Einführung in die diskrete Mathematik

Dozent
Professor Dr. Jens Vygen

Mitschrift
Maximilian Kessler

Zusammenfassung

Dies ist der Beginn meiner Vorlesungsmitschrift zur 'Einführung in die diskrete Mathematik', gehalten im WS 2020/2021 von Prof. Dr. Jens Vygen an der Universität Bonn.
In der Mitschrift sind eigentlich nur die ersten 6 Vorlesungen inbegriffen, und diese sind vermutlich sehr lückenhaft. Auch wurden die Notizen nicht überarbeitet und sind sicherlich fehlerbehaftet.
Nichtsdestotrotz sollte man einen Eindruck bekommen, um was es in der Vorlesung geht.

Inhaltsverzeichnis

1 Einführung **1**

2 Maximale Flüsse **14**

Lecture 1: Eulerkreise

Di 27 Okt 2020 16:10

Literatur: Kombinatorische Optimierung (Jens Vygen)

- Vorlesung findet vollständig online statt
- Vorlesung wird aufgezeichnet (eCampus)
- kein Skript, siehe dazu Literatur
- Übungsgruppen: 2 online und 3 präsenz
- Anmeldung am Mittwoch (morgen) 20 Uhr
- Übungsblätter (An und Abgabe über eCampus) immer bis Donnerstag, 16 Uhr

1 Einführung

Definition 1.1. Graphen sind ungerichtet oder gerichtet, in jedem Fall endlich. Sie können (meist) parallele Kanten haben, allerdings keine Schleifen. Wir notieren $G = (V(G), E(G))$ und $e = \{v, w\}$ bzw. $e = (v, w)$.

Ein ungerichteter Graph ist zusammenhängend

$$\Leftrightarrow \forall s,t \in V(G)\ \exists \text{ Weg von } s \text{ nach } t \text{ in G } (s\text{-}t\text{-Weg})$$
$$\Leftrightarrow \forall s,t \in V(G)\ \exists \text{ Kantenzug von } s \text{ nach } t \text{ in } G$$

.

Gegeben ein ungerichteter Graph, können wir in linearer Zeit ($\mathcal{O}(m+n)$, wobei $n = |V(G)|$ und $m = |E(G)|$) entscheiden, ob er zusammenhängend ist? (Ja, nutze Tiefen oder Breitensuche)

Wir können auch die Zusammenhangskomponenten bestimmen (in linearer Zeit)

Definition 1.2. Ein gerichteter Graph ist stark zusammenhängend

$$\Leftrightarrow \forall s,t \in V(G)\ \exists\ s\text{- }t\text{-Weg in } G$$
$$\Leftrightarrow \forall s,t \in V(G)\ \exists \text{ Kantenzug von } s \text{ nach } t \text{ in } G$$
$$\Leftrightarrow \forall \varnothing \neq X \subseteq V(G) : \delta_G^+(X) = \{e = (v,w) | v \in X, w \notin X\} \neq \varnothing$$
$$\Leftrightarrow \forall \varnothing \neq X \subseteq V(G) : \delta_G^-(X) = \delta^+_{V(G)\setminus X} \neq \varnothing$$

.

Können wir in linearer Zeit entscheiden,obein gegebener Graph stark zusammenhängend ist? (Ja, Nutze 2 mal BFS oder DFS)

Königsberger Brückenproblem:
Ein Spaziergang ist eine Kantenfolge, die jede Kante höchstens einmal durchläuft. Ein Spaziergang heiße eulersch , wenn er jede Kante genaueinmal durchläuft und geschlossen ist (d.h. Anfangs- und Endpunkt stimme überein).

Satz 1.3 (Euler 1736, Hierholzer 1873)**.** Ein zusammenhängender Graph hat einen eulerschen Spaziergang gdw

- $|\delta(v)|$ gerade ist $\forall v \in V(G)$
- $\left|\delta^+(v)\right| = \left|\delta^-(v)\right|$ für alle $v \in V(G)$, falls G gerichtet

Ein solcher Graphheißt auch eulersch.

Beweis. Die Bedingung ist offensichtlich notwendig. Wir zeigen, dass sie auch hinreichend ist. Sei $W = v_1, e_1, v_2, .., v_k, e_k, v_{k+1}$ ein längster Spaziergang. Alle Wanten in $\delta^+(v_{k+1})$ gehören zu W. Alle Kanten in $\delta^-(v_1)$ gehören ebenfalls zu W. Es foglt $v_1 = v_{k+1}$ (da G eulersch).

