Karlsruher Institut für Technologie

Algorithmen 1 – Sommersemester 2018
Zuverlässige Softwaresysteme im Kontext der Automobilindustrie
`http://verialg.iti.kit.edu`

Prof. Dr. Carsten Sinz – Markus Iser

Übungsblatt 11

Ausgabe: 04.07.2018 – 15:30
Abgabe: 13.07.2018 – 15:00

A Taillenweite bestimmen

Die *Taillenweite* eines gerichteten Graphen ist die Längce eines kürzesten Kreises im Graphen oder unendlich bei einem Wald.

A.1 Knotentaille (2 Punkte)

Implementieren Sie einen Algorithmus, der für einen bestimmten Knoten v die Länge eines kürzesten Kreises durch v findet, und analysieren Sie dessen Laufzeit.

Lösung

Die Grundidee ist eine Breitensuche bei v auszuführen. Die Tallienweite initialisiert man mit ∞.

Findet man bei der Breitensuche einen bereits besuchten Knoten, so schließt sich ein Kreis. Dieser braucht aber nicht notwendig v zu enthalten, er kann kürzer sein.

In einem ersten Algorithmenansatz muss man also prüfen, dass der gefundene Kreis auch die Wurzel enthält. Dazu läuft man ebenenweise die beiden Zweige der Breitensuche zurück, bis man einen gemeinsamen Knoten findet. Ist die gemeinsame Verzweigung die Wurzel, so ist ein Kreis durch v gefunden, sonst ein kürzerer.

Mit obigem Verfahren enumeriert man alle Kreise durch v, die keine Sehnen haben, also Querkanten zwischen Knoten des Kreises. Solche Kreise sind aber immer länger als die Kreise, die entsprechende Querkante verwenden.

Die Komplexität des ersten Algorithmenansatzes scheint die der Breitensuche, $\mathcal{O}(|V|+|E|)$ zu sein. Doch es gibt Graphen, in denen man sehr oft ($\mathcal{O}(\sqrt{|V|})$ mal) die Breitensuchen-Zweige rückwärts laufen muss, um kleinere Kreise auszuschließen. Dies lässt das Verfahren dann auf $\mathcal{O}(|V|^{\frac{3}{2}})$ ansteigen.

Doch man kann sich behelfen. Hierzu beobachtet man, dass die Prüfung auf verkürzte Kreise immer bis zur Wurzel zurücklaufen muss. D.h. beide Zweige der Breitensuche müssen von verschiedenen ausgehenden Kanten der Wurzel gestartet sein. Dieses Kriterium ist aber auch schon hinreichend, dass der gefundene Kreis die Wurzel enthält.

Wir wandeln daher die Breitensuche etwas anders ab und fügen eine „Trieb"-Variable für jeden Knoten hinzu. Für jeden besuchten Knoten speichert diese Variable den ersten Knoten des Breitensuch-Zweiges nach der Wurzel. Dieses entspricht dem ersten Trieb des Breitensuchbaums von der Wurzel.

Die Trieb-Variable wird bei Besuchen neuer Knoten einfach vererbt. Findet man dann einen bereits besuchten Knoten, kann anhand der beiden Trieb-Variablen in konstanter Zeit festgestellt werden, ob diese einen Kreis mit der Wurzel schließen. Damit hat das folgende Verfahren Zeitkomplexität $\mathcal{O}(|V| + |E|)$.

Algorithm 1: Tallienweite Knoten

```
Input: Graph G = (V, E), Knoten r
/* Initialisierung:                                                    */
1  girth := ∞
2  for v ∈ V do
3      p[v] := ⊥, d[v] := ⊥, t[v] := ⊥
4  p[r] := r, d[r] := 0, t[r] := r                          // Initialisiere Wurzel
   /* Verarbeite erste Ebene der BFS:                                  */
5  for w ∈ Adj(r) do
6      Q.enqueue(w)
7      p[w] := r, d[w] := d[r] + 1, t[w] := w                // Setze Triebnummer
8  while Q.notEmpty() do
9      v := Q.popTop()
10     for w ∈ Adj(v) do
11         if p[w] = ⊥ then
12             Q.enqueue(w)
13             p[w] := v, d[w] := d[v] + 1, t[w] := t[v]     // BFS und pflanze Trieb fort.
14         else if p[v] ≠ w then
               /* Hier schliesst sich ein Kreis                        */
15             if t[v] ≠ t[w] and d[v] + d[w] + 1 < girth then
                   /* Kreis von verschiedenen Trieben?                */
16                 girth := d[v] + d[w] + 1                 // Kleineren Kreis gefunden.
17 return girth
```

A.2 Graphentaille (2 Punkte, 1 Bonuspunkt)

Implementieren Sie einen Algorithmus, der die Taillenweite eines Graphen bestimmt, und analysieren Sie dessen Laufzeit. Lösungen in $\mathcal{O}(|V| \cdot |E|)$ geben einen Bonuspunkt.

