

28. Minimale Spann­b­äume

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

Günstigstes Stromnetz

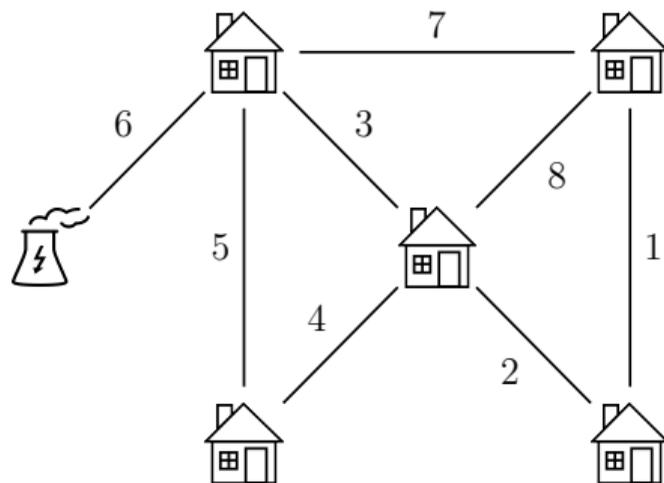
Günstigstes Stromnetz

Gegeben: Häuser



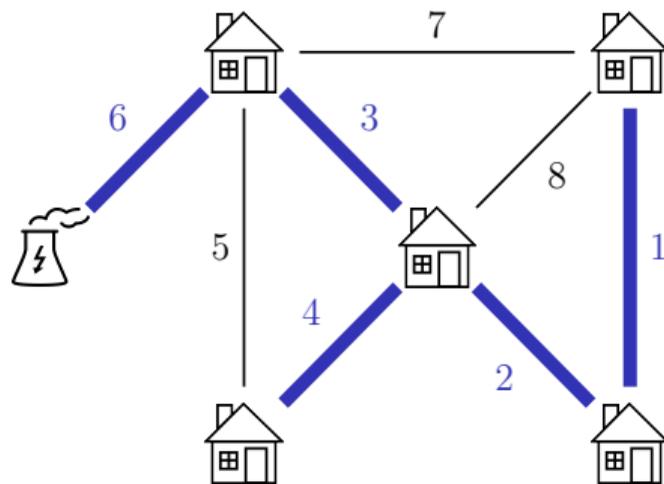
Günstigstes Stromnetz

Gegeben: Häuser und Kosten für Verbindung von Häusern mit Strom



Günstigstes Stromnetz

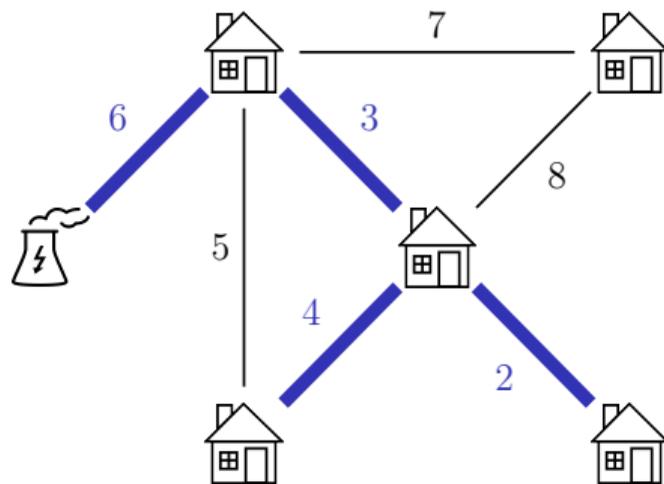
Gegeben: Häuser und Kosten für Verbindung von Häusern mit Strom



Gesucht: Günstigstes Stromnetz, das alle Häuser erreicht.

Günstigstes Stromnetz

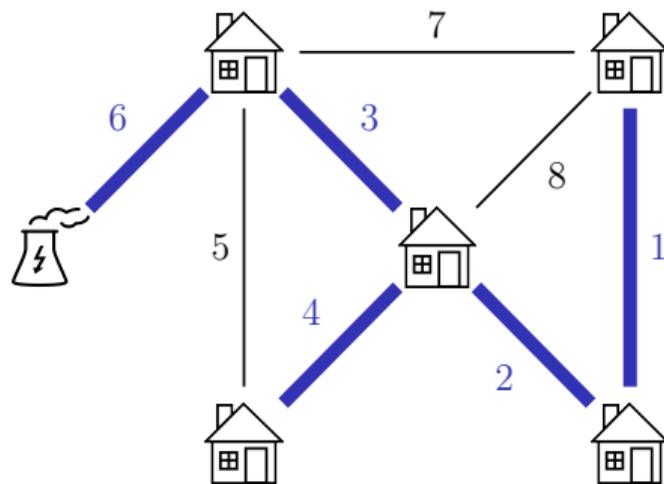
Gegeben: Häuser und Kosten für Verbindung von Häusern mit Strom



Gesucht: Günstigstes Stromnetz, das alle Häuser erreicht.

Günstigstes Stromnetz

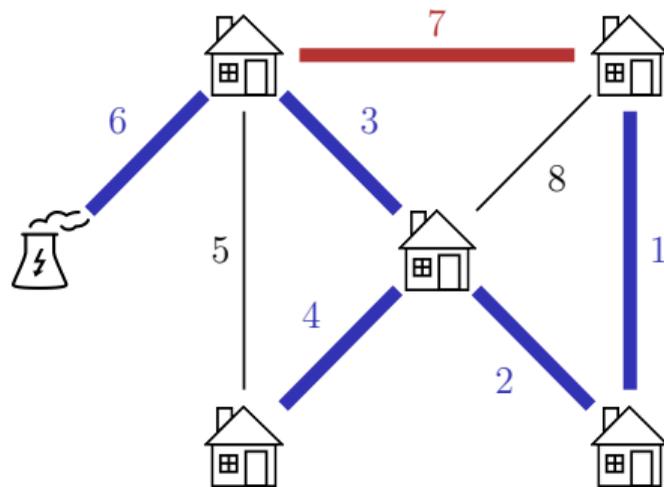
Gegeben: Häuser und Kosten für Verbindung von Häusern mit Strom



Gesucht: Günstigstes Stromnetz, das alle Häuser erreicht.

Günstigstes Stromnetz

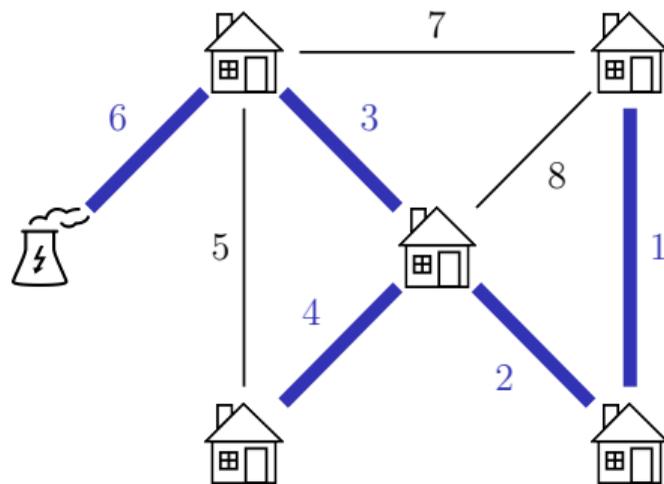
Gegeben: Häuser und Kosten für Verbindung von Häusern mit Strom



Gesucht: Günstigstes Stromnetz, das alle Häuser erreicht.

