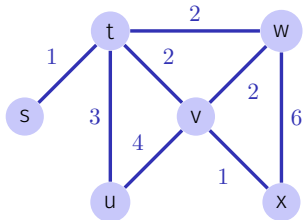
14. Minimale Spannbäume

Motivation, Greedy, Algorithmus von Kruskal, Allgemeine Regeln,
Union-Find Struktur, Algorithmus von Jarnik, Prim, Dijkstra
[Ottman/Widmayer, Kap. 9.6, 6.2, 6.1, Cormen et al, Kap. 23, 19]

Problem

Gegeben: Ungerichteter, zusammenhängender, gewichteter Graph $G = (V, E, c)$.

Gesucht: Minimaler Spannbaum $T = (V, E')$: zusammenhängender, zyklenfreier Teilgraph $E' \subset E$, so dass $\sum_{e \in E'} c(e)$ minimal.



Beispiele von Anwendungen

- Netzwerk-Design: finde das billigste / kürzeste Netz oder Leitungssystem, welches alle Knoten miteinander verbindet.
- Approximation einer Lösung des Travelling-Salesman Problems: finde einen möglichst kurzen Rundweg, welcher jeden Knoten einmal besucht.[18]

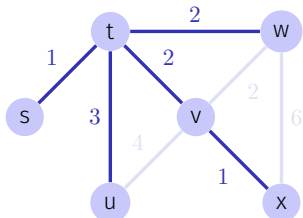
[18] Der beste bekannte Algorithmus zur exakten Lösung des TS Problems hat exponentielle Laufzeit

Greedy Verfahren

- Gierige Verfahren berechnen eine Lösung schrittweise, indem lokal beste Lösungen gewählt werden.
- Die meisten Probleme sind nicht mit einer greedy Strategie lösbar.
- Das Problem des Minimalen Spannbaumes kann mit einem gierigen Verfahren effizient gelöst werden.

Greedy Idee (Kruskal, 1956)

Konstruiere T indem immer die billigste Kante hinzugefügt wird, welche keinen Zyklus erzeugt.



(Lösung ist nicht eindeutig.)

Algorithmus MST-Kruskal(G)

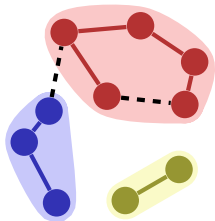
Input: Gewichteter Graph $G = (V, E, c)$
Output: Minimaler Spannbaum mit Kanten A.

Sortiere Kanten nach Gewicht $c(e_1) \leq ... \leq c(e_m)$
$A \leftarrow \emptyset$
for $k = 1$ **to** $|E|$ **do**
    if $(V, A \cup \{e_k\})$ kreisfrei **then**
        $A \leftarrow A \cup \{e_k\}$

return (V, A, c)

Zur Implementation

Gegeben eine Menge von Mengen $i \equiv A_i \subset V$. Zur Identifikation von Schnitten und Kreisen: Zugehörigkeit der beiden Endpunkte einer Kante zu einer der Mengen.



Zur Implementation

Allgemeines Problem: Partition (Menge von Teilmengen) z.B.
$\{\{1, 2, 3, 9\}, \{7, 6, 4\}, \{5, 8\}, \{10\}\}$
Benötigt: Abstrakter Datentyp „Union-Find“ mit folgenden Operationen

- Make-Set(i): Hinzufügen einer neuen Menge i.
- Find(e): Name i der Menge, welche e enthält.
- Union(i, j): Vereingung der Mengen mit Namen i und j.

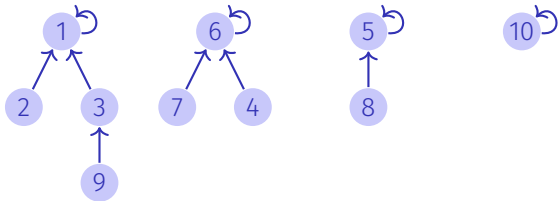
Union-Find Algorithmus MST-Kruskal(G)

Input: Gewichteter Graph $G = (V, E, c)$
Output: Minimaler Spannbaum mit Kanten A.

Sortiere Kanten nach Gewicht $c(e_1) \leq ... \leq c(e_m)$
$A \leftarrow \emptyset$
for $k = 1$ **to** $|V|$ **do**
  MakeSet(k)
for $k = 1$ **to** m **do**
  $(u, v) \leftarrow e_k$
  if Find(u) $\neq$ Find(v) **then**
    Union(Find(u), Find(v))
    $A \leftarrow A \cup e_k$
  else // konzeptuell: $R \leftarrow R \cup e_k$
return (V, A, c)