Lösung

Nach der Vorarbeit in (a) ist es nun einfach die Tallienweite zu bestimmen, indem man (a) auf jeden Knoten anwendet und das Minimum aller berechneten Kreise bestimmt.

Man kann noch leicht optimieren, indem man die Breitensuche in Tiefe des aktuell minimalen *girth* abbricht, da ab dieser Tiefe keine kürzeren Kreise gefunden werden.

Die Gesamtlaufzeit ist dann $\mathcal{O}(|V| \cdot (|V| + |E|))$. Betrachtet man noch speziell den kantenlosen Graphen, dann ist diese sogar $\mathcal{O}(\max(|V| \cdot |E|, |V|))$.

Algorithm 2: Tallienweite

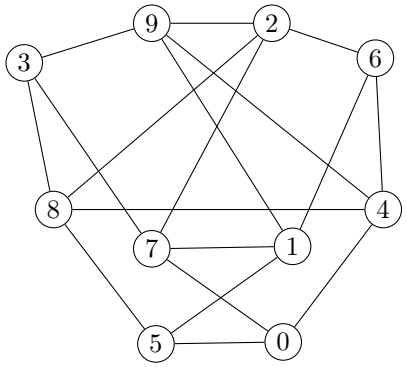
```
Input: Graph G = (V, E)
1  girth := ∞
2  for v ∈ V do
3      girth := min(girth, TallienweiteKnoten(G, v))
4  return girth
```

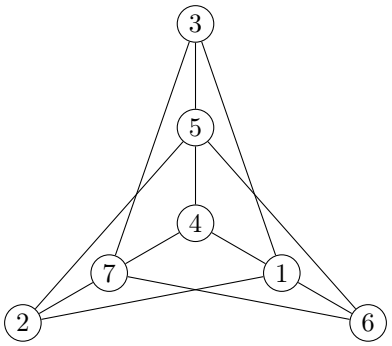
B Bipartite Graphen

Ein ungerichteter zusammenhängender Graph $G = (V, E)$ mit $|V| \geq 1$ heißt *bipartit*, wenn die Knotenmenge V so in zwei Mengen V_1 und V_2 aufgeteilt werden kann, dass für jede Kante $(u, v) \in E$ gilt $u \in V_1$ und $v \in V_2$ oder $v \in V_1$ und $u \in V_2$. Mit anderen Worten existiert eine Zerlegung von V, so dass Kanten nur zwischen Knoten aus V_1 und V_2 bestehen, nicht aber zwischen Knoten der V_i selbst.

B.1 Spielen (2 Punkte)

Zeigen oder widerlegen Sie jeweils die Bipartitheit der folgenden beiden Graphen!





B.2 Entwurf (2 Punkte)

Implementieren Sie einen Algorithmus, der in $\mathcal{O}(n + m)$ entscheiden, ob ein ungerichteter zusammenhängender Graph bipartit ist. Im Fall der Bipartitheit soll der Algorithmus eine mögliche Unterteilung V_1, V_2 ausgeben, sonst *nicht bipartit*.

Lösung

- Der linke Graph ist nicht bipartit: es existiert ein Kreis ungerade Länge $K = \langle 0, 5, 8, 3, 7 \rangle$ und ein ungerichteter zusammenhängender Graph ist genau dann bipartit, wenn er keinen Kreis ungerader Länge enthält.
- Der rechte Graph ist bipartit: Definiere $V_1 := \{2, 3, 4, 6\}, V_2 := \{1, 5, 7\}$.

