Algorithmus von Dijkstra

Programmieren und Software-Engineering
Theorie

23. Juni 2022

Algorithmus von Dijkstra

Der Algorithmus von Dijkstra berechnet die kürzesten Wege in einem gewichteten Graphen mit $w_{ij} \geq 0,$ für alle $[i, j] \in E$.

Grundidee:

- Ähnlichkeit zu DFS, jedoch andere Regeln für die Auswahl des nächsten Knoten.
- Ein Aufruf von Dijkstra berechnet die kürzesten Wege von einem Startknoten zu allen anderen Knoten des Graphen.
- In jedem Schritt werden **Zwischenergebnisse** $\delta_k, k \in V$ berechnet, bzw. aktualisiert.
- Diese Zwischenergebnisse sind die Länge des kürzesten *bisher gefundenen* Weges bis zu diesem Knoten.
- Wiederhole (bis alle Knoten abgeschlossen):
  1. Wähle Knoten k mit minimalem δ_k und schließe diesen ab.
  2. Speichere Verweis auf direkten Vorgänger.
  3. Aktualisiere die Werte δ_k für noch nicht abgeschlossene Nachbarknoten von k.

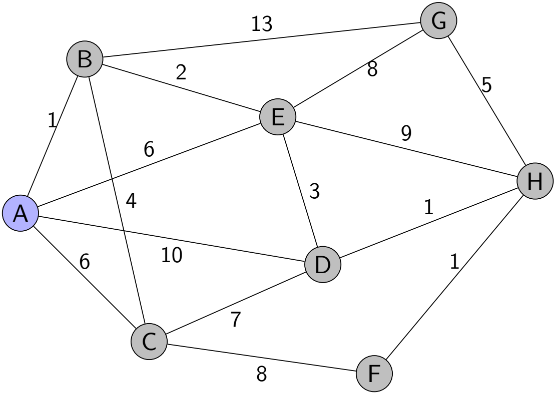
Algorithmus von Dijkstra

Algorithm 1: DIJKSTRA

Data: Graph G mit $w_{ij} \geq 0$ für alle $(i,j) \in E(G)$
Data: Startknoten s

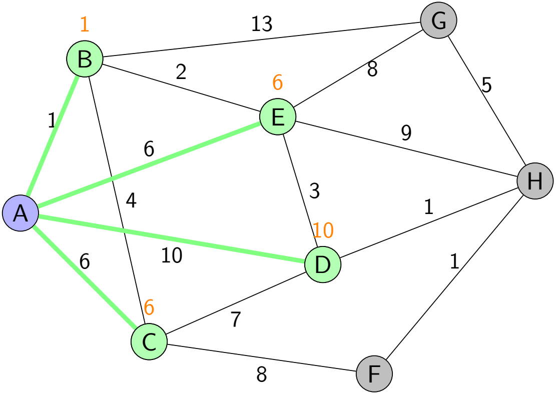
$\forall v \in V : \delta_v \leftarrow \infty$;
$\delta_s \leftarrow 0$;
Prioritätswarteschlange `Q` befüllt mit allen Knoten ;
while $Q \neq \emptyset$ **do**
    $u \leftarrow$ `Q.getMin();` // entnimmt Knoten u mit kleinstem δ_u
    Fertigstellung von Knoten u;
    Speichere Verweis auf direkten Vorgänger von u;
    for all $(u, v) \in E$, *v noch nicht fertiggestellt* **do**
        if $\delta_v > \delta_u + w_{uv}$ **then**
            $\delta_v \leftarrow \delta_u + w_{uv}$;

Algorithmus von Dijkstra: Beispiel



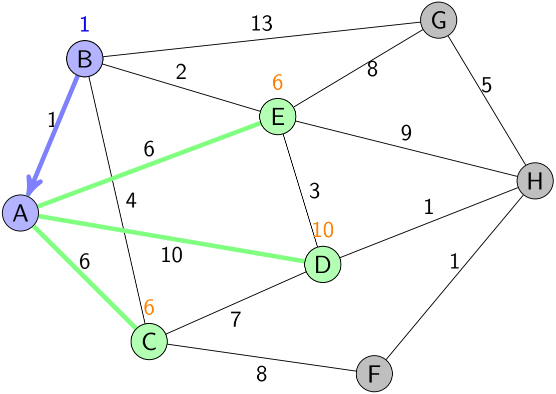
Wir suchen den kürstesten Weg vom Knoten *A* zum Knoten *H*. Im ersten Schritt wird der Startknoten "fertiggestellt".