Angenommen, W enthalte nicht sämtliche Kanten. Dann gibt es eine, sagen wir e, die nicht zu W gehört und von der mindestens ein Endpunkt von W besucht wird (da G zusammenhängend). Dann können wir e und W zu einem längeren Spaziergang als W kombinieren. Also gibt es keine solche Kante und W enthält bereits alle Kanten. □

Algorithmus 1 : Eulers Algorithmus

Eingabe : ein ungerichteter zusammenhängender Graph G, der eulersch ist
Ausgabe : ein eulerscher Spaziergang in G

Def `Eulers Algorithmus`**:**
  Wähle $v_1 \in V(G)$ beliebig
  return $w :=$ `Euler`(v_1)

Def `Euler`(v_1)**:**
1 $W := v_1$
  $x := v_1$
2 **if** $\delta(x) \neq \varnothing$ **then**
    go to 4
  else
    sei $e \in \delta(x)$, etwa $e = \{x, y\}$
3 $W := W, e, y$
  $x := y$
  go to 2
4 Sei $v_1, e_1, v_2, e_2, \ldots, v_k, e_k, v_{k+1}$ das aktuelle W
  for $i = 2$ **to** k **do**
    $W_i :=$ `Euler`(v_i)
5 **return** $W := v_1, e_1, W_2, e_2, W_3, \ldots, W_k, e_k, v_{k+1}$

Satz 1.4. EULERS ALGORITHMUS arbeitet korrekt. Seine Laufzeit ist $\mathcal{O}(m)$, wobei $m = |E(G)|$

Beweis. Wir zeigen per Induktion nach $|E(G)|$, dass $EULER(v)$ für beliebige eulersche Graphen (nicht notwendigerweise zusammenhängend) einen eulerschen Spaziergang in der Zusammenhangskomponente von G ergibt, die v enthält.
Offenbar durchlaufen wir keine Kanta zweimal und die Laufzeit ist $\mathcal{O}(m)$, denn jede Kante wird sofort gelöscht, sobald sie durchlaufen wird.
Wegen der Euler-Bedingung ist der Spaziergang geschlossen, wenn wir (4) erreichen. Sei G' der Graph zu diesem Zeitpunkt. G'ist auch eulersch. Sei G_v die Zusammenhangskomponenten von G, die v enthält. Für $e \in E(G_v) \cap E(G')$ $\exists$ minimales $i \in \{2, \ldots, v_k\}$, sodass e in derselben Zusammenhangskomponenten wie v_i von G' ist.
Nach Induktionsvoraussetzunggehört e zu W_i und der abschließende Spaziergang besucht tatsächlich alle Kanten.

□

Lecture 2: Hamiltonkreise, SCC-Algorithmus

Do 29 Okt 2020 16:15

Definition 1.5. Ein Hamiltonkreis (in G) ist ein Kreis in G, der alle Knoten enthält.

Frage: Gegeben ein ungerichteter Graph G, enthält G einen Hamilton-Kreis?

Satz 1.6 (Dirac,1952)**.** Jeder einfache ungerichtete Graph mit $n \geqslant 3$

Knoten und minimalem Grad mindestens $\frac{n}{2}$ enthält einen Hamilton-Kreis.

Beweis. Sei G ein solcher Graph. G ist zusammenhängend, da sonst jeder Knoten in einer kleinsten Zusammenhangskomponente weniger als $\frac{n}{2}$ Nachbarn hat.

Sei P ein längster Weg in G, mit Knoten $x_1, \ldots, x_{k+1}$. Die Nachbarn von x_1 und von x_{k+1} liegen alle auf P.
Sei

$$A = \{i \in \{1, \ldots, k\} \mid \{x_i, x_{k+1}\} \in E(G)\}$$
$$B = \{i \in \{1, \ldots, k\} \mid \{x_1, x_{i+1}\} \in E(G)\}$$

Wegen $|A| \geqslant \frac{n}{2}$ und $|B| \geqslant \frac{n}{2}$ und $k < k+1 \leqslant n$ ist $A \cap B \neq \varnothing$ Sei $j \in A \cap B$. Wir erhalten einen Kreis C der Länge $k+1$.

Wäre C nicht hamiltonsch, so gäbe es eine Kante $\{x_i, v\}$ mit $i \in \{1, \ldots, k+1\}$ und $v \notin V(C)$. (da G zusammenhängend). Dies ergibt abereinen Weg mit $k+2$ Knoten, Widerspruch. □

Bemerkung: Die untere Schranke $\frac{n}{2}$ ist optimal, ansonsten gibt es Gegenbeispiele der Form:

Graphendurchmusterung (Erinnerung) : DFS und BFS

Wir möchten jetzt die starken Zusammenhangskomponenten in linearer

Zeit berechnen

Algorithmus 2 : Strongly connected component algorithm

Eingabe : ein gerichteter Graph G
Ausgabe : eine Funktion $comp : V(G) \to \mathbb{N}$, die angibt, in welcher starnken Zusammenhangskomponenten ein Knoten liegt