Günstigstes Stromnetz

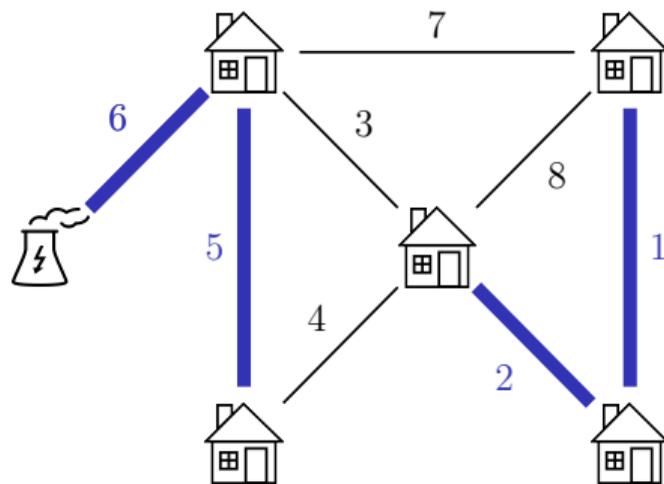
Gegeben: Häuser und Kosten für Verbindung von Häusern mit Strom



Gesucht: Günstigstes Stromnetz, das alle Häuser erreicht.

Günstigstes Stromnetz

Gegeben: Häuser und Kosten für Verbindung von Häusern mit Strom



Gesucht: Günstigstes Stromnetz, das alle Häuser erreicht.

Anforderungen ans Stromnetz

Anforderungen ans Stromnetz

- Jedes Haus muss mindestens eine Leitung im Stromnetz haben.

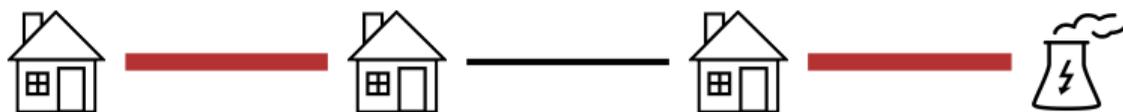


Anforderungen ans Stromnetz

- Jedes Haus muss mindestens eine Leitung im Stromnetz haben.



- Das Stromnetz muss zusammenhängend sein (nur 1 Stromnetz).

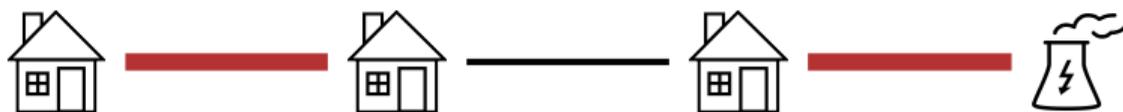


Anforderungen ans Stromnetz

- Jedes Haus muss mindestens eine Leitung im Stromnetz haben.



- Das Stromnetz muss zusammenhängend sein (nur 1 Stromnetz).

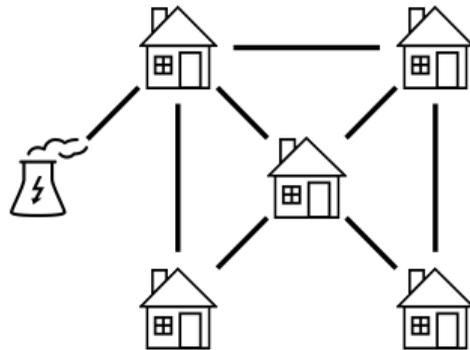


- Das Stromnetz soll keine Zyklen haben.



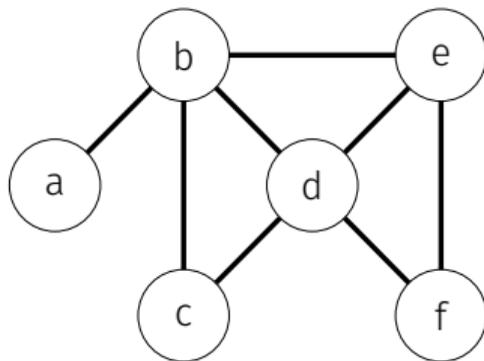
Spannbaum

Spannbaum



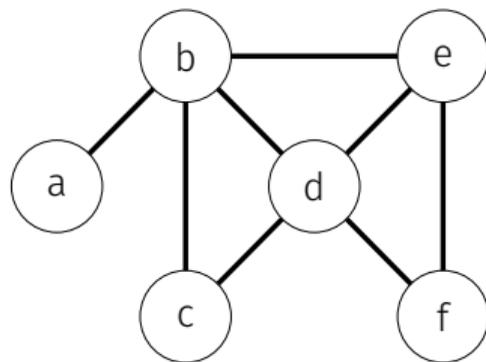
Spannbaum

Gegeben: ungerichteter, zusammenhängender Graph $G = (V, E)$



Spannbaum

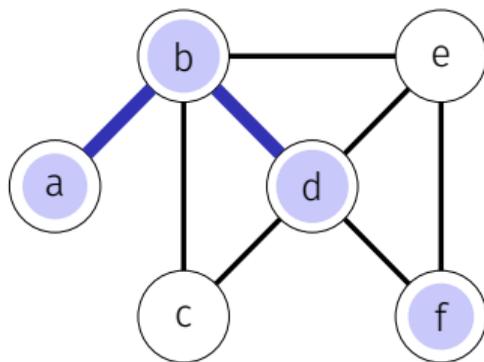
Gegeben: ungerichteter, zusammenhängender Graph $G = (V, E)$



Spannbaum von G :

Spannbaum

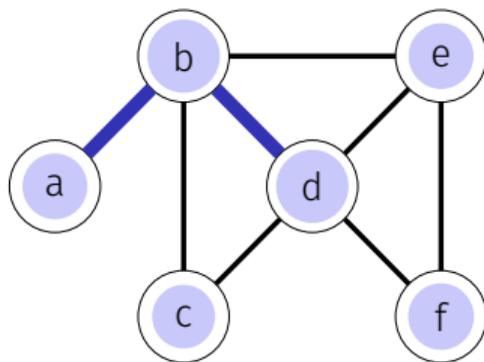
Gegeben: ungerichteter, zusammenhängender Graph $G = (V, E)$



Spannbaum von G : Teilgraph $T = (V', E')$ mit $V' \subseteq V, E' \subseteq E$ so dass

