WS 2005/06

Diskrete Strukturen

Ernst W. Mayr

Fakultät für Informatik
TU München

http://www14.in.tum.de/lehre/2005WS/ds/

31. Januar 2006

4.3 Matroide

Definition 300

Sei S eine endliche Menge, $U \subseteq 2^S$ eine Teilmenge der Potenzmenge von S. Dann heißt $M = (S, U)$ ein Matroid und jedes $A \in U$ heißt unabhängige Menge, falls gilt:

1. $\emptyset \in U$
2. $A \in U,\ B \subseteq A \quad \Longrightarrow \quad B \in U$
3. $$A, B \in U,\ |B| = |A| + 1 \quad \Longrightarrow \quad (\exists x \in B \setminus A)\Big[\big(A \cup \{x\}\big) \in U\Big]$$

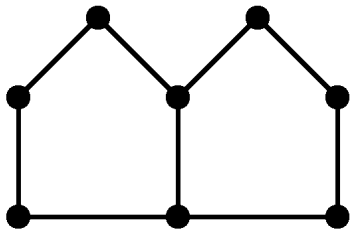
Jede bezüglich $\subseteq$ maximale Menge in U heißt Basis.
Nach 3. haben je zwei Basen gleiche Kardinalität. Diese heißt der Rang $\mathrm{r}(M)$ des Matroids.

Beispiel 301

Linear unabhängige Vektoren in einem Vektorraum.

Beispiel 302

G sei folgender Graph:



$S =$ Menge der Kanten von G
$U =$ Menge der kreisfreien Teilmengen von S

4.4 Greedy-Algorithmus

Sei $M = (S, U)$ ein Matroid, $w : S \to R$ eine Gewichtsfunktion.

```
algorithm greedy(S,U,w)
  B := ∅
  while (|B|<r(M)) do
    sei x ∈ {y ∈ S \ B; B ∪ {y} ∈ U} mit
      minimalem Gewicht
    B := B ∪ {x}
  od
end
```

Satz 303

Der Greedy-Algorithmus liefert eine Basis minimalen Gewichts.

Beweis:

Aus der Definition des Matroids (1.) folgt, dass die leere Menge $\emptyset$ eine unabhängige Menge ist.
Aus 3. folgt, dass in der `while`-Schleife wiederum nur unabhängige Mengen generiert werden.
Daher ist B am Ende des Algorithmus eine Basis (da inklusionsmaximal). Es bleibt zu zeigen, dass die gefundene Basis minimales Gewicht besitzt.
Sei also $B = \{b_1, \ldots, b_r\}$ die vom Algorithmus gelieferte Basis. Sei $b_1, \ldots, b_r$ die Reihenfolge der Elemente, in der sie der Greedy-Algorithmus ausgewählt hat. Dann gilt

$$w(b_1) \leq w(b_2) \leq \ldots \leq w(b_r).$$

Sei weiter $B' = \{b'_1, \ldots, b'_r\}$ eine minimale Basis, und es gelte o. B. d. A.

$$w(b'_1) \leq w(b'_2) \leq \ldots \leq w(b'_r) \ .$$

Sei $i \in \{1, \ldots, r\}$. Gemäß Eigenschaft 3 für Matroide folgt, dass es ein $b' \in \{b'_1, \ldots, b'_i\}$ gibt, so dass $\{b_1, \ldots, b_{i-1}, b'\} \in U$.

Damit ist $w(b_i) \leq w(b'_i)$ (für alle i), und daher wegen der Minimalität von B'

$$w(b_i) = w(b'_i) \qquad \text{für alle } i \ .$$

4.5 Minimale Spannbäume

Satz 304

Sei $G = (V, E)$ ein zusammenhängender, ungerichteter Graph, $F \subseteq 2^E$ die Menge der kreisfreien Teilmengen von E. Dann ist $M = (E, F)$ ein Matroid mit Rang $|V| - 1$.

Beweis:

Es sind die drei Eigenschaften eines Matroids zu zeigen.

1. $\emptyset$ ist kreisfrei und daher in F enthalten.
2. Ist A kreisfrei und B eine Teilmenge von A, dann ist auch B kreisfrei.

