ADS: Algorithmen und Datenstrukturen 2

Teil 4

Prof. Dr. Gerhard Heyer

Institut für Informatik
Abteilung Automatische Sprachverarbeitung
Universität Leipzig

24. April 2019

[Letzte Aktualisierung: 24/04/2019, 08:48]

Erste Zwischenbilanz

Wir haben bisher kennengelernt

- das Konzept und einige Hintergründe zur Kompression
- Graphen (ungerichtet, gerichtet, gewichtet), Spannbäume, transitive Hülle, Sortierung und Suche in Graphen, Zusammenhangskomponenten und Zentralität

Dabei sind im wesentlichen zwei Arten von Algorithmen zum Einsatz gekommen,

- Greedy-Algorithmen (Huffmann-Codierung, Kruskal)
- dynamische Programmierung (Warshall)

Heute soll das Konzept der Greedy-Algorithmen vertieft werden

Greedy-Algorithmen

Annahme: Gewichtsfunktion w misst Güte einer Lösung (oder Teillösung).
Ziel: Konstruiere Lösung mit maximalen Gewicht (w).

1. **Starte** mit leerer Lösung
2. **In jedem Schritt:** Erweitere die bisher konstruierte Teillösung. Unter k möglichen Erweiterungen, die zu Teillösungen $L_1, \ldots, L_k$ führen, wähle L_i, so dass $w(L_i)$ maximal. (greedy = *gierig*)

Also bedeutet "greedy": *Nimm immer das größte Stück*.

Oft wird die Greedy-Strategie im Sinne einer *Heuristik* verwendet.
(*Heuristik* bedeutet dabei: i.d.R. wird eine brauchbar gute Lösung erzeugt)

Greedy nutzt nur *lokale* Maxima. Es gibt aber auch Probleme, für die das Greedy-Prinzip optimal ist, d.h. tatsächlich immer zu einer optimalen Lösung führt.

Unabhängigkeitssystem

- Sei E eine endliche Menge und $\mathcal{M}$ eine Menge von Teilmengen von E, also $\mathcal{M} \subseteq \mathcal{P}(E)$. Dann heißt $(E, \mathcal{M})$ *Mengensystem*.
- Ein Mengensystem $(E, \mathcal{M})$ heißt *Unabhängigkeitssystem*, wenn $\emptyset \in \mathcal{M}$ und zu jeder Menge in $\mathcal{M}$ auch alle Teilmengen enthalten sind, also gilt:

  $$\text{Aus } A \in \mathcal{M} \text{ und } B \subseteq A \text{ folgt } B \in \mathcal{M}.$$

- Ein Unabhängigkeitssystem ist *abgeschlossen* unter Bildung von Teilmengen. Man kann eine in $\mathcal{M}$ gefundene Menge "kleiner" machen und bleibt in $\mathcal{M}$. Umgekehrt gilt, dass eine in $\mathcal{M}$ gefundene Menge immer nur durch die Bildung von Mengen aus $\mathcal{M}$ erzeugt worden ist.

Gewichtsfunktion und Optimierungsproblem

- Sei $(E, \mathcal{M})$ ein Mengensystem
- Sei w eine Funktion $w : E \to \mathbb{R}^+$, genannt *Gewichtsfunktion*
- Erweitere Notation (Gewicht einer Menge $A \in \mathcal{M}$):

$$w(A) := \sum_{e \in A} w(e)$$

- Optimierungsproblem: Finde $A^* \in \mathcal{M}$, das die Gewichtssumme maximiert. Für alle $A \in \mathcal{M}$ soll also gelten:

$$w(A^*) \geq w(A).$$

Kanonischer Greedy-Algorithmus

. . . für Unabhängigkeitssystem $(E, \mathcal{M})$ und Gewichtsfunktion $w : E \to \mathbb{R}^+$:

$A \leftarrow \emptyset$;
for $e \in E$ *in Reihenfolge absteigender Gewichte* $w(e)$ **do**
  if $A \cup \{e\} \in \mathcal{M}$ **then**
    $A \leftarrow A \cup \{e\}$
  end
end
return A

Liefert der Algorithmus "Greedy" immer die optimale Lösung?
(*immer* $\hat{=}$ für beliebige Mengensysteme und Gewichte?)