Def `strongly connected component`**:**
1 $R := \varnothing$
$N := 0$
2 **for all** $v \in V(G)$ **do**
  if $v \notin R$ **then**
    `visit1` (v)
3 $R := \varnothing$
$K := 0$
4 **for** $i := |V(G)|, \ldots, 1$ **do**
  if $\varphi^{-1}(i) \notin R$ **then**
    $k := k + 1$
    `visit2` (φ^{-1}(i))

Def `visit1`**:**
1 $R := R \cup \{v\}$
2 **for all** $(v, w) \in \delta^+(v)$ **do**
  if $w \notin R$ **then**
    `visit1` (w)
3 $n := n + 1$, $\varphi(v) := n, \varphi^{-1}(n) := v$

Def `visit2`**:**
1 $R := R \cup \{v\}$
2 **for all** $(w, v) \in \delta^-(v)$ **do**
  if $w \notin R$ **then**
    `visit2` (w)
3 $comp(v) := k$

Satz 1.7. Der Algorithmus funktioniert und hat Laufzeit $\mathcal{O}(m + n)$

Beweis. Laufzeit klar (wir machen 2 mal DFS) Auch klar: 2 Knoten in derselben starken Zusammenhangskomponente bekommen dieselbe comp-Nummer. Wirzeigen, dass 2 Knoten u und v mit $comp(u) = com(v)$ in derselben starken Zusammenhangskomponente liegen.
Sei $r(u)$ bzw. $r(v)$ der Knoten, der von u bzw. v erreichbar ist und unter all diesen die höchste φ-Nummer hat. Da $comp(u) = com(v)$ liegen u und v in derselben Anti-Arboreszens der zweiten DFS, d.h. $r = r(u) = r(v)$ ist die Wurzel dieser Anti-Arboreszenz. r ist also von u und von v aus erreichbar. r ist von u auserreichbar und $\varphi(r) \geqslant \varphi(u)$
r wurde nicht später als u zu R im ersten DFS hinzugefügt und der erste DFS-Branching □

Satz 1.8. Der Alg. bestimmt eine topologische Ordnung in dem gerichteten Graphen, der durch Kontraktion aller starken Zusammen-

hangskomponenten entsteht.

Definition 1.9. Ein gerichteter Graph heißt azyklisch, wenn er keine (gerichteten) Kreies enthält. Eine topologische Ordnung eines gerichteten Graphen G ist eine Nummerierung $f : V(G) \to \{1, \ldots, n\}$ (Bijektion), sodass mit $\forall (v, w) \in E(G)$ gilt, dass $f(v) < f(w)$

Korollar 1.10. Ein gerichteter Graph ist genau dann azyklisch, wenn er eine topologische Ordnung besitzt.

Definition 1.11. Sei $k \geqslant 2$. Ein ungerichteter Graph mit mehr als k Knoten heißt **k-fach** zusammenhängend (k-zusammenhängend, k-Knotenzusammenhängend), wenn er nach entfernen beliebiger $k-1$ Knoten zusammenhängend bleibt.
Ein ungerichteter Graph mit mind. zwei Knoten heißt **k-fach Kantenzusammenhängend**, wenn er nach Entfernen beliebiger $k-1$ Knoten immer zusammenhängend bleibt.
Das größte k bzw. l, so dass G k-zusammenhängend bzw. l-Kantenzusammenhängend ist, heißt der **Knotenzusammenhang** bzw. **Kantenzusammenhang** von G.

Lecture 3: Minimum Spanning Tree Problem

Di 03 Nov 2020 16:11

Definition 1.12. Sei G ein (gerichteter oder ungerichteter) Graph. Eine **Ohrenzerlegung** von G isteine Folge $r, P_1, \ldots, P_k$ mit $V(G) = \{r\} \cup V(P_1) \cup \ldots \cup V(P_k)$ und $E(G) = E(P_1) \cup \ldots \cup E(P_k)$, so dass jedes P_i entweder ein Pfad ist, von dem genau die Endpunkte zu $\{r\} \cup V(P_i) \cup \ldots \cup V(P_{i-1})$ gehören oder ein Kreis ist, von dem genau ein Knoten zu $\{r\} \cup V(P_1) \cup \ldots \cup V(P_{i-1})$ gehört.
$P_1, \ldots, P_k$ heißen die **Ohren**. Wenn $P_2, \ldots, P_k$ Pfade sind, heißt die Ohrenzerlegung **echt**.