3. Sind A und B kreisfrei, $|B| = |A| + 1$, dann existiert ein $b \in B$, so dass $A \cup \{b\}$ kreisfrei ist:

   Wir betrachten die Wälder (V, A) (mit $|A|$ Kanten und $|V| - |A|$ Zusammenhangskomponenten) und (V, B) (mit $|B|$ Kanten und $|V| - |B|$ Zusammenhangskomponenten). Diese Bedingungen lassen zwei Möglichkeiten zu:

   1. Es existiert eine Kante e in B, die zwei Zusammenhangskomponenten in (V, A) verbindet. Damit ist $A \cup \{e\}$ kreisfrei.
   2. Alle Kanten in B verlaufen innerhalb der Zusammenhangskomponenten in (V, A). (V, A) besitzt jedoch eine Zusammenhangskomponente mehr als (V, B). Daher muss es eine Zusammenhangskomponente in (V, A) geben, deren Knoten nicht in (V, B) auftauchen, was einen Widerspruch darstellt.

Kruskals Algorithmus:

```
algorithm kruskal
  sortiere E aufsteigend: w(e_1) ≤ ... ≤ w(e_m).
  F := ∅
  i := 0
  while |F|<|V|-1 do
    i++
    if F ∪ {e_i} kreisfrei then
      F := F ∪ {e_i}
    fi
  od
end
```

Satz 305

Kruskals Algorithmus bestimmt (bei geeigneter Implementierung) einen minimalen Spannbaum für $G = (V, E)$ in Zeit $O\Big(|E| \cdot \log\big(|V|\big)\Big)$.

Beweis:

Die Korrektheit folgt aus Satz 304.

Zur Laufzeit:

Die Sortierung von E nach aufsteigendem Gewicht benötigt

$$O\Big(|E| \cdot \log\big(|E|\big)\Big),$$

z. B. mit Heapsort oder Mergesort.

Da $|E| \leq (|V|)^2$, gilt auch

$$O\Big(|E| \cdot \log\big(|V|\big)\Big)$$

als Zeitbedarf für das Sortieren.

Implementierung des Tests auf Kreisfreiheit:

Repräsentation der Zusammenhangskomponenten:
Feld Z: $Z[i]$ ist die Zusammenhangskomponente des Knoten i.
Feld N: $N[j]$ ist die Anzahl der Knoten in der Zusammenhangskomponente j.
Feld M: $M[j]$ ist eine Liste mit den Knoten in der Zusammenhangskomponente j.

```
co Initialisierung oc
for all i ∈ V do
  Z[i] := i
  N[i] := 1
  M[i] := (i)
od
co Test auf Kreisfreiheit oc
sei e := {i, j}
```

Fortsetzung

```
co F ∪ {e} kreisfrei ⇔ Z[i]≠Z[j] oc
if Z[i]≠Z[j] then
  if N[Z[i]] <= N[Z[j]] then
    BigSet := Z[j]
    SmallSet := Z[i]
  else
    BigSet := Z[i]
    SmallSet := Z[j]
  fi
  N[BigSet] := N[BigSet] + N[SmallSet]
  for all k ∈ M[SmallSet] do
    Z[k] := BigSet
  od
  hänge M[SmallSet] an M[BigSet] an
fi
```

Zeitbedarf für den Test: $O(1)$ für jede Abfrage, damit dafür insgesamt

$$O\big(|E|\big).$$

Zeitbedarf für das Umbenennen der Zusammenhangskomponenten: Nach jedem Umbenennen befindet sich ein Knoten in einer mindestens doppelt so großen Zusammenhangskomponente. Daher ist die Anzahl der Umbenennungen je Knoten $\leq \log\big(|V|\big)$. Für das Umbenennen aller Knoten benötigt man dann

$$O\Big(|V| \cdot \log\big(|V|\big)\Big).$$

□

Bemerkung:
Es gibt Algorithmen für minimale Spannbäume der Komplexität $O(m + n \cdot \log n)$
und, für dünnbesetzte Graphen, der Komplexität $O(m \cdot \log^* n)$, wobei

$$\log^* x = \min_{n \in \mathbb{N}} \left\{ n : \underbrace{\log\Big(\log\big(\cdots \log(x) \cdots\big)\Big)}_{n} < 1 \right\}.$$