Spannbaum

Gegeben: ungerichteter, zusammenhängender Graph $G = (V, E)$

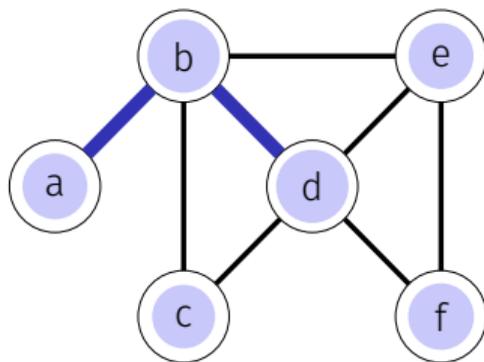


Spannbaum von G : Teilgraph $T = (V', E')$ mit $V' \subseteq V, E' \subseteq E$ so dass

- Spann-: $V' = V$ (spannt alle Knoten auf)

Spannbaum

Gegeben: ungerichteter, zusammenhängender Graph $G = (V, E)$

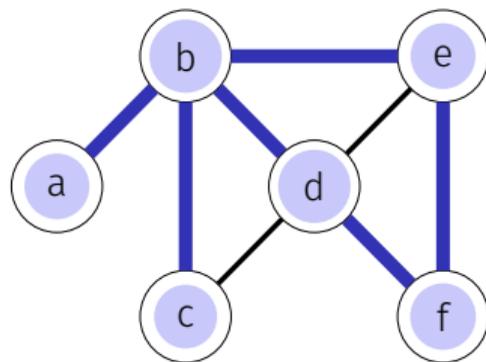


Spannbaum von G : Teilgraph $T = (V', E')$ mit $V' \subseteq V, E' \subseteq E$ so dass

- Spann-: $V' = V$ (spannt alle Knoten auf)
- Baum:

Spannbaum

Gegeben: ungerichteter, zusammenhängender Graph $G = (V, E)$

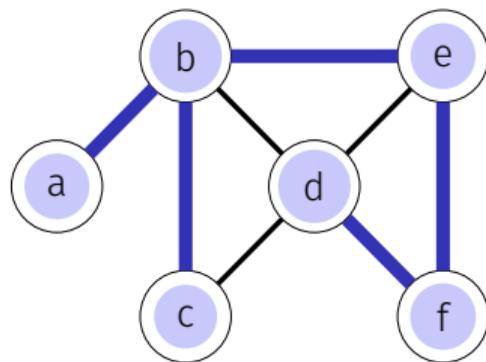


Spannbaum von G : Teilgraph $T = (V', E')$ mit $V' \subseteq V, E' \subseteq E$ so dass

- Spann-: $V' = V$ (spannt alle Knoten auf)
- Baum: zusammenhängend

Spannbaum

Gegeben: ungerichteter, zusammenhängender Graph $G = (V, E)$

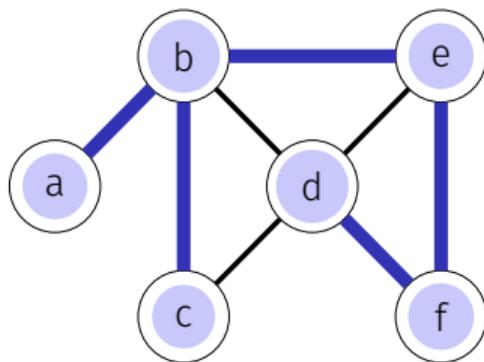


Spannbaum von G : Teilgraph $T = (V', E')$ mit $V' \subseteq V, E' \subseteq E$ so dass

- Spann-: $V' = V$ (spannt alle Knoten auf)
- Baum: zusammenhängend und zyklentfrei

Spannbaum

Gegeben: ungerichteter, zusammenhängender Graph $G = (V, E)$



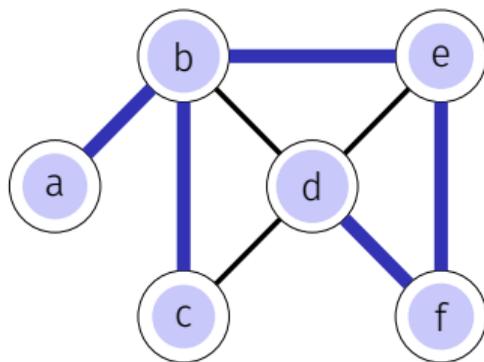
Spannbaum von G : Teilgraph $T = (V', E')$ mit $V' \subseteq V, E' \subseteq E$ so dass

- Spann-: $V' = V$ (spannt alle Knoten auf)
- Baum: zusammenhängend und zyklenfrei

⇒ für jedes Paar von Knoten: genau einen verbindenden Pfad

Spannbaum

Gegeben: ungerichteter, zusammenhängender Graph $G = (V, E)$



Spannbaum von G : Teilgraph $T = (V', E')$ mit $V' \subseteq V, E' \subseteq E$ so dass

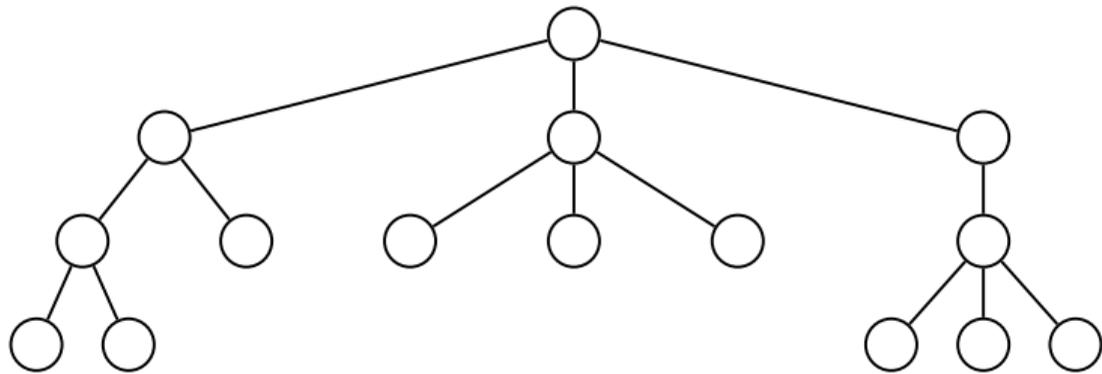
- Spann-: $V' = V$ (spannt alle Knoten auf)
- Baum: zusammenhängend und zyklenfrei

⇒ für jedes Paar von Knoten: genau einen verbindenden Pfad

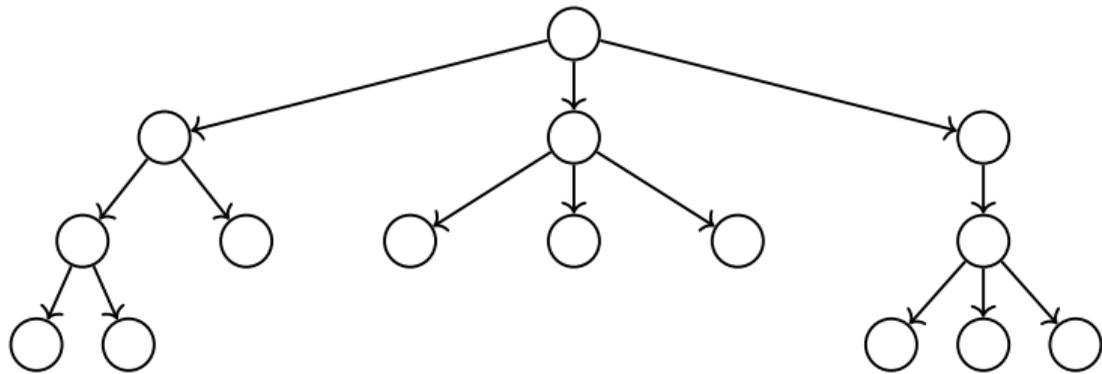
⇒ Spannbaum hat genau $|V| - 1$ Kanten ($|E'| = |V| - 1$)