Satz 1.13 (Whitney, 1932)**.** Ein ungerichteter Graph hat genau dann eine echte Ohrenzerlegung,

Beweis. $'\Rightarrow'$: Induktion nach # Ohren. Trivial wenn $k = 1$. Sei G 2-zusammenhängend, füge ein Ohre P (mit Endpunkten $x \neq y$ hinzu. Dann ist $G + P - v$ zusammenhängend $\forall v \in V(G + P)$.
$'\Leftarrow'$: Sei G 2-zusammenhängend. Sei G' der einfache Teilgraph, der durch Entfernen parallelere Kanten entsteht. $\Rightarrow G'$ ist ebenfalls 2-zusammenhängend. Da G' kein Baum ist, enthält G' einen Kreis mit mindestens 3 Kanten. Sei H ein maximaler Teilgraph von G', so dass H eine echte Ohrenzerlegung besitzt (insb. $|V(HR| \geqslant 3$). Angenommen, H enthielte nicht alle Knoten aus G. Dann existiert eine Kante $e = \{x, y\} \in E(G')$ mit $x \in H$ und $y \notin H$. Sei $z \in V(H) \backslash \{x\}$. Da G' 2-zusammenhängend ist, ist auch $G' \backslash x$ zusammenhängend, also enthält $G' \backslash x$ einen y-z-Weg. Sei z' der erste

Knoten auf diseem Weg, der zu H gehört. Bemerke: $z' \neq x$. Dann können wir $P_{[y,z']} + e$ als Ohr zu H hinzufügen. Etwaige fehlende Kanten können einzeln als Ohren hinzugefügt werden. □

MINIMUM SPANNING TREE (MST)
Instanz: ungerichter Graph G und Kostenfunktion $c : E(G) \to \mathbb{R}$
Aufgabe: finde einen aufspannenden Baum H in G mit $c(E(H))$ minimal oder entscheide, dass G unzusammenhängend.
MAXIMUM WEIGHT FOREST
Instanz: ungerichteter Graph G und Kostenfunktion $c : E(G) \to \mathbb{R}$.
Aufgabe: Finde einen Wald H in G mit $c(E(H))$ maximal.
Diese beiden Probleme sind äquivalent in folgendem Sinne:

Definition 1.14. P ist **linear reduzierbar** auf Q, wenn es in linearer Zeit berechenbare Funktionen f, g gibt, sodass f jede Instanz x von P in einer Instanz $f(x)$ von Q zuordnet und g jede Lösung von $f(x)$ einer Lösung von x.
P, Q sind **äquivalent**, wenn P auf Q und Q auf P linear reduzierbar sind.

Proposition 1.15. MST und MWF sind äquivalent.

Beweis. Sei (G, c) eine Instanz von MST. Setze $c'(e) = K \backslash c(e)$, wobei $k = 1 + \max_{e \in E(G)} c(e)$. Dann ist (G, c') eine Instanz von MWF.
Sei H ein MWFin (G, c'). Dann ist H (sofern zusammenhängend) MST in (G, c).
Sei (G, c) eine Instanz von MWF. Entferne alle Kanten e. Setze $c'(e) = -c(e)$ und füge minimale Zahl von Kanten (beliebiges Gewicht) hinzu, sodass der Graph zusammenhängend wird und erhalte (G', c'). Dies ist Instanz von MST. Sei $F := E(G') \backslash E(G)$. Sei H ein MST in (G', c'). Dann ist $H - F := (V(H), E(H) \backslash F)$ ein MWF in (G, c), denn wenn es einen teureren maximalen Wald H^* gäbe, wäre $H^* + F$ ein billigerer spanning tree in (G', c') als H. □

Satz 1.16 (Syvester 1857, Cayley 1889)**.** Für $n \in \mathbb{N}$ ist die Anzahl der aufspannenden Bäume im vollständigen Graphen K_n auf n Knoten gleich n^{n-2}.

Beweis. Sei t_n die Anzahl der aufspannenden Bäume in K_n. Sei

$$\mathfrak{B}_{n,k} = \{(B, f) \mid B \text{ Branching}, V(B) = \{1, \dots, n\}, |E(B)| = k, f : E(B) \to \{1, \dots, k\} \text{ Bijektion}\}.$$

Es ist nun $\mathfrak{B}_{n,n-1} = t_n \cdot n \cdot (n-1)!$.
$\mathfrak{h}B_{n,0} = 1$. Wir zeigen $|\mathfrak{B}_{n,i+1}| = n(n-i-1)|\mathfrak{B}_{n,i}|$. Dies impliziert: $|\mathfrak{B}_{n,n-1}| = n^{n-1}(n-1)!$ und wir erhalten $t_n = n^{n-2}$.
Für jedes $(B, f) \in \mathfrak{B}_{n,i+1}$ definieren wir $g(B, f) \in \mathfrak{B}_{n,i}$ durch entfernen der Kante $f^{-1}(i+1)$. Jedes $(B', f) \in \mathfrak{B}_{n,i}$ ist Bild von genau $n(n-i-1)$ Elementen von $\mathfrak{B}_{n,i+1}$:
Wir fügen eine Kante $e = (v, w)$ hinzu und setzen $f(e) = i + 1$, und es gibt n Möglichkeiten für v (alle Kanten) und $n - i - 1$) Möglichkeiten für w (die Wurzeln der Zusammenhangskomponenten von B', die nicht v enthalten). □

