben Gewicht n und alle anderen Kanten haben Gewicht -1. Dann ist jeder einfache Pfad mit Startknoten S, der alle Knoten von G enthält, ein kürzester einfacher Pfad von S zu seinem Endknoten (der Länge 2). Die Anzahl solcher Pfade ist gegeben durch $(n-1)!$ und damit ist die worst-case Laufzeit unseres Verfahrens superpolynomiell in der Anzahl an Knoten.

d) Ja, falls

- G gerichtet ist und keine Zyklen mit negativem Gewicht besitzt, dann kann jeder kürzeste Pfad in G zu einem einfachen Pfad gleicher Länge gemacht werden; das Problem ist also äquivalent dazu, die Länge eines kürzesten Pfades in G zu bestimmen (in diesem Fall ist dieses Problem wohldefiniert!) und kann mit dem gewöhnlichen Bellman-Ford-Algorithmus gelöst werden,
- (Spezialfall von obigem) G gerichtet ist und überhaupt keine Zyklen besitzt, dann ist jeder Pfad in G einfach und das Problem kann natürlich auch mit dem gewöhnlichen Bellman-Ford-Algorithmus gelöst werden.