Bäume

Bäume

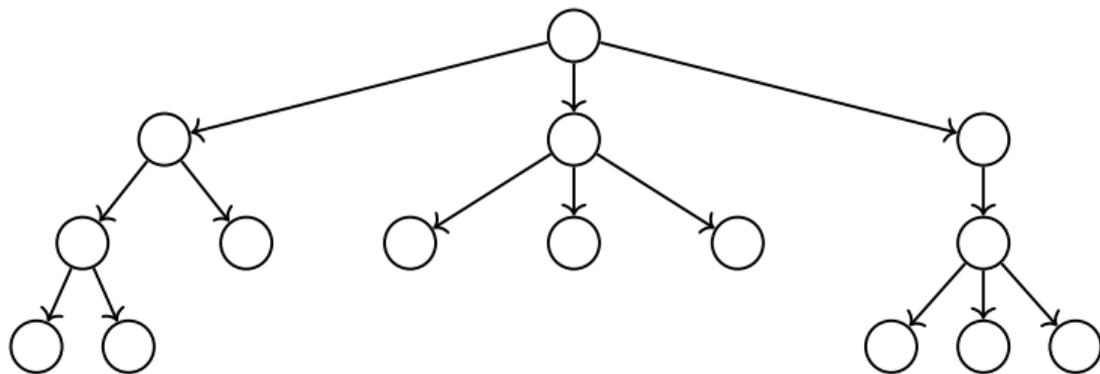


Bäume



Bisher waren Bäume gerichtete Bäume!

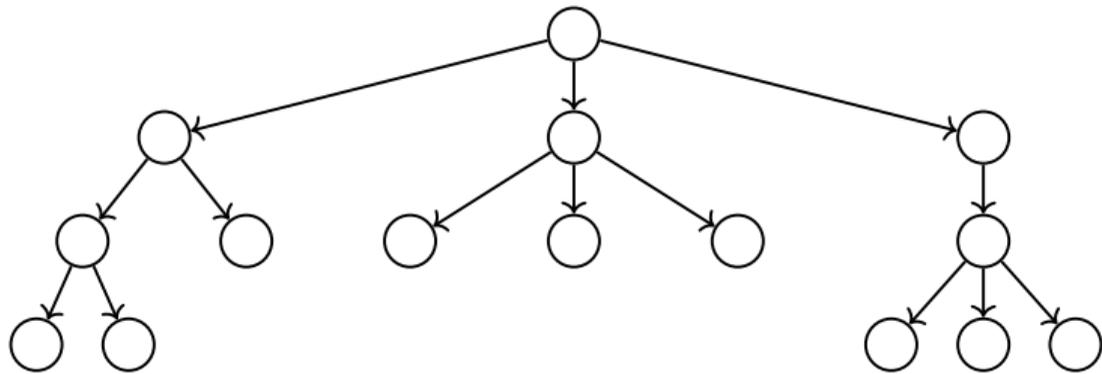
Bäume



Bisher waren Bäume gerichtete Bäume!

- zusammenhängend

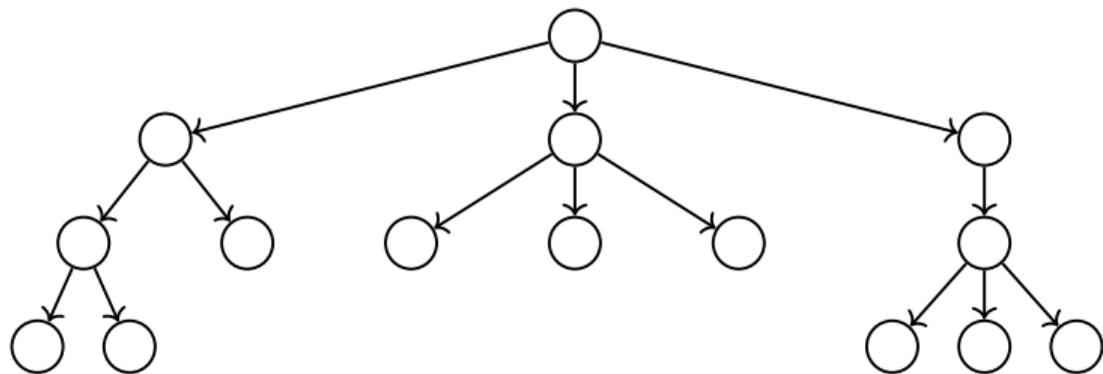
Bäume



Bisher waren Bäume gerichtete Bäume!

- zusammenhängend
- zyklenfrei

Bäume



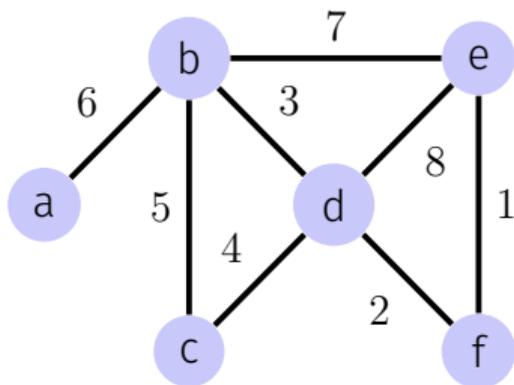
Bisher waren Bäume gerichtete Bäume!

- zusammenhängend
- zyklenfrei
- gerichtet von Eltern zu Kindern

Minimaler Spannbaum (MST)

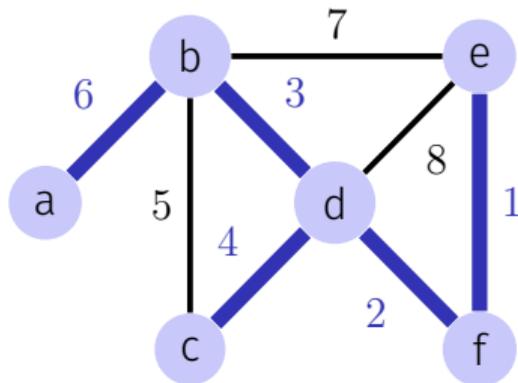
Minimaler Spannbaum (MST)

Gegeben: ungerichteter, gewichteter, zusammenhängender Graph $G = (V, E, c)$ mit Kantengewichten $c: E \rightarrow \mathbb{R}$



Minimaler Spannbaum (MST)

Gegeben: ungerichteter, gewichteter, zusammenhängender Graph $G = (V, E, c)$ mit Kantengewichten $c: E \rightarrow \mathbb{R}$



Gesucht: Spannbaum $T = (V, E')$ von G mit minimalem Gewicht $\sum_{e \in E'} c(e)$

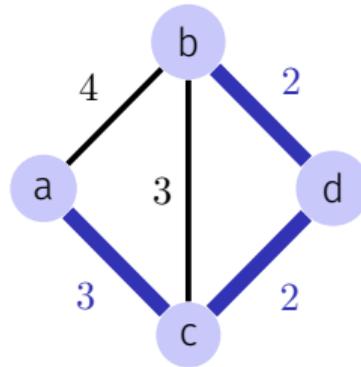
Beobachtungen

Beobachtungen

- Ist das das gleiche wie kürzeste Wege?