Algorithmus von Dijkstra: Beispiel



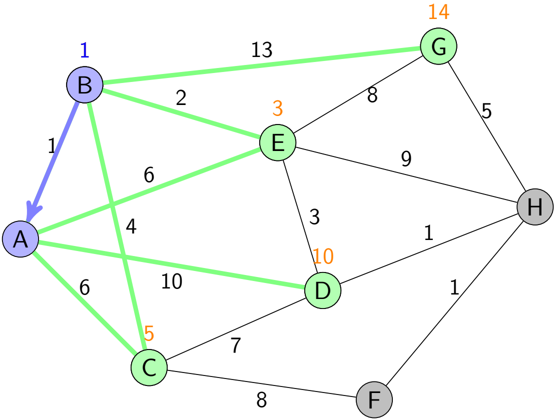
Im nächsten Schritt werden die Nachbarknoten B, C, D und E entdeckt. Die Zwischenwerte werden wie folgt berechnet:
$\delta_A = 0, \delta_B = \delta_A + 1 = 1, \delta E = \delta_A + 6 = 6, \delta_D = \delta_A + 10 = 10, \delta_C = \delta_A + 6 = 6.$

Algorithmus von Dijkstra: Beispiel



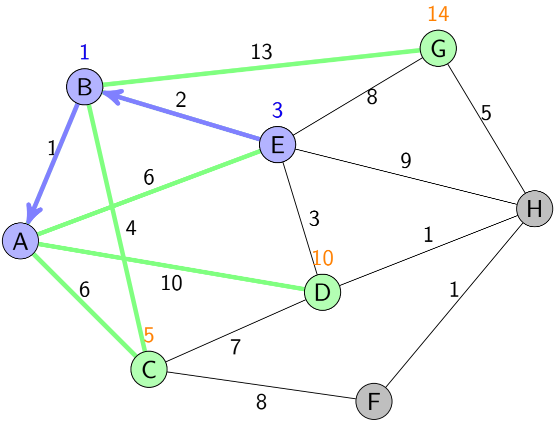
Nun wird der Knoten v mit kleinstem δ_v fertiggestellt. Im konkreten Beispiel ist dies Knoten B.

Algorithmus von Dijkstra: Beispiel



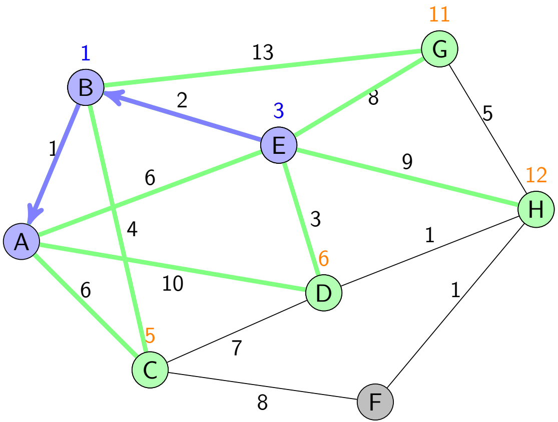
Ausgehend vom letzten fertiggestellten Knoten (B) werden nun neue Zwischenergebnisse für C, E und G berechnet. Wir erhalten $\delta_G = 1 + 13 = 14, \delta_E = 1 + 2 = 3, \delta_C = 1 + 4 = 5$. Für die Knoten C und E erhalten wir kleinere Werte als die bisher gefundenen.

Algorithmus von Dijkstra: Beispiel



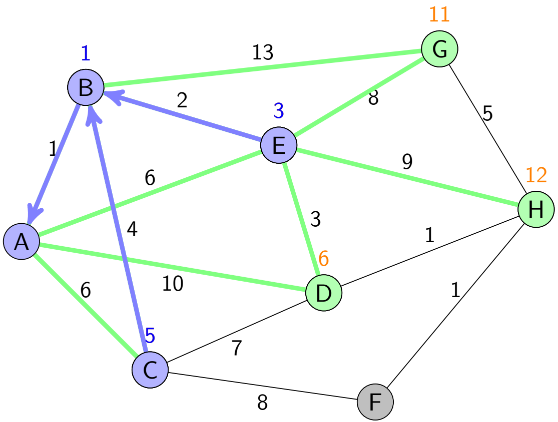
Knoten *E* wird fertiggestellt. Bei fertiggestellten Knoten merkt man sich wo man hergekommen ist (daher die blaue Kante).