Beispiel: Greedy nicht immer optimal

- $E = \{a, b, c\}$
- $\mathcal{M} = \{\, \emptyset, \{a\}, \{b\}, \{c\}, \{b, c\} \,\}$
- $w(a) = 3,\ w(b) = w(c) = 2$
- Dazu liefert der kanonische Greedy-Algorithmus die Lösung $\{a\}$ mit Gewichtssume $w(a) = 3$
- Die optimale Lösung ist $\{b, c\}$ mit Gewichtssumme $w(b) + w(c) = 4$.

Gesucht: Welche zusätzliche Bedingung muss erfüllt sein, damit Greedy optimal ist?

Matroid

Definition (Matroid)
Ein Unabhängigkeitssystem $(E, \mathcal{M})$ heißt *Matroid*, wenn für alle $A, B \in \mathcal{M}$ die folgende *Austauscheigenschaft* gilt:

$$\text{Ist } |A| > |B|, \text{ dann gibt es } x \in A \setminus B, \text{ so dass } B \cup \{x\} \in \mathcal{M}.$$

Beobachtung: Jedes Paar (A,B) mit $B \subseteq A$ erfüllt die Austauscheigenschaft.

Matroid garantiert Greedy-Optimalität

Satz (Greedy-Optimalität):
Sei $(E, \mathcal{M})$ ein Unabhängigkeitssystem

Für jede beliebige Gewichtsfunktion $w : E \to \mathbb{R}^+$ berechnet der kanonische Greedy-Algorithmus eine optimale Lösung

$$\Updownarrow$$

$(E, \mathcal{M})$ ist ein Matroid.

Greedy-Optimalität - Beweis (vgl. Schöning)

$\Rightarrow$

Nehmen wir an, die Austauscheigenschaft gelte nicht. Dann gibt es A und B in $\mathcal{M}$ mit $|A| > |B|$, so dass für alle $x \in A \setminus B$ gilt $B \cup \{x\} \notin \mathcal{M}$.
Wir definieren folgende Gewichtsfunktion $w : E \to \mathbb{R}^+$

$$w(x) = \begin{cases} \text{r}, & \text{x} \in B \\ \text{r}, & \text{x} \in A \setminus B \\ 0, & \textit{sonst} \end{cases}$$

Sei $w(A) = r = |A|$ und wegen $|A| > |B|$, $w(B) = |B| = r - 1$.
Der Greedy-Algorithmus wählt dann eine Menge T mit $B \subseteq T$ und $T \cap A \setminus B) = \emptyset$. Also ist $w(T) = w(x \in b) + |B| = 2r - 1$. Wählt man stattdessen eine Lösung $T' \supseteq A$, so hat diese den Wert $w(T') \geq r + |B| = 2r$. Der Greedy-Algorithmus liefert also nicht die optimale Lösung!

Greedy-Optimalität - Beweis (vgl. Schöning)

$\Leftarrow$

Sei (E, U) ein Matroid mit Gewichtsfunktion w, welche die Elemente von $E = \{e_1, \ldots, e_n\}$ absteigend anordnet, $w(e_1) \geq \cdots \geq w(e_n)$ und $T = \{e_{i_1}, \ldots, e_{i_k}\}$ die vom Greedy-Algorithmus gefundene Lösung.
Nehmen wir an, $T' = \{e_{j_1}, \ldots, e_{j_k}\}$ sei eine bessere Lösung mit $w(T') > w(T)$. Dann muss es einen Index μ geben mit $w(e_{j_\mu}) > w(e_{i_\mu})$ und μ dem kleinsten Index.
Es sei $A = \{e_{i_1}, \ldots, e_{i_{\mu-1}}\}$ und $B = \{e_{j_1}, \ldots, e_{j_\mu}\}$ und $|A| < |B|$. Wegen der **Austauscheigenschaft** muss es ein Element $e_{j_\sigma} \in B - A$ geben, so dass $A \cup \{e_{j_\sigma}\} \in \mathcal{M}$. Es gilt aber $w(e_{j_\sigma}) \geqslant w(e_{j_\mu}) > w(e_{i_\mu})$, also hätte der Greedy-Algorithmus das Element e_{j_σ} bereits vor e_{i_μ} auswählen müssen. Widerspruch.

