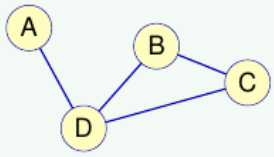
Graphen

1. Darstellung

1.1 Adjazenzmatrix

Adjazenzmatrix: Beispiel für einen ungerichteten Graphen

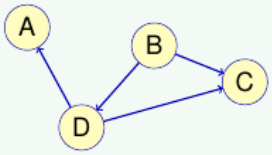
	A	B	C	D
A	⊥			
B	⊥	⊥		
C	⊥	⊤	⊥	
D	⊤	⊤	⊤	⊥



1.2 Adjazenzliste

Adjazenzlisten: Beispiel für einen gerichteten Graphen

A	
B	C, D
C	
D	A, C



1.3 Array-Einbettung

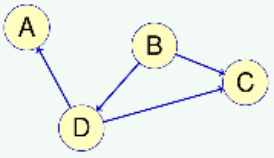
Array der Länge $|V|+1+|E|$

- Der Wert im Array ist der Index ab dem im Array die Nachfolger kommen

  → Das Intervall zwischen einem Wert und dem Wert des nächsten Array-Eintrags (z. B. *[a[0]..a[1]]*) verweist auf die Einträge im Array in denen die Indices der Nachbarn des Knotens stehen
- Nach den Knoten kommt ein „Dummy-Feld“ mit der Länge des Arrays
- Danach stehen die Indices der Knoten auf die verwiesen werden (Nachbarn)

Array-Einbettung: Beispiel

A	B	C	D					
5	5	7	7	9	2	3	0	2
0	1	2	3	4	5	6	7	8



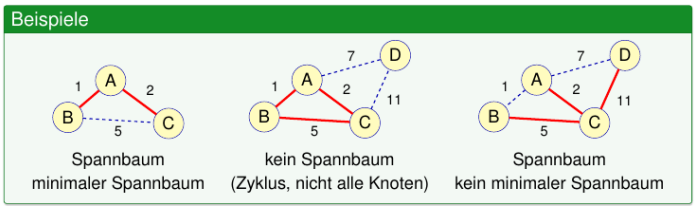
Erläuterung:

- An Stelle 0 steht der Index an dem der erste Nachfolger von Knoten $v_0 = A$ stehen würde. Da jedoch an Stelle 1 derselbe Index steht, hat *A* keine Nachfolger.
- An Stelle 1 steht der Index, an dem der erste Nachfolger von Knoten $v_1 = B$ steht ($\rightsquigarrow$ 5). Da an Stelle 2 eine 7 steht, stehen im Bereich $[5, 6]$ alle Nachfolger von *B*.
- Da an Stelle 2 und 3 derselbe Wert steht, hat *C* keine Nachfolger.
- Die Nachfolger von *D* stehen im Intervall $[7, 8]$.
- Der Wert an Stelle 4 verweist auf das erste Element „hinter“ dem Array.

2. SPANNBAUM

Enthält alle Knoten und ist zyklenfrei.

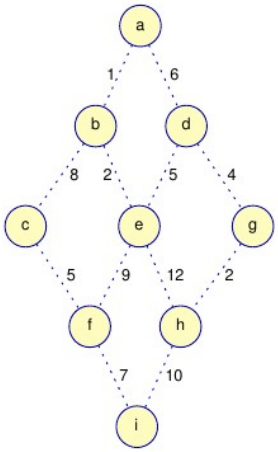
Minimal: Geringste Gesamtkosten aller Kanten



2.1 KRUSKAL-ALGORITHMUS (GIERIG):

1. Leerer Graph der nur die Knoten enthält
2. Kanten werden aufsteigen nach Gewicht sortiert (Union/Find)
3. Falls Graph noch nicht zusammenhängend, füge oberste Kante ein, die zwei unzusammenhängende Knoten verbindet (kein Zyklus)

Laufzeikomplexität: $O(\log(|E|)\cdot|E|)$

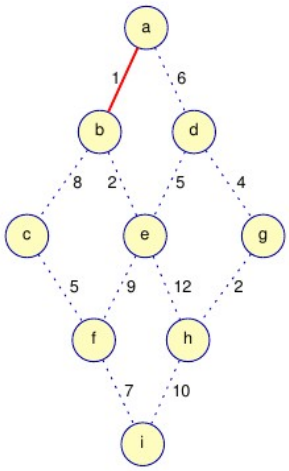


Zu Beginn enthält der Spannbaum keine Kanten.

Verfügbare Kanten (sortiert nach Gewicht):

- [a,b] mit Gewicht 1
- [b,e] mit Gewicht 2
- [h,g] mit Gewicht 2
- [d,g] mit Gewicht 4
- ...

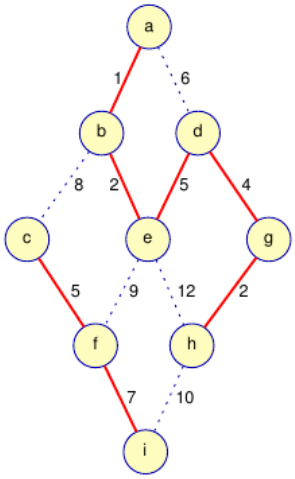
Kante [a,b] wird wegen minimalem Gewicht zu *S* hinzugefügt.



Verfügbare Kanten (sortiert nach Gewicht):

- [b,e] mit Gewicht 2
- [h,g] mit Gewicht 2
- [d,g] mit Gewicht 4
- ...

Es könnte sowohl Kante [b,e] als auch Kante [h,g] hinzugefügt werden. Es wird zufällig eine gewählt.