Satz 1.17. Sei (G, c) eine Instanz von MST und sei T ein aufspannender Baum. Dann sind folgenden Aussagen äquivalent:

(a) T ist optimal, d.h. T hat minimales Gewicht
(b) Für jede Kante $e = \{x, y\} \in E(g) \backslash E(T)$ ist e eine Kante maximalen Gewichts auf dem Kreis in $T + e$ (Fundamentalkreis von e bezüglich T)
(c) Für jedes $e = \{x, y\} \in E(T)$ ist e eine Kante minimalen Gewichts in $\delta_G(X)$, wobei X die Knotenmenge der Zusammenhangskomponente von $T - e$ ist, die x enthält (Fundamentalschnitt von e bezüglich T).
(d) Wir können $E(T) = \{e_1, \ldots, e_{n-1}\}$ ordnen, so dass für jedes $i \in \{1, \ldots, n-1\}$ eine Menge $X \subseteq V(G)$ existert, sodass e_i eine billigste Kante in $\delta_G(X)$ ist und $e_j \notin \delta_G(X)$.

Beweis. $(a) \Rightarrow (b)$: Angenommen, $\exists e = \{x, y\} \in E(G) \backslash E(T)$ und f eine Kante auf dem x- y-Weg in T mit $c(f) > c(e)$. Dann ist $T - f + e$ ein billigerer Spannmaub.

Angenommen, $\exists e \in E(T)$ und X Knotenmenge einer Zusammenhangskomponente von $T - e$, $\{x, y\} = f \in \delta(x)$ mit $c(f) < c(e)$. Dann ist $f \notin E(T)$ und der x- y-Weg in T muss eine kante aus $\delta(X)$ enthalten, und e ist die einzige solche Kante $\Rightarrow \neg(b)$.

□

Lecture 4: Minimum Spanning Tree

Di 10 Nov 2020 16:15

TODO: Beweis von Satz über MST

Algorithmus 3 : Kruskals Algorithmus

"test"
Eingabe : Ein zwusammenhängender ungerichteter Graph G, Kostenfunktion $c : E(G) \to \mathbb{R}$
Ausgabe : ein optimaler aufspannender Baum

```
1 Sortiere die kanten E(G) = {e_1, ..., e_m} sodass c(e_1) ⩽ c(c_2) ⩽ ... ⩽ c(e_m)
2 T := (V(G), ∅)
3 for i = 1 to m do
      if T + e_i ist ein Wald then
          T := T + e_i
```

Satz 1.18. Kruskals Algo ist korrekt.

Algorithmus 4 : Prims Algorithmus

Eingabe : ein zusammenhängender ungerichteter Graph G, Kosten $c : E(G) \to \mathbb{R}$

Ausgabe : ein optimaler aufspannender Baum T

```
1 Wähle v ∈ V(G), T := ({v}, ∅)
2 while V(T) ≠ V(G) do
      wähle e ∈ δ_G(V(T)) minimalen Gewichts
      T := T + e
```

Satz 1.19. Primes Algorithmus ist korrekt. Seine Laufzeit ist:
- Naiv: $O(mn)$
- Besser: $O(m + n^2)$
- Mit Binärheap: $O(m \log n)$
- Mit Fibonacci-Heap: $O(m + n \log n)$

Lecture 5: Fibonaccii Heaps

Di 10 Nov 2020 16:16

Satz 1.20. Man kann eine Datenstruktur für eine endliche Menge (anfangs leer) führen, in der jedem Element u ein Schlüssel $d(u) \in \mathbb{R}$ zugeordnet ist, und eine beliebige Folgo von
- p INSERT-Operationen (füge ein Elmeent u mit Schlüssel $d(u)$ hinzu
- n DELETEMIN-Operationen (finde eine Element u mit $d(u)$ minimal und entferne u aus der Menge)
- m DECREASEKEY - Operationen (verringere den Schlüssel eines Elments u auf einen neuen Wert).

ausführen in der Gesamtzeit $\mathcal{O}(m+p+n \cdot \log p)$. Diese Datenstruktur heißt **Fibonacci-Heap**

Bemerkung. PRIMS ALGO mit F-Heap braucht Laufzeit $O(m + n \log n)$

Beweis. Wir speichern die Menge, die wir U nennen, in einem Branching (U, E) mit einer Funktion $\varphi : U \to \{0, 1\}$ mit folgenden Eigenschaften:

(i) Für alle $(U, V) \in E$ ist $d(u) \leqslant d(v)$ (Heapordnung)

(ii) Für alle $u \in U$ können die kinder von u so mit $1, \ldots, |\delta^+(n)|$ nummeriert werden, dass für das i-te Kind v

$$\left|\delta^+(v)\right| + \varphi(v) \geqslant i - 1.$$

(iii) Falls u und v verschieden Wurzeln sind, gilt $\left|\delta^+(u)\right| \neq \left|\delta^+\right|(v)$.

