Laufzeit

Finden eines Matchings maximaler Kardinalität dauert nur $O(|E| \cdot \min\{|V_1|, |V_2|\})$ mit der Ford–Fulkerson–Methode.

- Der Fluß ist höchstens $f^* = \min\{|V_1|, |V_2|\}$.
- Finden eines Pfads dauert $O(|E|)$.

Variante der Ford–Fulkerson–Methode

Dieser Algorithmus funktioniert bei ganzzahligen Kapazitäten:

Algorithmus

$K \leftarrow 2^{\lfloor \log_2(\max\{ c(u,v) \mid (u,v) \in E \}) \rfloor}$
$f \leftarrow 0$
while $K \geq 1$ **do**
  while es gibt einen augmentierenden Pfad p mit $c_f(p) \geq K$ **do**
    augmentiere f entlang p
  $K \leftarrow K/2$
return f

Die Laufzeit ist $O(|E|^2 \log K)$.
$\Rightarrow$ vergleiche mit $O(|E| f^*)$.

Variante der Ford–Fulkerson–Methode

Theorem

Die Laufzeit dieser Variante beträgt $O(|E|^2 \log C)$, *wobei* $C = \max\{ c(u,v) \mid (u,v) \in E \}$.

Beweis.

- Die Restkapazität eines minimalen Schnitts ist stets höchstens $2K|E|$.
- Für jedes K gibt es nur $|E|$ Augmentierungen
- Es gibt $O(\log C)$ verschiedene K

□

Der Edmonds–Karp–Algorithmus

Die Ford–Fulkerson–Methode kann sehr langsam sein, auch wenn das Netzwerk klein ist.

Der Edmonds–Karp–Algorithmus ist polynomiell in der Größe des Netzwerks.

Algorithmus

Initialisiere Fluß f zu 0
while es gibt einen augmentierenden Pfad **do**
  finde einen kürzesten augmentierenden Pfad p
  augmentiere f entlang p
return f

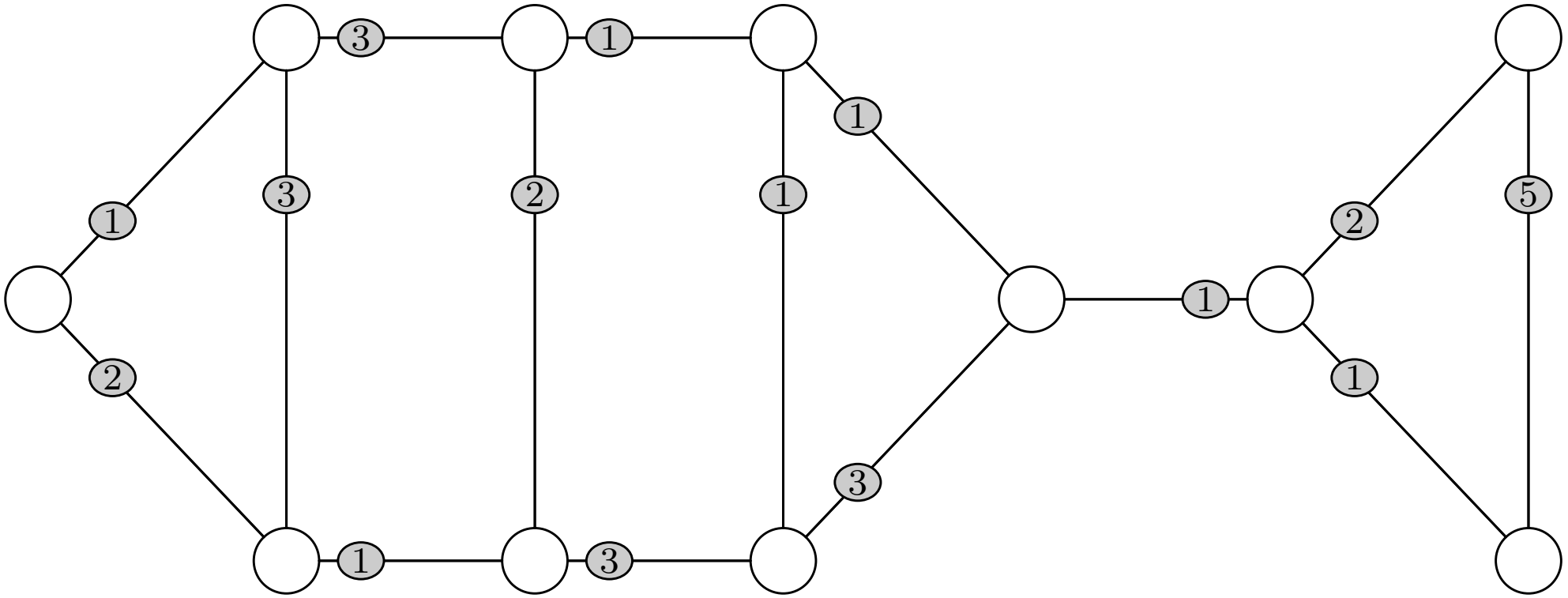
Unterschied: Es wird ein kürzester Pfad gewählt