Nutzen - Algorithmendesign

Gegeben eine **Optimierungsaufgabe**,

- strukturiere die Anwendung in ein passend gewähltes Teilmengensystem (E, M)
  - Lösungen des Optimierungsproblems müssen Lösungen des zu (E, M) gehörenden Maximierungs- bzw. Minimierungsproblems sein
- überprüfe, ob (E, M) die Matroid-Eigenschaft besitzt
- falls ja, wende den kanonischen Greedy-Algorithmus an
  - das erfordert eine Prozedur, mit der überprüft werden kann, ob eine gegebene Teilmenge $A \subseteq M$ eine Lösung bzw. Teillösung des Optimierungsproblems ist

Auftragsplanung

Gegeben:

- Menge E von n Aufträgen mit jeweils gleichem Zeitbedarf (ein Tag)
- Zu jedem Auftrag $x \in E$: Gewinn $w(x) > 0$ (Gewichtsfunktion).
- Anzahl Tage $d(x)$, bis zu dem der Auftrag x abgeschlossen sein muss

Gesucht: Menge $L \subseteq E$ von Aufträgen, so dass

a) der Gesamtgewinn $\sum_{x \in L} w(x)$ maximal ist, und

b) L sich so sortieren lässt, dass jeder Auftrag vor seinem Abschlusstermin erledigt wird.

Auftragsplanung: Beispiel

Auftragsmenge $E = \{a, b, c, d, e\}$

Auftrag x	Wert w(x)	Termin d(x)
a	13	2
b	7	1
c	9	1
d	3	2
e	5	1

Auftragsplanung: Kanonischer Greedy-Algorithmus

Ordne Aufträge in E absteigend nach Gewinn
Jetzt der kanonische Greedy-Algorithmus wie folgt:

$A \leftarrow \emptyset$;
for $e \in E$ *in Reihenfolge absteigenden Gewinns* $w(e)$ **do**
  if $A \cup \{e\} \in \mathcal{M}$ **then**
    $A \leftarrow A \cup \{e\}$
  end
end
return A

In A seien die Aufträge nach ihrem Abschlusstermin geordnet:
$d_1 \leq d_2 \leq \cdots \leq d_i$

Auftragsplanung: Lösungsraum A

Lösung	Gewinn
$\emptyset$	0
$\{c\}$	9
$\{b\}$	7
$\{e\}$	5
$\{a\}$	13
$\{d\}$	3
$\{c, a\}$	22
$\{c, d\}$	12
$\{b, a\}$	20
$\{b, d\}$	10
$\{e, a\}$	18
$\{e, d\}$	8
$\{a, d\}$	16

Auftragsplanung

(E,M) ist ein Matroid mit A $= \{(d_1, p_1), (d_2, p_2), \ldots, (d_k, p_k)\}$

Es ist $A \in M$ genau dann, wenn A eine zulässige Lösung darstellt.

- $\mathcal{M}$ ist Unabhängigkeitssystem
- Es gilt die Austauscheigenschaft

Die Liste L der optimalen Lösung ergibt sich aus
$w_{\max(}\{(\mathrm{d}_1, p_1), (d_2, p_2)\} = \mathrm{w}_{\max(}(\mathrm{d}_1, p_1) + w_{\max(}(\mathrm{d}_2, p_2)$

Greedy Konstruktion des Huffmann-Baums

Die Buchstaben eines Alphabets $E = a_1, a_2, \ldots, a_n$ seien so durchnummeriert, dass $p(a_1) \leq p(a_2) \leq \cdots \leq p(a_n)$ (mit $w(a_i) = p(a_i)$).

1. Erstelle für jeden Buchstaben einen einzelnen Knoten und notiere am Knoten die Häufigkeit.
2. Wiederhole die folgenden Schritte so lange, bis nur noch ein Baum übrig ist:
   - Wähle die m Teilbäume mit der geringsten Häufigkeit in der Wurzel, bei mehreren Möglichkeiten die Teilbäume mit der geringsten Tiefe.
   - Fasse diese Bäume zu einem neuen (Teil-)Baum zusammen.
   - Notiere die Summe der Häufigkeiten in der Wurzel.

Huffmann-Codebaum

Ein Huffmann-Baum selber ist kein Matroid, kann aber in eine isomorphe Mengenstruktur abgebildet werden.