Beim Entwurf des Algorithmus nutzen wir die Eigenschaft aus, dass es zwischen zwei nicht aufeinander folgenden Leveln bei einer Breitensuche keine Kanten geben kann, da diese sonst von der Breitensuche schon vorher gefunden worden wären. D.h. man kann die Knoten der Level der Breitensuche jeweils abwechselnd in die V_1 und V_2 packen. Sollte die Breitensuche eine Kreuzkante finden, so ist der Graph nicht bipartit, da der Graph dann einen Kreis ungerader Länge als Teilgraph enthält. Kreise ungerader Länge sind nicht bipartit.

Algorithm 3: Is Bipartit

```
Input: G = (V[1..n], E[1..m])
(parent, d) ← bfs(V[1], G)                    // BFS in G from first node
V_1, V_2: Queue of Vertices
f[1..n]: Array of {weiß, schwarz}
for i ← 1 to n do
    if d[i] mod 2 = 0 then
        V_1.push_back(i)
        f[i] ← white
    else
        V_2.push_back(i)
        f[i] ← black
for e ∈ E do
    {u, v} ← e
    if f[u] = f[v] then
        return "nicht bipartit"
return (V_1, V_2)
```

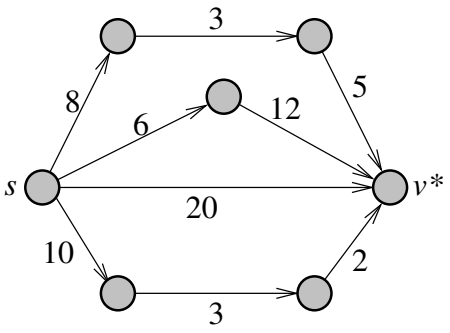
C Kürzeste Pfade

C.1 Spielen (2 Punkte)

Geben Sie einen gewichteten gerichteten Graph mit positiven Kantengewichten an, der einen Knoten v^* enthält, dessen vorläufige Distanz von Dijkstras Algorithmus mindestens dreimal mittles *decreaseKey* verringert wird. Geben Sie zusätzlich zum Graph auch den Knoten v^* und den Startknoten einer entsprechenden Dijkstra-Suche an.

Lösung

Der Startknoten der Dijkstra-Suche ist s, der Knoten, dessen vorläufiges Gewicht mindestens dreimal mit *decreaseKey* verringert wird, ist (wie in Aufgabenstellung bezeichnet) v^*:



C.2 Beweisen (2 Punkte)

Zeigen Sie: In gewichteten gerichteten Graphen sind Teilpfade von kürzesten Pfade wiederum kürzeste Pfade.

Lösung

Sei $P = (u, \ldots, x, \ldots, y, \ldots, v)$ ein kürzester Pfad von u nach v und $Q \leftarrow (x, \ldots, y)$ ein Teilpfad.
Annahme zum Widerspruch: Q ist kein kürzester Pfad von x nach y. Da ein Pfad Q existiert und somit der Fall, dass es keinen Pfad gibt ausgeschlossen wird, gibt es also einen kürzeren Pfad $Q' = (x, \ldots, y)$. Ersetze in P den Teilpfad Q durch Q', das ergibt einen gültigen Pfad von u nach v der kürzer ist als P. Dies ist ein **Widerspruch,** da P kürzester Pfad. □

C.3 Baum (4 Punkte)

Ein gerichteter Graph heiße ein *gerichteter Baum* mit *Wurzel* r, wenn alle seine Knoten vom Knoten r aus erreichbar sind und der Eingangsgrad aller seiner Knoten ≤ 1 ist.

Behauptung: Sei $G = (V, E)$ ein gewichteter gerichteter Graph ohne negative Kreise, in dem von einem Knoten s aus alle anderen Knoten erreichbar sind. Dann gibt es einen Teilgraph T von G, der $|V|$ Knoten hat, ein gerichteter Baum mit Wurzel s ist und dessen Pfade alle kürzeste Pfade in G sind.

Beweisen Sie die Behauptung. Nehmen Sie zunächst an, dass alle kürzesten Pfade eindeutig sind und betrachten Sie den Teilgraph T der aus allen kürzesten Pfaden besteht, die an s beginnen. Erweitern Sie dieses Ergebnis auf den Fall, dass kürzeste Pfade nicht immer eindeutig sind.

Lösung

Offensichtlich ist T_0 ein Teilgraph mit $|V|$ Knoten.