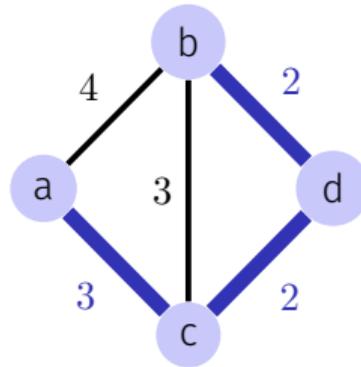
Beobachtungen

- Ist das das gleiche wie kürzeste Wege? Nein!



Beobachtungen

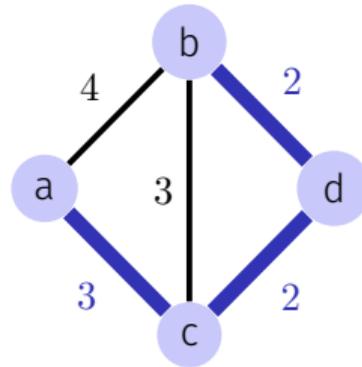
- Ist das das gleiche wie kürzeste Wege? Nein!



- Ist der minimale Spannbaum eindeutig?

Beobachtungen

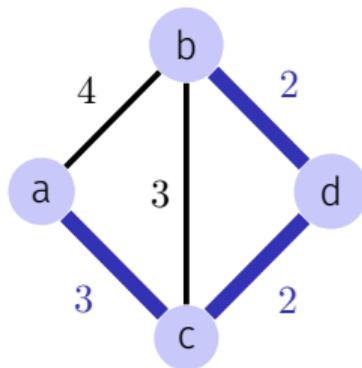
- Ist das das gleiche wie kürzeste Wege? Nein!



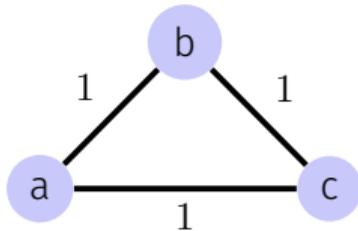
- Ist der minimale Spannbaum eindeutig? Nicht immer.

Beobachtungen

- Ist das das gleiche wie kürzeste Wege? Nein!



- Ist der minimale Spannbaum eindeutig? Nicht immer.



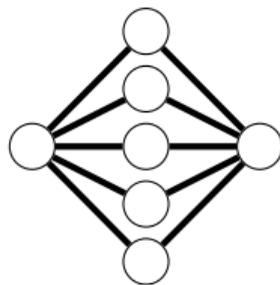
Trivialer Brute-Force Algorithmus?

Trivialer Brute-Force Algorithmus?

Alle Spannbäume ausprobieren?

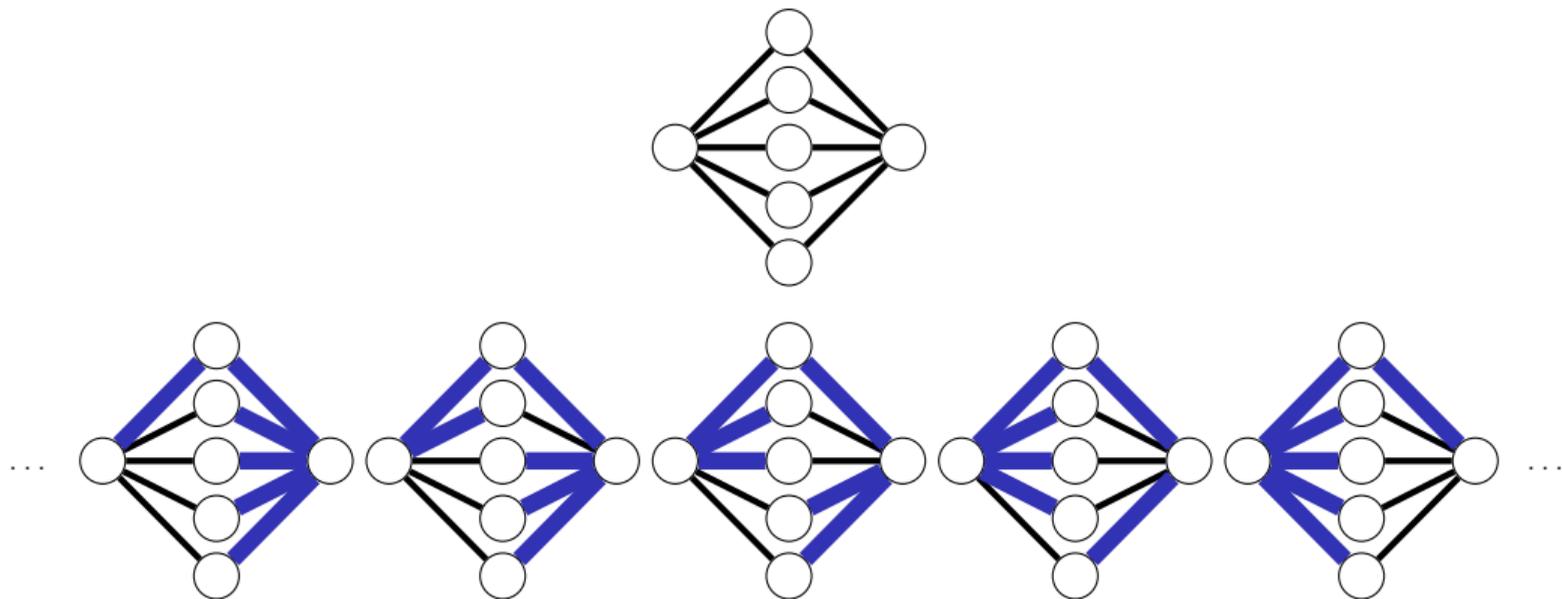
Trivialer Brute-Force Algorithmus?

Alle Spannbäume ausprobieren?