Der Edmonds–Karp–Algorithmus

Algorithmus

for each edge $(u, v) \in E$ **do**
  $f(u, v) \leftarrow 0$
  $f(v, u) \leftarrow 0$
while there exists a path from s to t in G_f **do**
  $p \leftarrow$ a shortest path from s to t in G_f
  $c_f(p) \leftarrow \min\{ c_f(u, v) \mid (u, v) \text{ is in } p \}$
  for each edge (u, v) in p **do**
    $f(u, v) \leftarrow f(u, v) + c_f(p)$
    $f(v, u) \leftarrow -f(u, v)$
return f

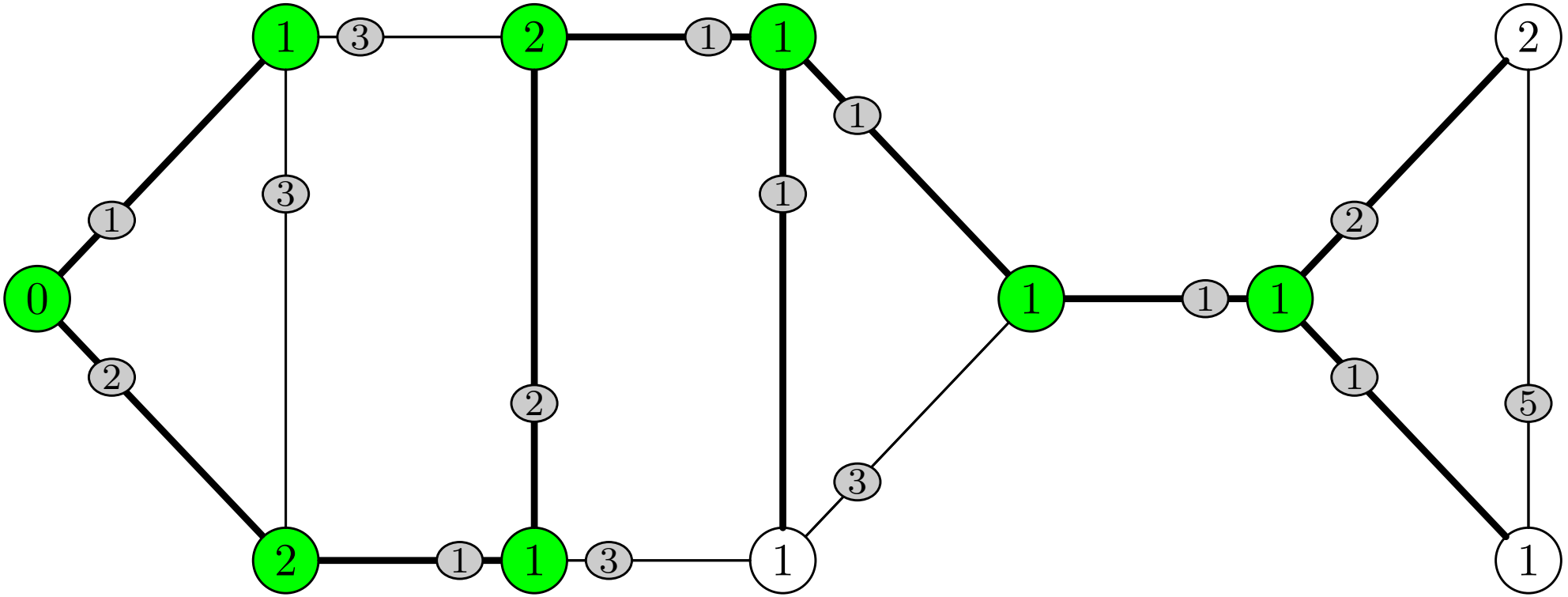
Minimale Spannbäume



Eingabe: Ungerichteter Graph mit Kantengewichten

Ausgabe: Ein Baum, der alle Knoten enthält und minimales Kantengewicht hat

Der Algorithmus von Prim – Beispiel



- Beginne mit leerem Baum (nur Wurzel)
- Hänge wiederholt billigste Kante an ohne einen Kreis zu schließen

Java

```
public void prim(Map<Node, Node> p) {
  SimpleIterator<Node> it;
  for(it = nodeiterator(); it.more(); it.step()) it.key().weight = 1e10;
  it = nodeiterator(); it.key().weight = 0.0;
  Heap<Node> H = new Heap<Node>();
  for(it = nodeiterator(); it.more(); it.step()) H.insert(it.key());
  while(!H.isempty()) {
    Node u = H.extract_min();
    Iterator<Node, Edge> et;
    for(et = neighbors.find(u).iterator(); et.more(); et.step()) {
      Node v = et.key();
      Edge e = et.data();
      if(H.iselement(v) && e.weight < v.weight) {
        p.insert(v, u);
        v.weight = e.weight; H.decrease_key(v);
      }
    }
  }
}
```

Der Algorithmus von Prim – Laufzeit

Laufzeit für einen Graphen $G = (V, E)$.