Verfügbare Kanten (sortiert nach Gewicht):

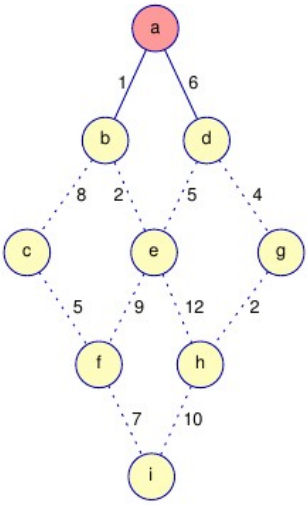
- [b,c] mit Gewicht 8
- [e,f] mit Gewicht 9
- [h,i] mit Gewicht 10
- ...

Es wird die Kante [b,c] hinzugefügt.

2.2 PRIM-ALGORITHMUS (GIERIG):

1. Leerer Graph der nur die Knoten enthält
2. Ausgehende Kanten werden nach Gewicht sortiert (Min-Heap)
3. Oberste Kante einfügen

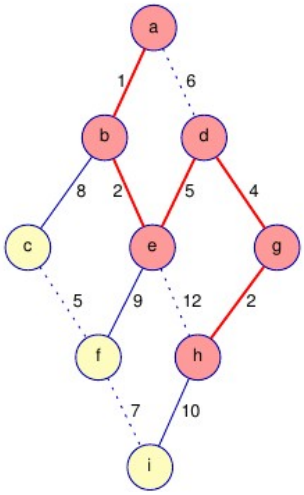
Laufzeitkomplexität: $O(|E|+|V|\cdot\log(|V|))$



Füge Knoten *a* zu *T* hinzu.

Entferne *a* aus der Datenstruktur und aktualisiere Kosten der Nachbarn von *a*:

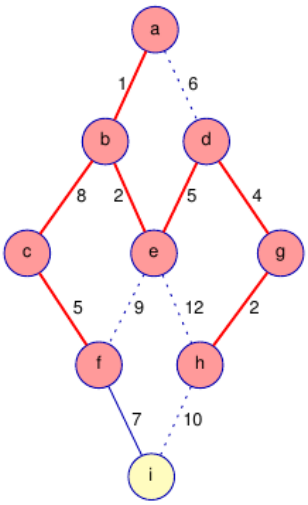
Knoten	Kosten
b	1
d	6
c	∞
e	∞
f	∞
g	∞
h	∞
i	∞



Füge Knoten *h* und Kante $[g, h]$ zu *T* hinzu.

Entferne *h* aus der Datenstruktur und aktualisiere Kosten der Nachbarn von *h*:

Knoten	Kosten
c	8
f	9
i	10

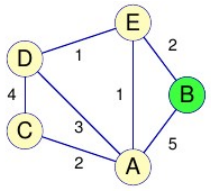


Füge Knoten *f* und Kante $[c, f]$ zu *T* hinzu.

Entferne *f* aus der Datenstruktur und aktualisiere Kosten der Nachbarn von *f*:

Knoten	Kosten
i	7

3. KÜRZESTE WEGE



3.1 DIJKSTRA-ALGORITHMUS (GIERIG):

Alle Kantengewichte müssen positiv sein.

1. Markiere alle Knoten als unbesucht, Startknoten hat Distanz 0 der Rest ∞
2. Solange noch Knoten unbesucht sind:
   1. Gehe zu unbesuchtem Knoten mit geringster Distanz
   2. Berechne Distanz (*Kantengewicht + Eigene-Distanz-zu-Startknoten*): Falls neue Distanz kleiner ist als Alte → Aktualisieren

Algorithmus von Dijkstra mit A als Startknoten:

A	B	C	D	E	Prio-Queue
0	∞	∞	∞	∞	A
0	5	2	3	1	E, C, D, B
0	3	2	2	1	C, D, B
0	3	2	2	1	D, B
0	3	2	2	1	B
0	3	2	2	1	

Unbesuchte Nachbarn von *B*:
- Keine unbesuchten Knoten mehr

Laufzeitkomplexität: $O(\log(|V|)\cdot|V|+|E|)$

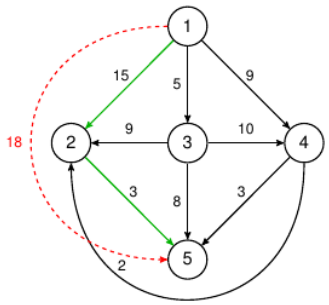
3.2 FLOYD-WARSHALL-ALGORITHMUS:

Betrachtet Wege von A nach C über B (A → B → C)!

1. Betrachte für jeden Knoten einzeln: Die Pfade von seinen Vorgängern (A) über sich selbst (B) zu seinen Nachfolgern (C)
2. Falls Länge dieses Pfades (A → B → C) kürzer ist als vorherige Länge (A → C), zeichne neue Kante mit entsprechendem Gewicht ein

Laufzeitkomplexität: $O(|V^3|)$

Vorgänger u_j	Knoten v_i	Nachfolger w_k	$\|(u_j,v_i)\|+\|(v_i,w_k)\|$ $\gamma_{j,i,k}$	„alte Länge" γ_{alt}
1	2	5	18	∞
	2			
	2			
	3			
	3			
	3			
	4			
	4			
	4			
	4			



- erstelle neue Kante von 1 nach 5, da bislang keine vorhanden war

Vorgänger u_j	Knoten v_i	Nachfolger w_k	$\|(u_j,v_i)\|+\|(v_i,w_k)\|$ $\gamma_{j,i,k}$	„alte Länge" γ_{alt}
1	2	5	18	∞
3	2	5	12	8
4	2	5	5	3
1	3	2	14	15
1	3	4	15	9
1	3	5	13	18
1	4	2	11	14
1	4	5	12	13
3	4	2	12	9
3	4	5	13	8