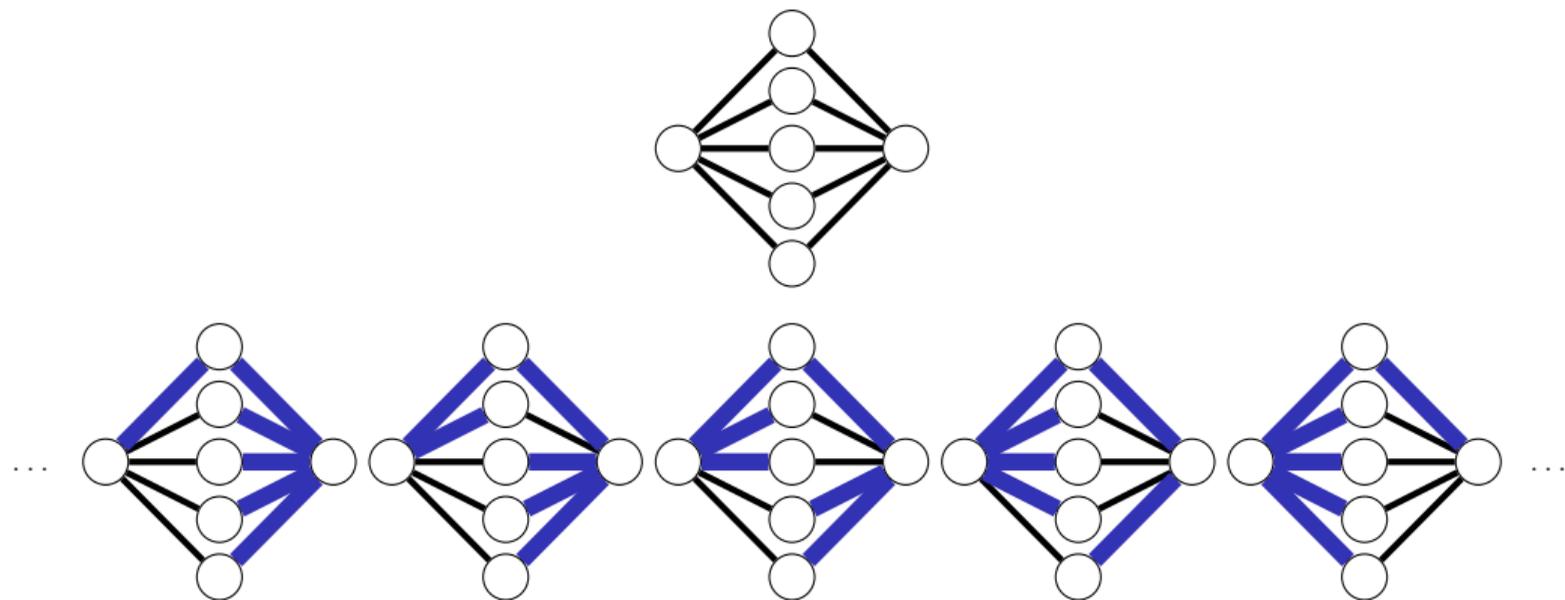
Trivialer Brute-Force Algorithmus?

Alle Spannbäume ausprobieren?



Trivialer Brute-Force Algorithmus?

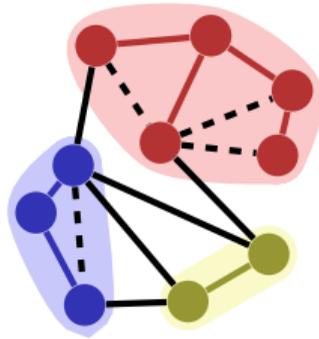
Alle Spannbäume ausprobieren?



⇒ Ineffizient: Es gibt Graphen mit exponentiell vielen Spannbäumen.

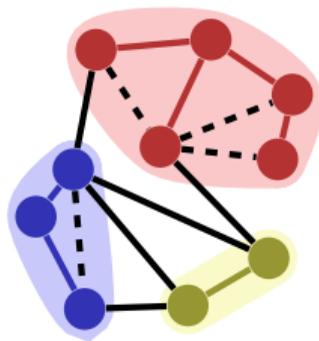
28.2 Kruskal's Algorithmus

Kruskals Algorithmus



Kruskals Algorithmus

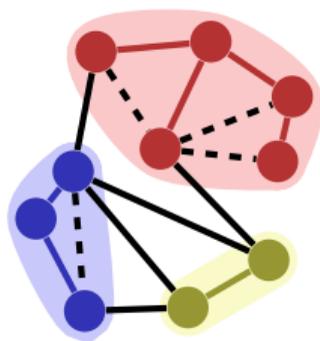
Idee (Greedy): leichteste Kante hinzufügen falls sie keinen Zyklus erzeugt



Kruskals Algorithmus

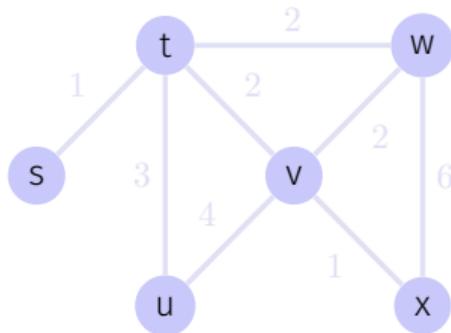
Idee (Greedy): leichteste Kante hinzufügen falls sie keinen Zyklus erzeugt

Invariante: nach i Schritten, i Kanten vom MST und die entsprechenden Komponenten bekannt



Beispiel

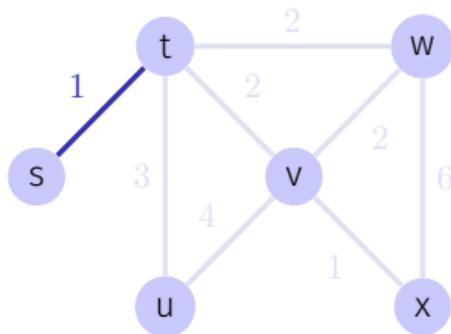
Konstruiere T indem immer die billigste Kante hinzugefügt wird, welche keinen Zyklus erzeugt.



(Lösung ist nicht eindeutig.)

Beispiel

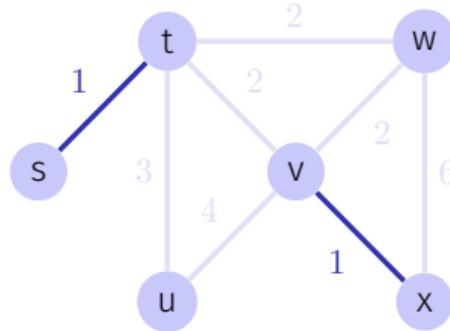
Konstruiere T indem immer die billigste Kante hinzugefügt wird, welche keinen Zyklus erzeugt.



(Lösung ist nicht eindeutig.)

Beispiel

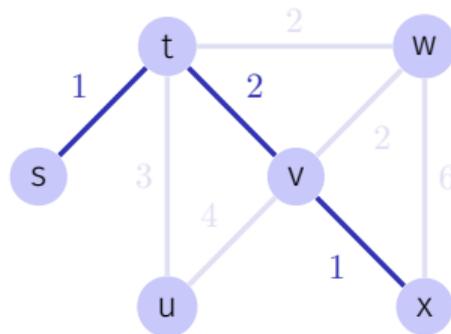
Konstruiere T indem immer die billigste Kante hinzugefügt wird, welche keinen Zyklus erzeugt.



(Lösung ist nicht eindeutig.)

Beispiel

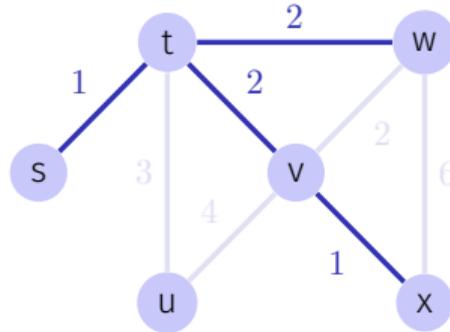
Konstruiere T indem immer die billigste Kante hinzugefügt wird, welche keinen Zyklus erzeugt.