- Anzahl von extract_min: $|V|$
- Anzahl von decrease_key: $|E|$

Laufzeit ist $O((|V| + |E|) \log |V|)$, falls wir einen Heap als Prioritätswarteschlange verwenden.

Laufzeit ist $O(|V| \log |V| + |E|)$, falls wir stattdessen einen Fibonacci-Heap nehmen.

Korrektheit des Algorithmus folgt später.

Greedy Algorithmen – Münzen wechseln

Es gibt diese acht Euromünzen:

Was ist die minimale Zahl von Münzen um 3.34 Euro zu zahlen?

Antwort:
Wir brauchen sechs Münzen:

Es ist unmöglich, weniger Münzen zu verwenden.

Münzwechsel – Ein Greedy-Algorithmus

Wir wollen den Betrag n wechseln:

Algorithmus

$r := n$;
while $r > 0$ **do**
  Choose biggest coin c *with* $value(c) \leq r$;
  $S := S \cup \{c\}$;
  $r := r - value(c)$
od;
return S

Korrektheit

Lemma A

Sei C eine Münze und v ein Betrag, der mindestens so groß ist wie der Wert von C.
Dann ist es suboptimal, v mit Münzen kleiner als C auszudrücken.

Beweise das Lemma für jede Münze von der kleinsten bis zur größten.

Nimm  als Beispiel.

Sei v mindestens 1 EUR. Nehmen wir an, v kann mit genau k  und kleineren Münzen für die verbleibenden $v - 50k$ Cents optimal bezahlt werden.

Da diese $v - 50k$ Cents optimal ausgezahlt werden, muß $v - 50k < 50$ und somit auch $100 \leq v < 50(k+1)$ gelten. Es folgt $k \geq 2$, ein Widerspruch zur Optimalität.

Korrektheit

Theorem

Der Greedy-Algorithmus für den Münzwechsel ist optimal.

Beweis.

Nimm an, $C_1, C_2, \ldots, C_n$ ist eine Greedy-Lösung (mit $C_i \geq C_{i+1}$).
Zeige mit Induktion über k, daß eine optimale Lösung mit $C_1, C_2, \ldots, C_k$ beginnt.

Falls dem nicht so wäre, gäbe es eine optimale Lösung $C_1, C_2, \ldots, C_{k-1}, C'_k, \ldots, C'_m$ wobei $C_k > C'_i$ für $i = k, \ldots, m$.
Da $C'_k + \cdots + C'_m \geq C_k$ ist dies ein Widerspruch zu Lemma A. □

Briefmarkensammeln

Funktioniert der Greedy-Algorithmus auch für Briefmarken aus Manchukuo?

Welche Briefmarken für einen 20 fen–Brief?

Der Greedy-Algorithmus führt nicht zu einer optimalen Lösung und findet manchmal gar keine!

Matroide

Der Korrektheitsbeweis für Greedy-Algorithmen kann sehr trickreich sein. Münzwechsel ist hierfür ein Beispiel.
Viele Beweise können allerdings mit Hilfe von der Theorie der Matroide geführt werden.

Definition (Matroid)

Ein Matroid $M = (S, \mathcal{I})$ besteht aus einer Basis S und einer Familie $\mathcal{I} \subseteq 2^S$ von unabhängigen Mengen mit:

1. Falls $A \subseteq B$ und $B \in \mathcal{I}$, dann $A \in \mathcal{I}$ (M ist hereditary).
2. Falls $A, B \in \mathcal{I}$ und $|A| < |B|$ dann gibt es ein $x \in B$ so daß $A \cup \{x\} \in \mathcal{I}$ (M hat die Austauscheigenschaft).

Matroide

Beispiel – Der graphische Matroid

Sei $G = (V, E)$ ein ungerichteter Graph.

Sei $\mathcal{F} = \{ F \subseteq E \mid (V, F) \text{ ist azyklisch} \}$.

Dann ist (E, F) ein Matroid.

Zum Beweis machen wir zunächst eine Beobachtung:
Es sei $G = (V, E)$ ein Wald. Dann verbindet die Kante $e \in E$ zwei Bäume in G gdw. $G = (V, E \cup \{e\})$ kreisfrei ist.