Angenommen, es gibt $u \neq s$ mit Eingangsgrad ≥ 2 in T_0. Also gibt es zwei Kanten (x, u) und (y, u) in T_0 mit $x \neq y$. Da T_0 Vereinigung von kürzesten Pfaden ist, die bei s beginnen, gibt es zwei kürzeste Pfade $P = (s, \ldots, x, u, \ldots, v)$ und $Q = (s, \ldots, y, u, \ldots, w)$ in T_0. Nach vorheriger Teilaufgabe sind $(s, \ldots, x, u)$ und $(s, \ldots, y, u)$ zwei kürzeste Pfade. Da sie verschieden sind, ist das eine Widerspruch zur Annahme, dass kürzeste Pfade eindeutig sind. Also ist T_0 ein gerichteter Baum mit Wurzel s.

Angenommen, es gibt einen Pfad $(u, \ldots, v)$ in T_0, der nicht kürzester Pfad in G ist. Da es in einem gerichteten Baum genau einen Pfad von der Wurzel zu einem Knoten gibt (andernfalls käme ein Eingangsgrad ≥ 2 vor), muss es in T_0 einen Pfad $(s, \ldots, u)$ geben, so dass $(s, \ldots, u, \ldots, v)$ der eindeutige Pfad von s nach v ist in T_0. Da T_0 nach Definition nur aus kürzesten Pfaden in G besteht, ist $(s, \ldots, u, \ldots, v)$ kürzester Pfad in G und somit $(u, \ldots, v)$ kürzester Pfad von u nach v in G. Dies ist ein **Widerspruch.** D.h. wenn alle kürzesten Pfade in G eindeutig sind, dann sind alle Pfade in T_0 kürzeste Pfade in G. Sei $(s, \ldots, u)$ kürzeste Pfad von s nach u in T_0 und somit (nach Definition von T_0 auch kürzester Pfad in G). Da es in einem gerichteten Baum genau einen Pfad von der Wurzel zu einem Knoten gibt (andernfalls käme ein Eingangsgrad ≥ 2 vor), muss $(s, \ldots, u, \ldots, v)$ der kürzeste Pfad von s nach v sein und somit $(u, \ldots, v)$ kürzester Pfad von u nach v. D.h. wenn alle kürzesten Pfade in G eindeutig sind, dann sind alle Pfade in T_0 kürzeste Pfade.

Ergänzung für nicht-eindeutige kürzeste Pfade

Ein Graph ohne negative Kreise hat eindeutige kürzeste Distanzen. Da s jeden Knoten erreichen kann, gibt es eine eindeutige kürzeste Distanz $\mu_s(v)$ zu jedem Knoten v. Konstruiere einen Graphen G' aus G der eindeutige kürzeste Pfade hat und $\mu_s(\cdot)$ nicht verändert: Solange es in G' noch zwei verschiedene kürzeste Pfade $P = (s, \ldots, u)$ und $Q = (s, \ldots, u)$ gibt, gibt es auch noch zwei kürzeste Pfade $P' = (s, \ldots, x, u')$ und $Q' = (s, \ldots, y, u')$ mit $x \neq y$. Entferne iterativ (y, u') aus G' bis die kürzesten Pfad eindeutig sind.

Dass sich $\mu_s(\cdot)$ wirklich nicht ändert, überlegt man sich so: Beim entfernen von Kanten werdem kürzeste Distanzen allenfalls größer. Betrachte den kürzesten Pfad $\langle s, \ldots, y, u', \ldots, v\rangle$ von s nach v mit Präfix Q'. Dann ist $\langle s, \ldots, x, u', \ldots, v\rangle$ mit Präfix P' ein Pfad gleicher Länge und folglich ebenfalls kürzester Pfad von s nach v. Nach Entfernen von (y, u'), was den Pfad Q' zerstört, gibt es also immer noch einen Pfad von s nach v der Länge $\mu_s(v)$. Es ist $\mu_s(v)$ aber immer noch eine kürzeste Distanz, da alle kürzesten Distanzen allenfalls größer geworden wären.

