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 := R $\cup$ { 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

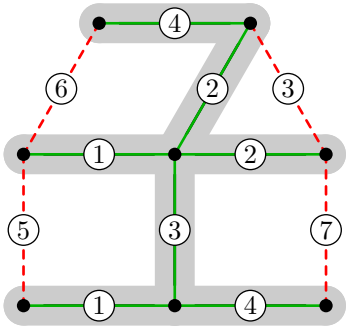
Theorem

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. □

Kruskal: Minimaler Spannbaum



Kruskals Algorithmus – Implementierung

Algorithmus

function Kruskal(G, w) :
A := ∅;
for each vertex v in V[G] **do**
  Make_Set(v)
od;
sort the edges of E into nondecreasing order by weight;
for each edge { u, v}in E, nondecreasingly **do**
  if Find_Set(u) ≠ Find_Set(v) **then**
    A := A ∪ {{ u, v}};
    Union(u, v)
  fi
od;
return A

Korrektheit und Laufzeit der Implementierung

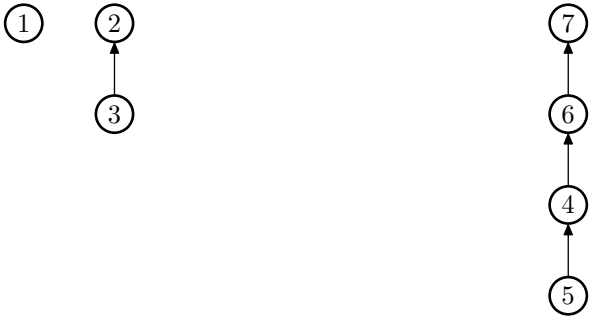
Die Korrektheit folgt als Korollar aus der Korrektheit des allgemeinen Greedy-Algorithmus und der Beobachtung zum graphischen Matroid.
Laufzeit mit $m = |E|$ und $n = |V|$ und $m \geq n - 1$:
n Make-Set-Operationen,
$O(m \log m)$ Zeit für das Sortieren der Kanten, und
$O(m)$ Find-Set- und Union-Operationen.

Mit einer geeigneten Union-Find-Implementierung zusammen $O(m \log m)$.

Union-Find: naiv

union(a, b): Hänge *find*(a) bei *find*(b) ein

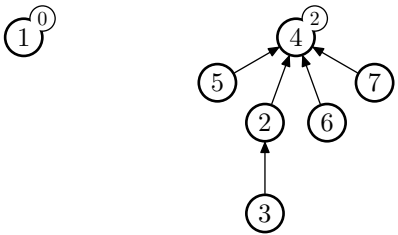
Beispiel: *union*$(5, 4)$, *union*$(3, 2)$, *union*$(5, 6)$, *union*$(4, 7)\ldots$



Union-Find: union by rank

union(a, b): Verwende die ranghöhere Wurzel

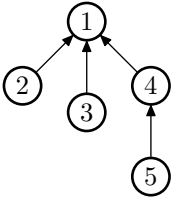
Beispiel: *union*$(5, 4)$, *union*$(3, 2)$, *union*$(5, 6)$, *union*$(4, 7)$, *union*$(3, 4)$



Union-Find: Pfadkompression

find(*a*): Komprimiere durchlaufene Pfade

Beispiel: *find*(4)



Union-Find

Algorithmus

procedure Make_Set(x) :
p[x] := x;
rank[x] := 0

- Wir betrachten jeweils eine Menge $\{0, \ldots, n-1\}$
- Ein Array für die Eltern eines für den Rang
- Ein Baum pro Menge repräsentiert durch die Wurzel
- Wurzel hier kurzgeschlossen

Union-Find

Algorithmus

function Find_Set(x) :
if $x \neq p[x]$ **then** $p[x] := \text{Find\_Set}(p[x])$ **fi**;
return p[x];

Algorithmus

procedure Union(x, y) :
$x := \text{Find\_Set}(x)$;
$y := \text{Find\_Set}(y)$;
if $\text{rank}[x] > \text{rank}[y]$ **then** $p[y] := x$
else $p[x] := y$;
if $\text{rank}[x] = \text{rank}[y]$ **then** rank[y]++**fi**;
fi;

Java

```
public class Partition {
  int[] s;
  public Partition(int n) {
    s = new int[n];
    for(int i = 0; i < n; i++) s[i] = i;
  }
  public int find(int i) {
    int p = i, t;
    while(s[p] ≠ p) p = s[p];
    while(i ≠ p) { t = s[i]; s[i] = p; i = t; }
    return p;
  }
  public void union(int i, int j) { s[find(i)] = find(j); }
}
```

Union-Find – Analyse

Zunächst keine Pfadkompression.

Lemma

Falls eine Union-Find-Datenstruktur einen Baum mit m Elementen enthält, dann ist seine Höhe höchstens $\log m + 1$.

Beweis.

Wird ein Element neu hinzugefügt, dann ist die Höhe des Baums $1 = \log(1) + 1$.
Werden zwei Bäume zu einem kombiniert, dann ist die Höhe anschließend unverändert, außer die Höhen beider Bäume waren exakt gleich h.
Falls die Bäume vorher k und m Elemente enthielten, galt $h \leq \log(k) + 1 \leq \log(m) + 1$.
Daher $h + 1 \leq \log(2m) + 1 \leq \log(k + m) + 1$. □