Aus (ii) folgt:

(iv) Ist $\left|\delta^+\right|(u) \geqslant k$, so hat u mindestens $\sqrt{2}^k$ Nachkommen (inkl. u selbst). (Beweise per Induktion).

Hieraus folgt dann $\left|\delta^+(u)\right| \leqslant 2\log|U| \leqslant 2 \log p \quad (\star)$.

Wir speichern nun
- $|U|$
- die Wurzeln von (U, E) mit einer Funktion $b : \{0, 1, \ldots, \lfloor 2 \log|U| \rfloor\}$, wobei eine Wurzel u mit $\delta^+(u) = k$ in $b(k) = u$ gespeichert wird.

Dies ist möglich wegen $(\star)$ und (iii). Achtung: Aus $b(i) = u$ folgt weder, dass u Wurzel ist, noch dass $\left|\delta^+(u)\right| = i$.
- Eine doppelt verkettete Liste der Kinder jedes Elements (in bel. Reihenfolge)
- den Vorgänger (falls er existiert)
- den Ausgangsgrad jedes Elements

Wir implementieren die Operationen wie folgt: TODO: finish proof □

Wir betrachten nun folgende Probleme:

Maximum weight Branching
Instanz: ger. Graph G, $c : E(G) \to \mathbb{R}$
Ausgabe: finde maximales gewichtetes Branching in G

Minimum weight arborescence
Instanz: ger. Graph G, $c : E(G) \to \mathbb{R}$
Ausgabe: finde min. gewichtete Arboreszenz in G oder entschedie, dass keine existiert.

Minimum weight rooted arborescence
Instanz: ger. Graph G, $c : E(G) \to \mathbb{R}$, Knoten $r \in V(G)$
Ausgabe: finde min. gerichtete Arboreszenz mit Wurzel r in G oder entschedie, dass keine existiert.

Proposition 1.21. Diese Probleme sind alle äquivalent.

Lecture 6: Branching Algorithmus

Do 12 Nov 2020 16:16

Gegeben: ger. Graph G und $c : E(G) \to \mathbb{R}$. Gesucht: Branching B in G mit minimalem Gewicht, wobei

$$c(B) := c(E(B)) = \sum_{e \in E(B)} c(e).$$

Was ist ein Branching?
(1) der zugrundeliegende Graph ist ein Wald
(2) $\left|\delta^-(v)\right| \leqslant 1 \ \forall v \in V(G)$

Idee: Finde zunächst max. gewichtetet Teilgraphen B_0 mit Eigenschaft (2). Wenn (1) verletzt ist, enthält der B_0 zugrunde liegende ungerichtete Graph einen Kreis, also auch B_0 selbst.

Lemma 1.22 (Karp, 1972)**.** Sei B_0 ein max. gewichteter Teilgraph von G mit

$$\left|\delta^-_{B_0}(v)\right| \leqslant 1 \quad \forall v \in V(B_0) = V(G).$$

Dann existiert ein maximales gewichtetes Branching B in G, so dass fr jeden Kreis C in B_0 gilt:

$$|E(C) \backslash E(B)| = 1.$$

Algorithmus 5 : Edmonds Barching Algorithmus

Eingabe : ger. Graph G, $c : E(G) \to \mathbb{R}_+$
Ausgabe : ein Branching B maximalen Gewichts

1 $i := 0, G_0 := G, c_0 := c$
2 sei B_i eine max. gewichteter Teilgraph von G_i mit $\left|\delta^-_{B_i}(v)\right| \leqslant 1$ für alle $v \in V(B_i)$
3 **if** B_i *enthält keinen Kreis* **then**
  $B := B_i$
  goto 5
4 Sei $\mathcal{C}$ die Menge der Kreise in B_i. Kontrahiere diese, das heißt:
$V(G_{i+1}) := \mathcal{C} \cup (V(G_i) \backslash \bigcup_{C \in \mathcal{C}} V(C))$
fr $e = (v, w) \in E(G_i)$ sei $e' = (v', w')$ und $\Phi_{i+1}(e') = e$, wobei

$$v' = \begin{cases} c & \text{falls } v \in V(C), C \in \mathcal{C} \\ v & \text{sonst} \end{cases}, \qquad w' = \begin{cases} c & \text{falls } w \in V(C), C \in \mathcal{C} \\ w & \text{sonst} \end{cases}.$$