(Lösung ist nicht eindeutig.)

Beispiel

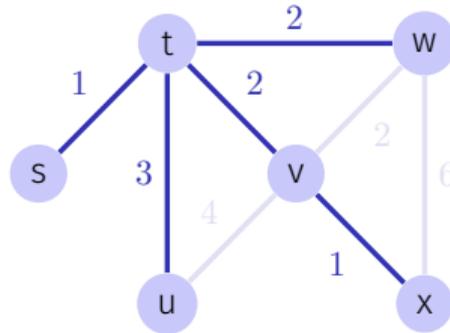
Konstruiere T indem immer die billigste Kante hinzugefügt wird, welche keinen Zyklus erzeugt.



(Lösung ist nicht eindeutig.)

Beispiel

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 $c(e_1) \leq \dots \leq c(e_m)$

$A \leftarrow \emptyset$

for $k = 1$ **to** $|E|$ **do**

if $(V, A \cup \{e_k\})$ kreisfrei **then**
 $A \leftarrow A \cup \{e_k\}$

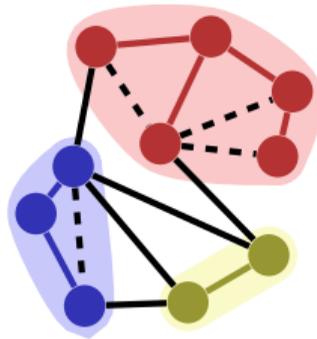
return (V, A, c)

(Korrektheitsbeweis im Handout.)

Zur Implementation

Zur Implementation

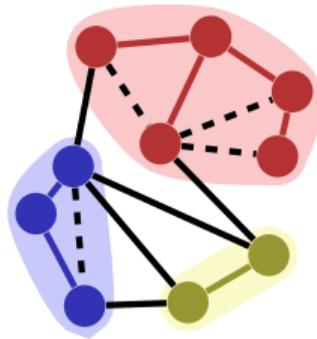
Gegeben eine Menge von Mengen $i \equiv V_i \subset V$.



Zur Implementation

Gegeben eine Menge von Mengen $i \equiv V_i \subset V$.

Zur Identifikation von 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

- $\text{Make-Set}(i)$: Hinzufügen einer neuen Menge i .
- $\text{Find}(e)$: Name i der Menge, welche e enthält.
- $\text{Union}(i, j)$: Vereinigung 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 $c(e_1) \leq \dots \leq c(e_m)$

$A \leftarrow \emptyset$

for $k = 1$ **to** $|V|$ **do**

\lfloor 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))

$A \leftarrow A \cup e_k$

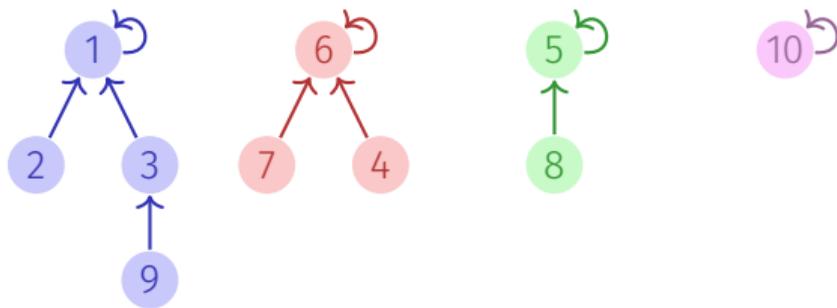
else

// konzeptuell: $R \leftarrow R \cup e_k$

return (V, A, c)

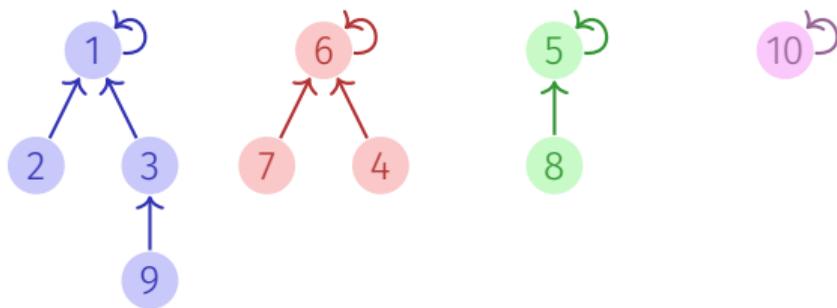
Implementation Union-Find

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	6	5	3	10

Implementation Union-Find

Index	1	2	3	4	5	6	7	8	9	10
Parent	1	1	1	6	5	6	6	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)⁴⁶ $p[j] \leftarrow i$;

⁴⁶ 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) $p[i] \leftarrow i; g[i] \leftarrow 1; \mathbf{return} \ i$

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 $\Theta(\log n)$
(Beweis: Handout)

Alternative 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).⁴⁷

⁴⁷Wenn kombiniert mit der Vereinigung nach Grösse, wird hier nicht vertieft. Siehe z.B. Cormen et al, Kap. 21.4

Laufzeit des Kruskal Algorithmus

- Sortieren der Kanten: $\Theta(|E| \log |E|) = \Theta(|E| \log |V|)$.⁴⁸
 - Initialisieren der Union-Find Datenstruktur $\Theta(|V|)$
 - $|E| \times \text{Union}(\text{Find}(x), \text{Find}(y))$: $\mathcal{O}(|E| \log |E|) = \mathcal{O}(|E| \log |V|)$.
- Insgesamt $\Theta(|E| \log |V|)$.

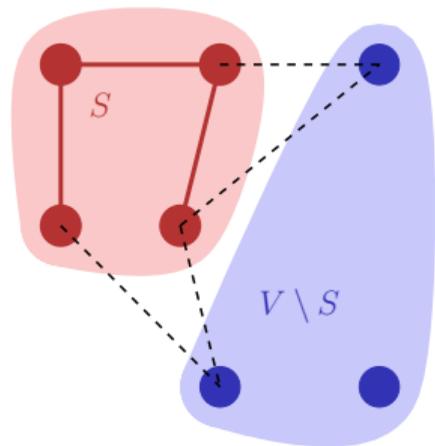
⁴⁸da G zusammenhängend: $|V| \leq |E| \leq |V|^2$

28.4 Algorithmus von Jarnik, Prim, Dijkstra

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

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

```
A ← ∅  
S ← {v0}  
for i ← 1 to |V| do  
  Wähle billigste (u, v) mit u ∈ S, v ∉ S  
  A ← A ∪ {(u, v)}  
  S ← S ∪ {v} // (Färbung)
```



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

Implementation und Laufzeit

Implementation wie bei Dijkstra's Kürzeste Wege Algorithmus. Einziger Unterschied:

Kürzeste Wege

Relaxiere (u, v) :

```
if  $d_s[v] > d[u] + c(u, v)$  then  
     $d_s[v] \leftarrow d_s[u] + c(u, v)$   
     $\pi_s[v] \leftarrow u$ 
```



Minimaler Spannbaum

Relaxiere (u, v) :

```
if  $d_s[v] > c(u, v)$  then  
     $d_s[v] \leftarrow c(u, v)$   
     $\pi_s[v] \leftarrow u$ 
```

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

Beispiele von Anwendungen

Beispiele von Anwendungen

- Netzwerk-Design: finde das billigste/kürzeste Netz oder Leitungssystem, welches alle Knoten miteinander verbindet.