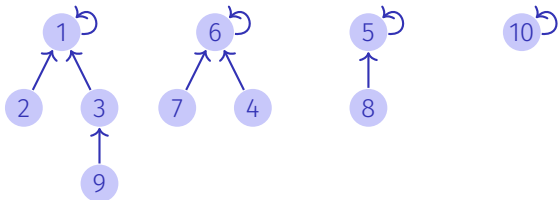
Implementation Union-Find

Idee: Baum für jede Teilmenge in der Partition, z.B.
$\{\{1,2,3,9\},\{7,6,4\},\{5,8\},\{10\}\}$



Baumwurzeln = Namen (Stellvertreter) der Mengen,
Bäume = Elemente der Mengen

Implementation Union-Find



Repräsentation als Array:

Index	1	2	3	4	5	6	7	8	9	10
Parent	1	1	1	6	5	6	5	5	3	10

Implementation Union-Find

Index	1	2	3	4	5	6	7	8	9	10
Parent	1	1	1	6	5	6	5	5	3	10

Make-Set(i)	$p[i] \leftarrow i$; **return** i
Find(i)	**while** $(p[i] \neq i)$ **do** $i \leftarrow p[i]$ **return** i
Union(i, j) [19]	$p[j] \leftarrow i$;

[19] i und j müssen Namen (Wurzeln) der Mengen sein. Andernfalls verwende Union(Find(i),Find(j))

Optimierung der Laufzeit für Find

Baum kann entarten: Beispiel Union$(8,7)$, Union$(7,6)$, Union$(6,5)$, ...

Index	1	2	3	4	5	6	7	8	..
Parent	1	1	2	3	4	5	6	7	..

Laufzeit von Find im schlechtesten Fall in $\Theta(n)$.

Optimierung der Laufzeit für Find

Idee: Immer kleineren Baum unter grösseren Baum hängen. Benötigt zusätzliche Grösseninformation (Array) g

Make-Set(i)	$p[i] \leftarrow i$; $g[i] \leftarrow 1$; **return** i
Union(i, j)	**if** $g[j] > g[i]$ **then** swap(i, j) $p[j] \leftarrow i$ **if** $g[i] = g[j]$ **then** $g[i] \leftarrow g[i] + 1$

$\Rightarrow$ Baumtiefe (und schlechteste Laufzeit für Find) in $\Theta(\log n)$

Weitere Verbesserung

Bei jedem Find alle Knoten direkt an den Wurzelknoten hängen.

Find(i):

$j \leftarrow i$
while $(p[i] \neq i)$ **do** $i \leftarrow p[i]$
while $(j \neq i)$ **do**
    $t \leftarrow j$
    $j \leftarrow p[j]$
    $p[t] \leftarrow i$
return i

Laufzeit: amortisiert *fast* konstant (Inverse der Ackermannfunktion).[20]

[20] Wird hier nicht vertieft.

Laufzeit des Kruskal Algorithmus

- Sortieren der Kanten: $\Theta(|E| \log |E|) = \Theta(|E| \log |V|)$. [21]
- Initialisieren der Union-Find Datenstruktur $\Theta(|V|)$
- $|E|\times$ Union(Find(x),Find(y)): $\mathcal{O}(|E| \log |E|) = \mathcal{O}(|E| \log |V|)$.

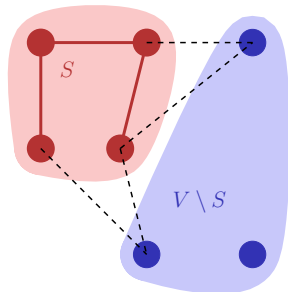
Insgesamt $\Theta(|E| \log |V|)$.

[21] da G zusammenhängend: $|V| \leq |E| \leq |V|^2$

Algorithmus von Jarnik (1930), Prim, Dijkstra (1959)

Idee: Starte mit einem $v \in V$ und lasse von dort unter Verwendung der Auswahlregel einen Spannbaum wachsen:

$A \leftarrow \emptyset$
$S \leftarrow \{v_0\}$
for $i \leftarrow 1$ **to** $|V|$ **do**
  Wähle billigste (u, v) mit $u \in S$, $v \notin S$
  $A \leftarrow A \cup \{(u, v)\}$
  $S \leftarrow S \cup \{v\}$ // (Färbung)



Anmerkung: man benötigt keine Union-Find Datenstruktur. Es genügt, Knoten zu färben, sobald sie zu S hinzugenommen werden.

Laufzeit

Trivial $\mathcal{O}(|V| \cdot |E|)$.
Verbesserung (wie bei Dijkstras Kürzeste Pfade):

- Mit Min-Heap, Kosten:
  - Initialisierung (Knotenfärbung) $\mathcal{O}(|V|)$
  - $|V|\times$ ExtractMin = $\mathcal{O}(|V| \log |V|)$,
  - $|E|\times$ Insert oder DecreaseKey: $\mathcal{O}(|E| \log |V|)$,

$\mathcal{O}(|E| \cdot \log |V|)$