Beweis.

- Vererbungseigenschaft: Wegnehmen von Kanten kann keine Kreise schließen.
- Austauscheigenschaft: Seien $G_A = (V, A)$ und $G_B = (V, B)$ Teilwälder von G und $|A| < |B|$
  1. Beobachtung: Ein Wald mit k Kanten besteht aus $|V| - k$ Bäumen.
  2. G_A hat mehr Bäume als G_B
  3. Es gibt einen Baum T in G_B, der zwei Bäume in G_A verbindet
  4. Es gibt eine Kante in T, die keinen Kreis in G_A schließt

Matroide

Wir nennen eine unabhängige Menge A maximal, wenn keine echte Obermenge von A unabhängig ist.

Lemma (Lemma G1)

Alle maximalen unabhängigen Mengen eines Matroids haben die gleiche Größe.

Beweis.

Angenommen, daß A und B maximale unabhängige Mengen sind, aber $|A| < |B|$. Nach der Austauscheigenschaft gibt es ein $x \in B$,

so daß $A \cup \{x\}$ unabhängig ist. Das widerspricht der Maximalität von A. □

Gewichtete Matroide

- Ein gewichtetes Matroid ist ein Matroid $M = (S, \mathcal{I})$ mit einer Gewichtsfunktion $w\colon S \to \mathbf{Q}$.
- Wir nennen eine Menge maximalen Gewichts unter allen maximalen unabhängigen Mengen optimal.
- Viele Optimierungsprobleme können durch das Finden einer optimalen Menge in einem Matroid gelöst werden.

Beispiel

Ein minimaler Spannbaum ist eine optimale Menge im graphischen Matroid.

Der Greedy-Algorithmus auf gewichteten Matroiden

Sei $M = (S, \mathcal{I})$ ein Matroid mit Gewichten w.

Dieser Algorithmus findet eine optimale Menge.

Algorithmus

function *Greedy*(S) :
$R := \emptyset$;
sort S *into* $(s[1], ..., s[n])$ *with* $w(s[i]) \geq w(s[i+1])$;
for $i = 1, ..., n$ **do**
  if $R \cup \{s[i]\} \subseteq I$ **then** $R := A \cup \{s[i]\}$ **fi**
od;
return R

Die Laufzeit ist $O(n \log n + nf(n))$, wenn ein Vergleich konstante Zeit braucht, und der Unabhängigkeitstest $O(f(n))$.

Korrektheit

Lemma (G2)

Sei $M = (S, \mathcal{I})$ ein Matroid mit Gewichten w. Wenn $x \in S$ maximales Gewicht unter allen x hat, für die $\{x\}$ unabhängig ist, dann gibt eine optimale Menge, die x enthält.

Beweis.

Sei B eine optimale Menge mit $x \notin B$. Wir haben $w(y) \leq w(x)$ für jedes $y \in B$, weil $\{y\}$ nach der Vererbungseigenschaft unabhängig ist. Beginne mit $A = \{x\}$ und füge Elemente von B zu A hinzu (A bleibt unabhängig, wegen der Austauscheigenschaft) bis $|A| = |B|$. Dann ist $A = B - \{y\} \cup \{x\}$ für ein $y \in B$ und daher gilt $w(A) \geq W(B)$.

□

Kontraktion eines Matroids

Definition

Sei $M = (S, \mathcal{I})$ ein Matroid und $x \in S$.
Dann ist $M' = (S', \mathcal{I}')$ die Kontraktion von M um x, wobei

- $S' = \{ y \in S \mid \{x, y\} \in \mathcal{I}, x \neq y \}$,
- $\mathcal{I}' = \{ A \subseteq S - \{x\} \mid A \cup \{x\} \in \mathcal{I} \}$.

Wir beobachten, daß der Greedy-Algorithmus auf M' arbeitet, nachdem er x gewählt hat.

Lemma (G3)

Sei $M = (S, \mathcal{I})$ ein Matroid mit Gewichten w. Sei x das erste Element, daß durch den Greedy-Algorithmus gewählt wird. Dann ist eine optimale Menge für die Kontraktion M' von M um x zusammen mit x eine optimale Menge für M.

Beweis.

Sei $x \in A$ und $B = A - \{x\}$. Dann ist A maximal unabhängig in M gdw. B maximal unabhängig in M' ist.
Da $w(A) = w(B) + w(x)$ gilt, ist A optimal in M gdw. B optimal in M' ist. □

Korrektheit des Greedy-Algorithmus

Theorem

Der Greedy-Algorithmus berechnet eine optimale Menge in einem gewichteten Matroid.

Beweis.

Aus $G2$ und $G3$ folgt die Korrektheit mittels Induktion über die Größe der maximalen unabhängigen Menge, die nach $G1$ wohldefiniert ist. □