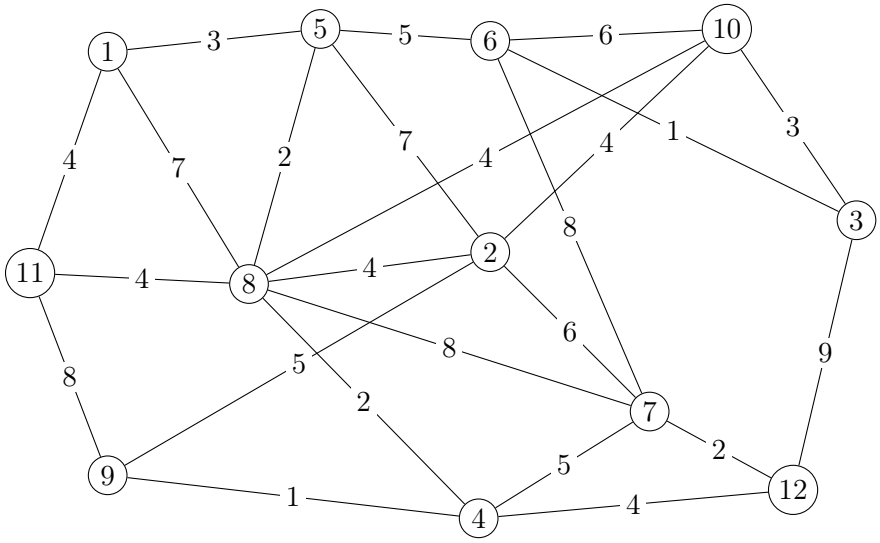
Wir haben aber in vorhergehendem Teil schon gezeigt, dass bei eindeutigen kürzesten Pfaden ein entsprechender Baum exisiert. Es lässt sich T also iterativ aus G konstruieren: Als erstes entfernt man alle Kanten aus G, die nicht zu einem kürzesten Pfad gehören, der bei s beginnt. Dann entfernt man wie oben beschrieben iterativ Kanten, bis alle kürzesten Pfade, die bei s beginnen, eindeutig sind. Es gibt aber nur Kanten die zu solchen kürzesten Pfaden gehören. Also haben alle Knoten Eingangsgrad ≤ 1, sonst gäbe es mehr als einen kürzesten Pfad. Außerdem sind alle Knoten noch erreichbar. Der zu diesem Zeitpunkt übrige Graph ist ein gesuchter Baum. □

Dass in T_M alle Knoten von s aus erreichbar sind, folgt direkt aus der Definition von T_M. Zu zeigen ist noch, dass der Knotengrad ≤ 1 ist für alle Knoten.

Annahme zum Widerspruch: Es gebe in T_M einen Knoten u mit Eingangsgrad ≥ 2. Also gibt es zwei Kanten (x, u) und (y, u) in T_M mit $x \neq y$. Da T_M Vereinigung von kürzesten Pfaden ist, die alle von s ausgehen, gibt es zwei kürzeste Pfade $P = (s, \ldots, x, u, \ldots, v)$ und $Q = (s, \ldots, y, u, \ldots, w)$. Nach vorheriger Teilaufgabe sind $(s, \ldots, x, u)$ und $(s, \ldots, y, u)$ zwei kürzeste Pfade. Da sie verschieden sind, ist dass eine **Widerspruch** zur Definition von M, nach der in M zu jedem Knoten $\neq s$ genau ein kürzester Pfad ausgehend von s existiert. Also ist T_M ein gerichteter Baum mit Wurzel s. □

D Minimum Spanning Tree

Berechnen Sie einen minimum spanning tree (MST) des angegebenen Graphen. Geben Sie jeweils die Kanten des MST in der Reihenfolge an, in der sie der Algorithmus auswählt. Um Eindeutigkeit herzustellen, verwenden Sie Knoten 1 als Startknoten von Jarník-Prim und sortieren Sie die Kanten bei *beiden* Algorithmen bei Unentschieden lexikographisch nach den Nummern des sortierten Endknotentupels (Beispiel: $(5, 3) < (4, 10)$ weil $3 < 4$)



Lösung

D.1 Jarník-Prim (2 Punkte)

Reihenfolge: $\{1,5\},\{5,8\},\{8,4\},\{4,9\},\{1,11\},\{8,2\},\{2,10\},\{10,3\},\{3,6\},\{4,12\},\{12,7\}$

D.2 Kruskal (2 Punkte)

Reihenfolge: $\{3,6\},\{4,9\},\{4,8\},\{8,5\},\{7,12\},\{1,5\},\{3,10\},\{1,11\},\{2,8\},\{2,10\},\{4,12\}$

In beiden Fällen entsteht folgender Graph