Setze $E(G_{i+1}) = \left\{e' = (v', w' \mid e \in E(G_i), v' \neq w'\right\}$
Fr $e = (v, w) \in E(G_i)$ mit $e'(v', w') \in E(G_{i+1})$ setzen wir die Kosten als:

$$c_{i+1}(e') = \begin{cases} c_i(e) & \text{falls } w' \notin C \ \forall C \in \mathcal{C} \\ c_i(e) - c_i(\alpha(e, C)) + c_i(e_C) & \text{falls } w' = c \in \mathcal{C} \end{cases}.$$

wobei $\alpha(e, C)$ die Kante von C ist, die denselben Endpunkt wie e hat und e_C die billigste Kante von C ist.
Setze $i := i + 1$
goto 2
5 **while** $i > 0$ **do**
  $B' := (V(G_{i-1}), \{\Phi_i(e) \mid e \in E(B)\}$
  for *each Kreis* C *in* B_{i-1} **do**
    if $\exists e \in \delta^-_{B'}(V(C))$ **then**
      $E(B') := E(B') \cup (E(C) \backslash \{\alpha(e, C)\})$
    else
      $E(B') := E(B') \cup (E(C) \backslash \{e_C\})$

Algorithmus 6 : Dijkstras Algorithmus

Eingabe : gerichteter Graph G, $c : E(G) \to \mathbb{R}_+, s \in V(G)$
Ausgabe : Kürzeste Wege von s zu allen $v \in V(G)$ und deren Längen.
Genauer: $l(v) = \text{dist}_{(G,c)}(s, v)$ und $p(v) :=$ Vorgänger von v auf einem kürzesten s-v-Weg

Def `Dijkstras Algorithmus`(G, c, s)**:**
1 $l(s) := 0, l(v) := \infty$ für $v \in V(G) \backslash \{s\}$, $R := \varnothing$
2 Finde $v \in V(G) \backslash R$ mit $l(v)$ minimal
3 $R := R \cup \{v\}$
4 **for all** $(v, w) \in \delta^+(v)$ *mit* $w \notin R$ **do**
  if $l(w) > l(v) + c(e)$ **then**
    $l(w) := l(v) + c(e)$
    $p(w) := v$
5 **if** $R \neq V(G)$ **then**
  go to (3)

Satz 1.23. `Dijkstras Algorithmus` ist korrekt.

Beweis. Folgened Invarianten gelten stets :
(a) $\forall v \in V(G)\setminus\{s\}$ mit $l(v) < \infty$ ist $p(v) \in R, l(p(v)) + c((p(v), v)) = l(v)$ und die Folge $v, p(v), p(p(v)), \ldots$ enthält s.
(b) $\forall v \in R : l(v) = \text{dist}_{(G,c)}(s, v)$

(a) und (b) gelten trival nach (1). In (4) wird $l(w)$ auf $l(v) + c(e)$ und $p(w)$ auf v gesetzt, aber nur, wenn $v \in R$. TODO: fertig machen! □

Bemerkung. Naive Implementierung hat Laufzeit $O(m + n^2)$. Besser: mit F-Heaps erhalten wir Laufzeit $O(m+n\log n)$ (speichere stets alle $v \in V(G)\setminus R$ mit $\delta^+(R) \cap \delta^-(v) \neq \varnothing$, mit Schlüssel $l(v)$. Dann haben wir n insert, n deletemin und $\leqslant m$ decreasekey Operationen)

Kürzeste Wege in ungerichtetet Graphen G mit $c(e) \geqslant 0$. Kann darauf zurückgeführt werden: ersetze jede Kante $e = \{v, w\}$ durch zwei Kanten (v, w) und (w, v) mit demselben Gewicht.

Bemerkung. Der Algorithmus funktioniert auch auf Graphen mit konservativen Kantengewichten, für allgemeiner Gewichte gibt es folgenden Algorithmus:

Algorithmus 7 : Moore-Bellman-Ford-Algorithmus

Eingabe : ein gerichteter Graph G, Kostenfunktion $c : E(G) \to \mathbb{R}$ und Knoten $s \in V(G)$
Ausgabe : Kürzeste Wege von s zu allen $v \in V(G)$ und deren Längen oder ein negativer Kreis

Def `Moore-Bellman-Ford-Algorithmus`(G, c, s)**:**
1 $l(s) := 0$, $l(v) := \infty$ $\forall v \in V(G)\setminus\{s\}$
2 **for** $i = 1$ **to** $n - 1$ **do**
  for all $(v, w) \in E(G)$ **do**
    if $l(w) > l(v)$ **then**
      $l(w) := l(v) + c(e)$
      $p(w) := v$
  if $\exists$ *Kante* $e = (v, w) \in E(G)$ *mit* $l(w) > l(v) + c(e)$ **then**
    $x_n := w, x_{n-1} := v, x_{n-i-1} := p(x_{n-i})$ für $i = 1, \ldots, n-1$ und gib einen Kreis c in $(V(G), \{(x_{i-1}, x_i) : i = 1, \ldots, n\}$ zurück.