- E = Menge der Buchstaben eines Alphabets
- Gewichtsfunktion $w(a_i) = p(a_i)$ und $w(a_i + a_j) = p(a_i) + p(a_j)$
- M = Menge der Teilbäume (a_i, a_j)
- Unabhängigkeitssystem, weil Teilbäume eines binären Teilbaums selber binäre Bäume sind
- Austauscheigenschaft erfüllt, weil jeder Knoten als Menge seiner Teilbäume repräsentiert wird, alle Söhne also stets Teilmengen ihrer Väter sind

Optimalität des Huffmann-Codebaums

- In einem optimalen Codebaum B haben die beiden Blätter a_1 und a_2 einen gemeinsamen Vaterknoten.
- Für $((a_1, a_2), (a_3), \ldots, (a_n))$ löst der Huffmann-Algorithmus die Aufgabenstellung mit $p(a_1, a_2)) = p(a_1) + p(a_2)$ optimal
- Die Expansion jedes Teilbaums, der nach der obigen Regel konstruiert worden ist, ist wieder optimal

. . . und noch mal der Kruskal-Algorithmus

Kruskal-Algorithmus (1956) für minimalen Spannbaum auf ungerichtetem zusammenhängendem Graph (V, E) mit Kantengewicht $w(e)$ für $e \in E$

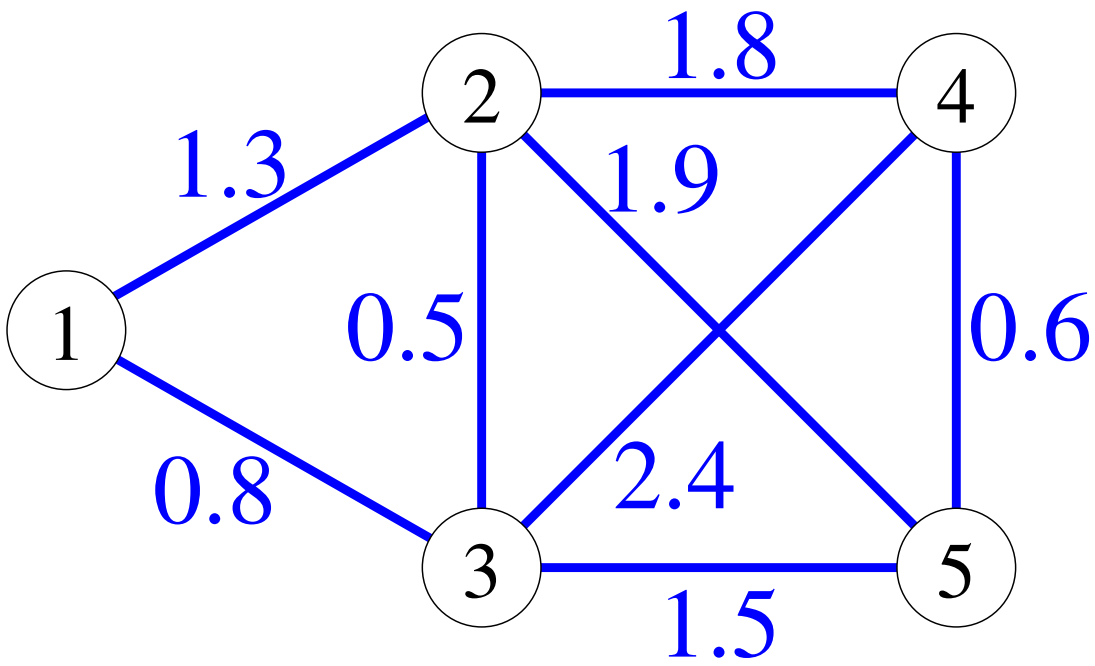
1. Erzeuge Liste Q mit der Kantenmenge E aufsteigend sortiert nach Gewicht (kleinstes zuerst). Initialisiere F als leere Menge.
2. Entferne vorderste Kante $e = \{u, v\}$ (mit kleinstem Gewicht) aus Q
3. Falls (V, F) keinen Pfad zwischen u und v enthält:

$$F := F \cup \{e\}$$

4. Falls Q nicht leer, zurück zu 2.
5. Ausgabe F.

Greedy-Algorithmus zur *Minimierung* der Summe der Kantengewichte

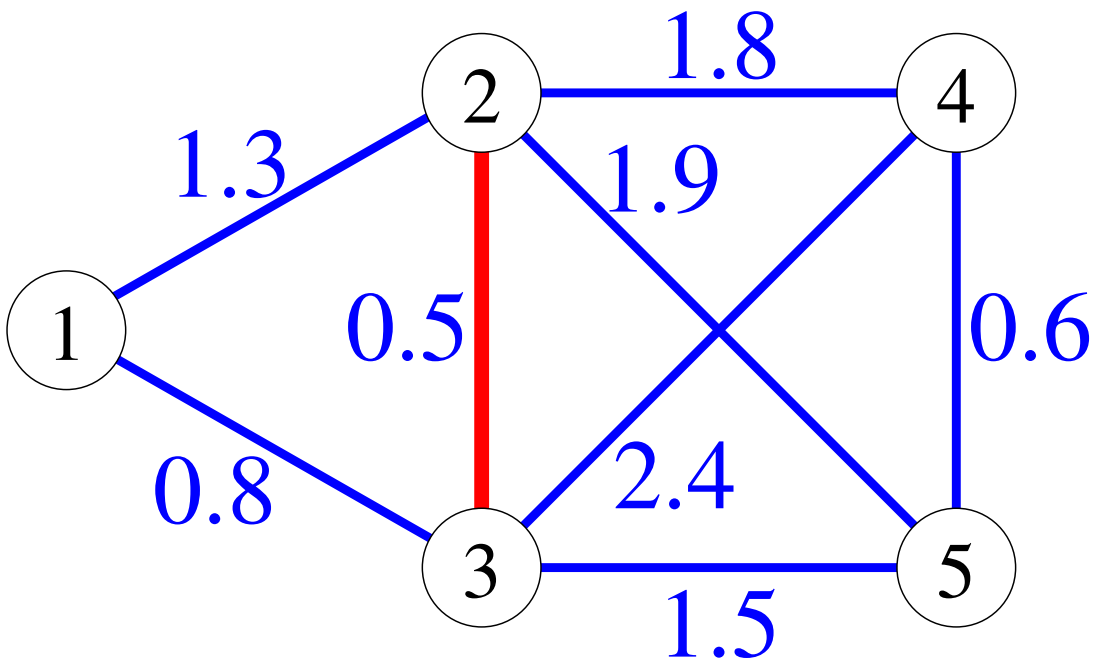
Beispiel-Lauf: Kruskal-Algorithmus



$Q = [\{2,3\},\{4,5\},\{1,3\},\{1,2\},\{3,5\},\{2,4\},\{2,5\},\{3,4\}]$

$F = \{\}$

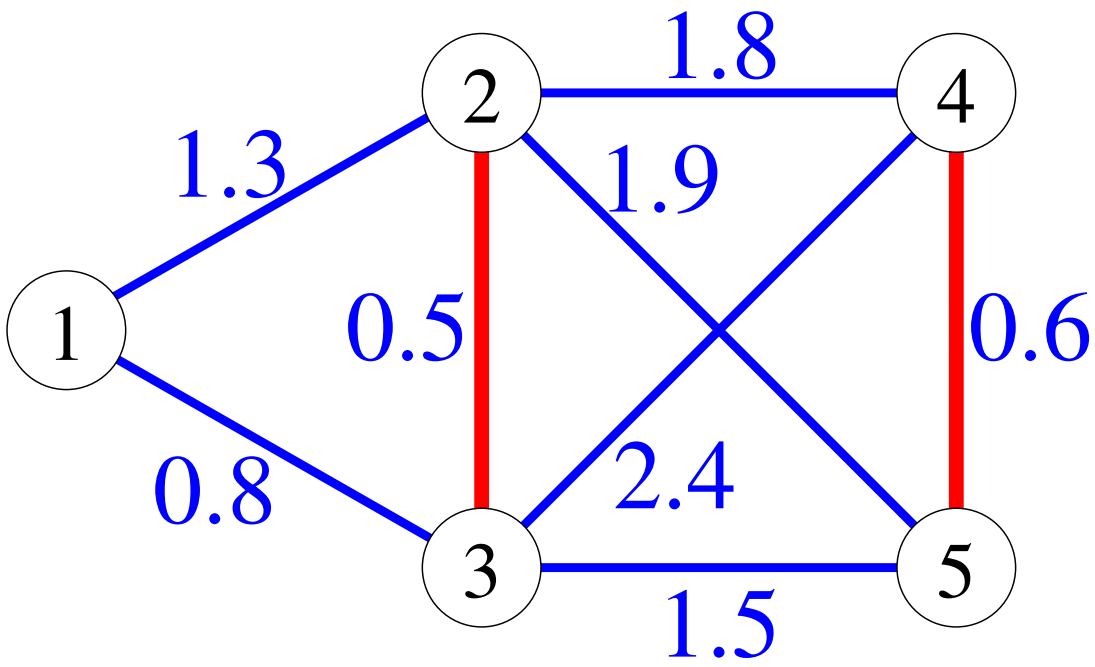
Beispiel-Lauf: Kruskal-Algorithmus



$Q = [\{4,5\}, \{1,3\}, \{1,2\}, \{3,5\}, \{2,4\}, \{2,5\}, \{3,4\}]$

$F = \{\{2,3\}\}$

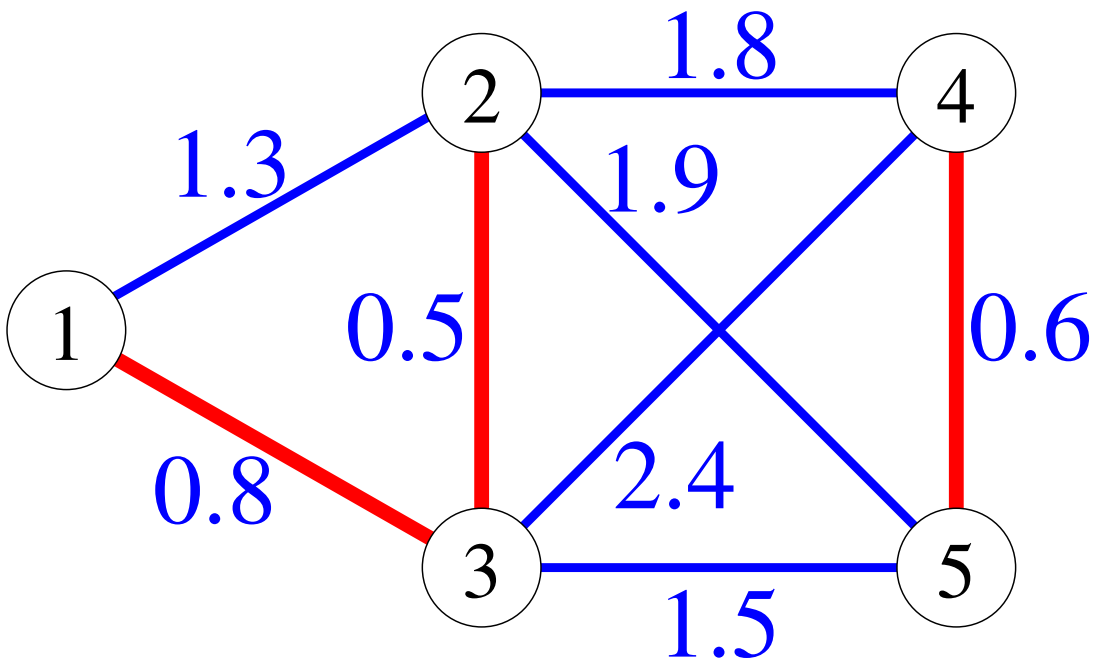
Beispiel-Lauf: Kruskal-Algorithmus



$Q = [\{1,3\},\{1,2\},\{3,5\},\{2,4\},\{2,5\},\{3,4\}]$