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.

28.6 Fibonacci Heaps

Fibonacci Heaps

Datenstruktur zur Verwaltung von Elementen mit Schlüsseln. Operationen

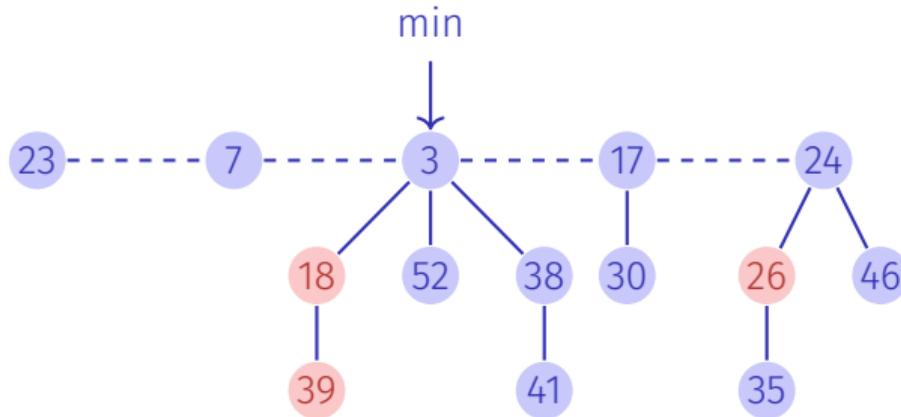
- $\text{MakeHeap}()$: Liefere neuen Heap ohne Elemente
- $\text{Insert}(H, x)$: Füge x zu H hinzu
- $\text{Minimum}(H)$: Liefere Zeiger auf das Element m mit minimalem Schlüssel
- $\text{ExtractMin}(H)$: Liefere und entferne (von H) Zeiger auf das Element m
- $\text{Union}(H_1, H_2)$: Liefere Verschmelzung zweier Heaps H_1 und H_2
- $\text{DecreaseKey}(H, x, k)$: Verkleinere Schlüssel von x in H zu k
- $\text{Delete}(H, x)$: Entferne Element x von H

Vorteil gegenüber Binary Heap?

	Binary Heap (worst-Case)	Fibonacci Heap (amortisiert)
MakeHeap	$\Theta(1)$	$\Theta(1)$
Insert	$\Theta(\log n)$	$\Theta(1)$
Minimum	$\Theta(1)$	$\Theta(1)$
ExtractMin	$\Theta(\log n)$	$\Theta(\log n)$
Union	$\Theta(n)$	$\Theta(1)$
DecreaseKey	$\Theta(\log n)$	$\Theta(1)$
Delete	$\Theta(\log n)$	$\Theta(\log n)$

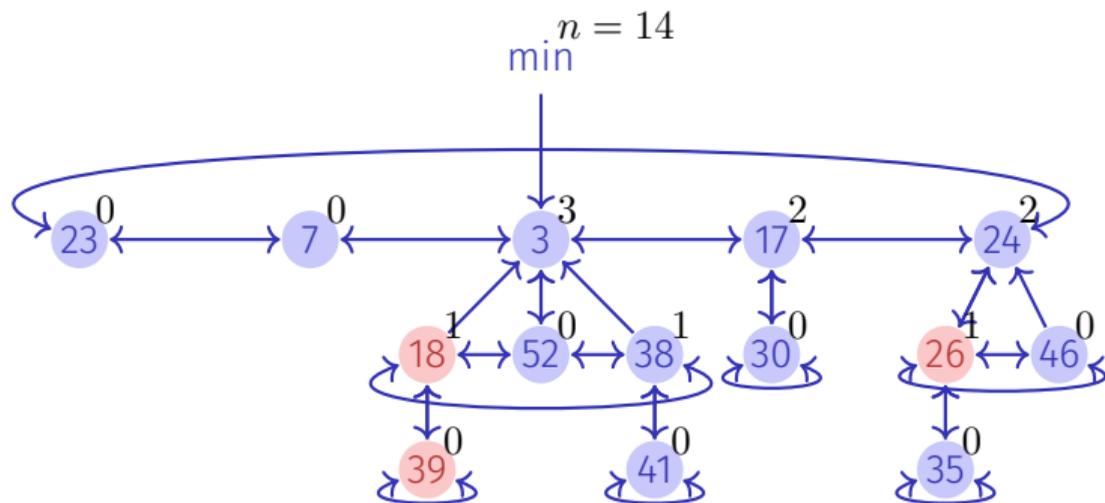
Struktur

Menge von Bäumen, welche der Min-Heap Eigenschaft genügen.
Markierbare Knoten.



Implementation

Doppelt verkettete Listen von Knoten mit marked-Flag und Anzahl Kinder.
Zeiger auf das minimale Element und Anzahl Knoten.



Einfache Operationen

- MakeHeap (trivial)
- Minimum (trivial)
- Insert(H, e)
 1. Füge neues Element in die Wurzelliste ein
 2. Wenn Schlüssel kleiner als Minimum, min-pointer neu setzen.
- Union (H_1, H_2)
 1. Wurzellisten von H_1 und H_2 aneinander hängen
 2. Min-Pointer neu setzen.
- Delete(H, e)
 1. DecreaseKey($H, e, -\infty$)
 2. ExtractMin(H)

ExtractMin

1. Entferne Minimalknoten m aus der Wurzelliste
2. Hänge Liste der Kinder von m in die Wurzelliste
3. Verschmelze solange heapgeordnete Bäume gleichen Ranges, bis alle Bäume unterschiedlichen Rang haben:
Rangarray $a[0, \dots, n]$ von Elementen, zu Beginn leer. Für jedes Element e der Wurzelliste:
 - a Sei g der Grad von e .
 - b Wenn $a[g] = nil$: $a[g] \leftarrow e$.
 - c Wenn $e' := a[g] \neq nil$: Verschmelze e mit e' zu neuem e'' und setze $a[g] \leftarrow nil$. Setze e'' unmarkiert Iteriere erneut mit $e \leftarrow e''$ vom Grad $g + 1$.

DecreaseKey (H, e, k)

1. Entferne e von seinem Vaterknoten p (falls vorhanden) und erniedrige den Rang von p um eins.
2. Insert(H, e)
3. Vermeide zu dünne Bäume:
 - a Wenn $p = nil$, dann fertig
 - b Wenn p unmarkiert: markiere p , fertig.
 - c Wenn p markiert: unmarkiere p , trenne p von seinem Vater pp ab und Insert(H, p). Iteriere mit $p \leftarrow pp$.

Skizze der amortisierten Analyse im Handout.