Satz 1.24 (Moore, Bellman, Ford 1956-59)**.** Der `Moore-Bellman-Ford-Algorithmus` ist korrekt und hat Laufzeit $O(nm)$.

Beweis. Sei stets $k(v)$ die Iteration, in der $l(v)$ zuletzt (auf den aktuellen Wert) geändert wurde. □

Lecture 9: Maximale Flüsse

Di 24 Nov 2020 15:46

2 Maximale Flüsse

Max Flow Problem:
Eingabe ein **Netzwerk** (G, u, s, t) , d.h. G ist gerichteter Graph, $u : E(G) \to \mathbb{R}_+$ ist Kostenfunktion und $s, t \in V(G)$ mit $s \neq t$.
Ausgabe: finde einen **s-t-Fluss**, d.h. ein $f : E(G) \to \mathbb{R}_+$ mit $f(e) \leqslant u(e)$ $\forall e \in E(G)$ und

$$\operatorname{ex}_f(v) := f(\delta^-(v)) - f(\delta^+(v)) = \begin{cases} 0 & \forall v \neq s, t \\ \geqslant 0 & \text{für } v = s \end{cases}.$$

mit maximalem Wert $\operatorname{value}(f) := \operatorname{ex}_f(t) = -\operatorname{ex}_f(s)$

Satz 2.1 (Max-Flow-Min-Cut-Theorem)**.** Sei (G, u, s, t) ein Netzwerk. Dann ist der maximale Wert eines s-t- Flusses gleich der maximalen Kapazität eines s-t-Schnittes, d.h.

$$\min\left\{u(\delta^+(R)) : s \in R \subseteq V(G)\backslash\{t\}\right\}.$$

Beweis. Sei $R \subseteq V(G)$ mit $s \in R$ und $t \notin R$ und f ein s-t-Fluss. Dann ist

$$\begin{aligned}
\operatorname{value}(f) = \operatorname{ex}_f(t) &= f(\delta^-(t)) - f(\delta^+(t)) \\
&= f(\delta^+(V(G)\backslash\{t\})) - f(\delta^-(V(G)\backslash\{t\})) \\
&= f(\delta^+(R)) - f(\delta^-(R)) + \sum_{v \in (V(G)\backslash\{t\}) R} \left(\underbrace{f(\delta^+(v)) - f(\delta^-(v))}_{=0} \right) \\
&= f(\delta^+(R)) - f(\delta^- R)) \\
&\leqslant u(\delta^+(R)) = 0
\end{aligned}$$

Damit ist die notwendige Bedingung gezeigt. Um Gleichheit zu zeigen, sei f eine maximaler Fluss (d.h. Fluss mit maximalem Wert). Dieser existiert, da ex eine lineare Funktion (also stetig) über einer kompakten Menge maximiert wird.
Wir definieren folgenden Graphen, den **Residualgraphen** G_f (für einen beliebigen s-t-Fluss) durch:

$$V(G_f) = V(G), \quad E(G_f) := \{e \in E(G) : f(E) < u(e)\} \sqcup \{e = (w, v) : e = (v, w) \in E(G), f(e) > 0\}.$$

Sei R die Menge der von s aus in G_f (für maximalen Fluss f) erreichbaren Knoten. Falls $t \in R$, existiert also ein s-t-Weg P in G_f (augmentierender Weg), so können wir einen Fluss mit größerem Wert als f berechnen, durch:
Sei $\gamma := \min_{e \in E(P)} u_f(e)$, wobei

$$u_f(e) = \begin{cases} u(e) - f(e) & e \in E(G) \\ f(e) & \text{Rückwärtskanten} \end{cases}.$$

Für $e \in E(P) \cap E(G)$ setze $f'(e) := f(e) + \gamma$ und für $\overleftarrow{e} \in E(P)$ mit $e \in E(G)$ setze $f'(e) := f(e) - \gamma$. Für andere Kanten bleibt $f'(e) = f(e)$. Nun ist f' ein s-t-Fluss mit $\operatorname{value}(f') = \operatorname{value}(f) + \gamma$. Dies kann also nicht passieren, da wir f maximal gewählt haben.
Falls $t \notin R$, so ist

$$\operatorname{value}(f) = f(\delta^+(R)) - f(\delta^-(R)) = u(\delta^+(R)) - 0.$$

denn $f(e) = u(e)$ für alle $e \in \delta^+(R)$, da $e \notin E(G_f)$ und $f(e) = 0$ $\forall E \in \delta^-(R)$, da $\overrightarrow{e} \notin E(G_f)$ $\square$