$F = \{\{2,3\},\{4,5\}\}$

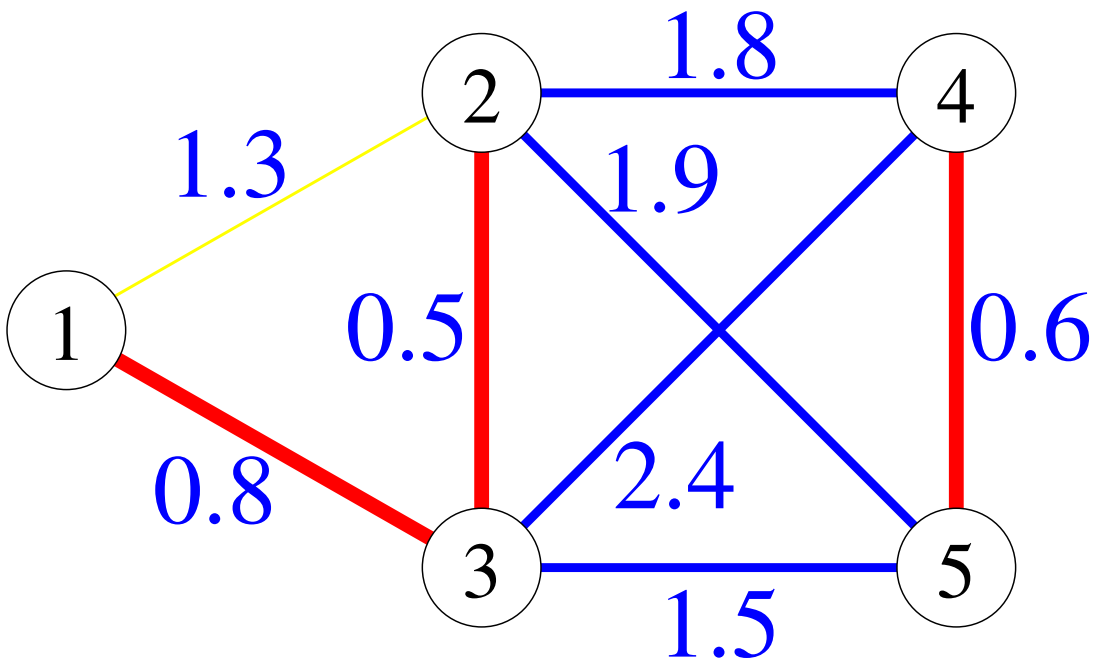
Beispiel-Lauf: Kruskal-Algorithmus



$Q = [\{1,2\},\{3,5\},\{2,4\},\{2,5\},\{3,4\}]$

$F = \{\{2,3\},\{4,5\},\{1,3\}\}$

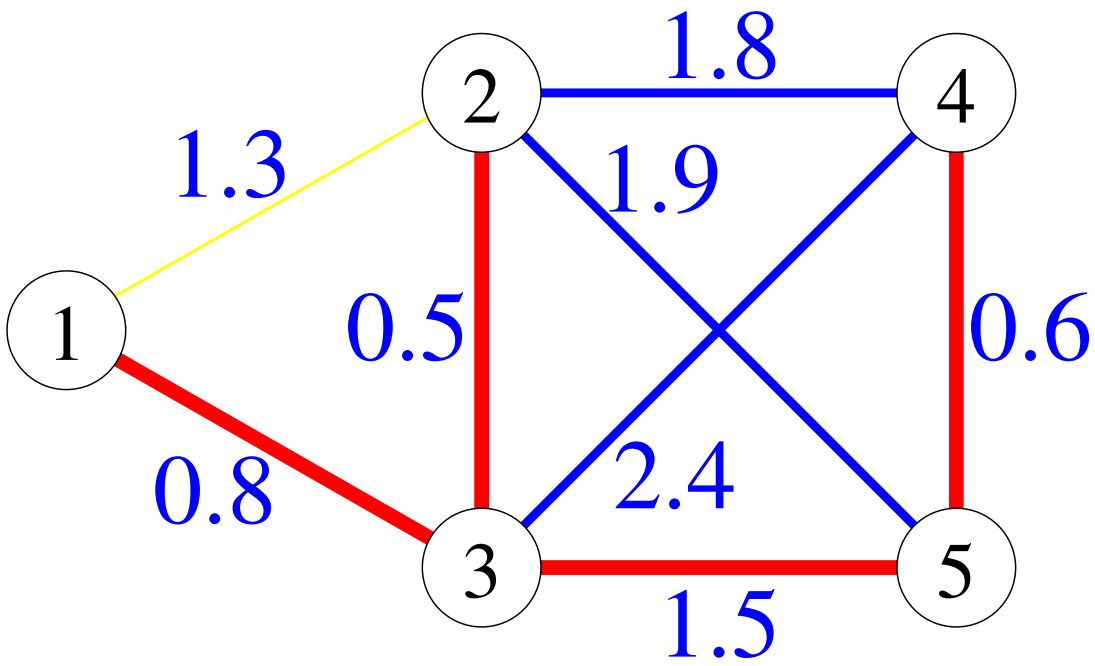
Beispiel-Lauf: Kruskal-Algorithmus



$Q = [\{3,5\},\{2,4\},\{2,5\},\{3,4\}]$

$F = \{\{2,3\},\{4,5\},\{1,3\}\}$

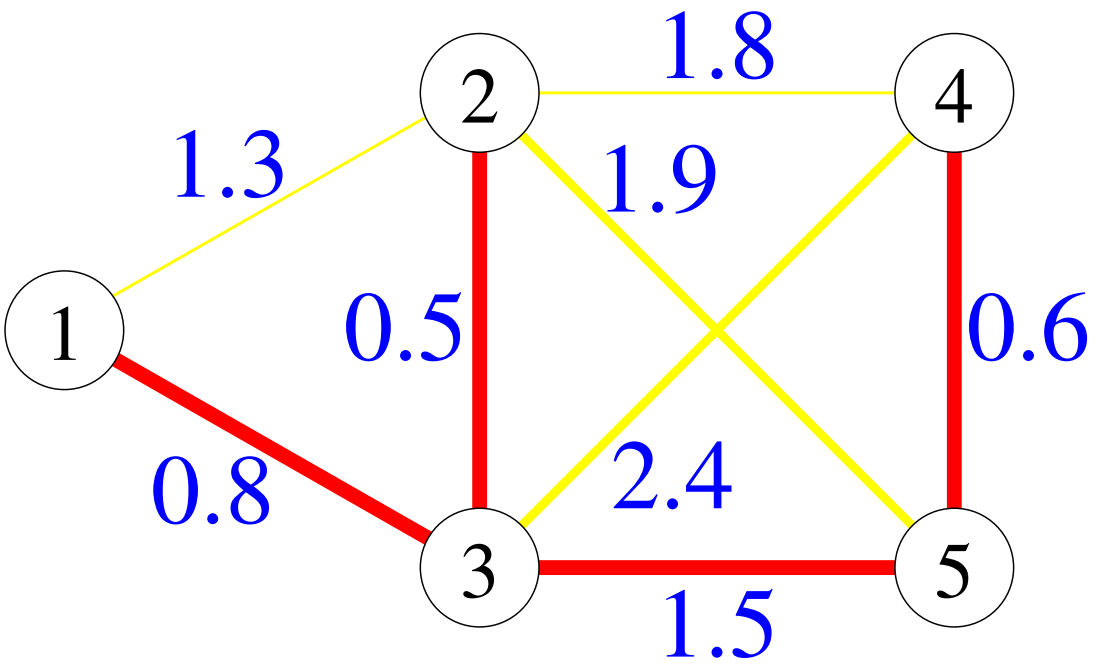
Beispiel-Lauf: Kruskal-Algorithmus



$Q = [\{2,4\},\{2,5\},\{3,4\}]$

$F = \{\{2,3\},\{4,5\},\{1,3\},\{3,5\}\}$

Beispiel-Lauf: Kruskal-Algorithmus



$Q = []$

$F = \{\{2,3\},\{4,5\},\{1,3\},\{3,5\}\}$

Kruskal: Korrektheit (1)

- E = Kantenmenge, $\mathcal{M}$ = Lösungsraum (Menge der möglichen Spannbäume)
- Gewichtsfunktion $w : E \to \mathbb{R}$ = Kantengewichte
- $\mathcal{M}$ enthält $F \subseteq E$ genau dann wenn (V, F) ein ungerichteter, kreisfreier, kantenbewerteter Graph ist

Wende Satz über Greedy-Optimalität an:

Kruskal findet optimale Lösung für beliebige Kantengewichte

$$\Updownarrow$$

$(E, \mathcal{M})$ ist ein Matroid.

Kruskal *minimiert*, während der kanonische Greedy-Algorithmus *maximiert*. Der Algorithmus ist direkt anwendbar nach Transformation der Kantengewichte $w'(e) = c - w(e)$. Die Konstante c ist so groß zu wählen, dass alle $w'(e)$ positiv werden.

Kruskal: Korrektheit (2)

Wir nehmen als Startbedingung des Kruskal-Algorithmus an, dass

$$\mathrm{w}(\mathrm{e}_1) < w(e_2) \cdots < w(e_k)$$

- $(E, \mathcal{M})$ ist ein *Unabhängigkeitssystem*, denn Teilgraphen von azyklischen Graphen sind azyklisch.
- Die *Austauscheigenschaft* ist gegeben, weil jeder Teilgraph als Menge seiner Kanten repräsentiert werden kann, wobei

$$w_{\mathrm{min}}(\{e_1, e_2, \ldots, e_k\}) = w_{\mathrm{min}}(\mathrm{e}_1) + \mathrm{w}_{\mathrm{min}}(\{\ \mathrm{e}_2, \ldots, e_k\})$$