Algorithmus von Dijkstra: Beispiel



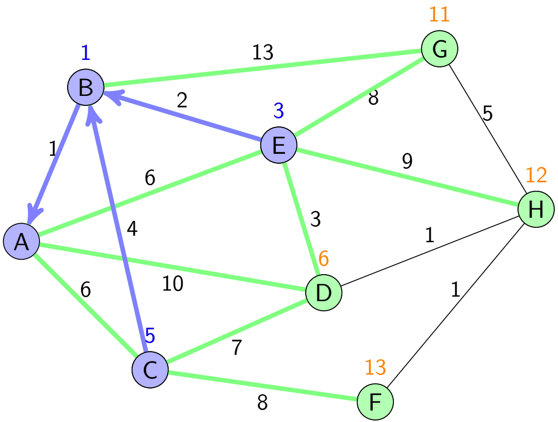
Berechnung neuer Zwischenergebnisse für D, G und H.

Algorithmus von Dijkstra: Beispiel



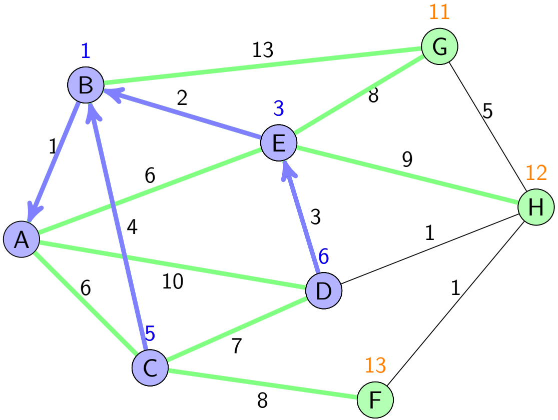
Knoten C wird fertiggestellt, da er nun den kleinsten Zwischenwert δ hat.

Algorithmus von Dijkstra: Beispiel



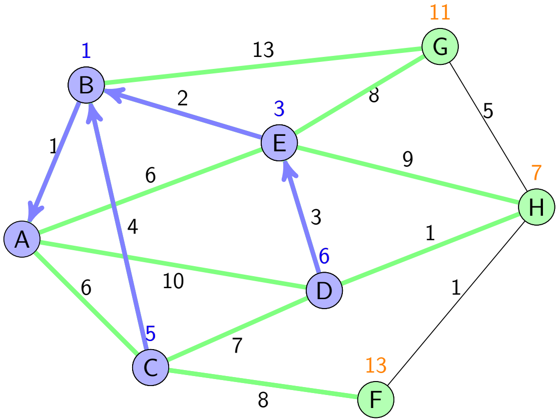
Ausgehend von C werden die Werte δ_D und δ_F berechnet. Da $5 + 7 > 6$ kommt es bei δ_D zu keiner Änderung.

Algorithmus von Dijkstra: Beispiel



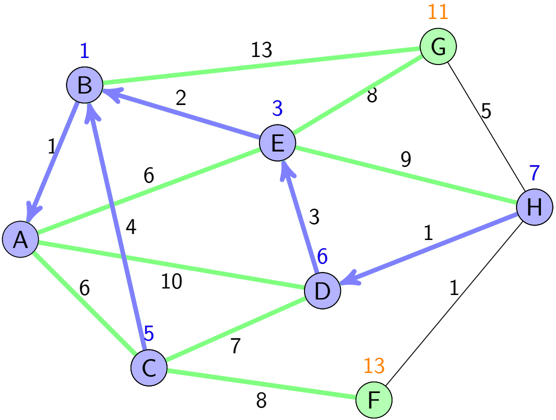
Fortgefahren wird mit Knoten D, da kleinstes δ.

Algorithmus von Dijkstra: Beispiel



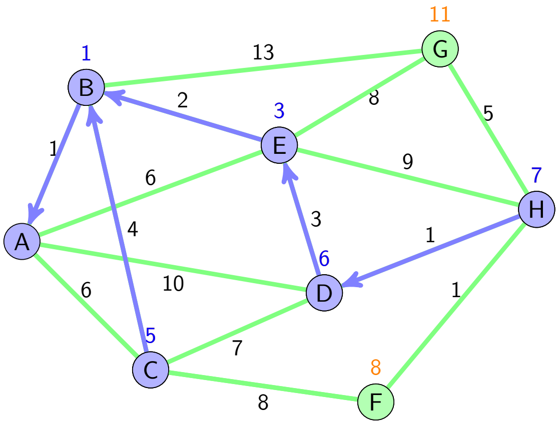
δ_H wird aktualisiert.

Algorithmus von Dijkstra: Beispiel



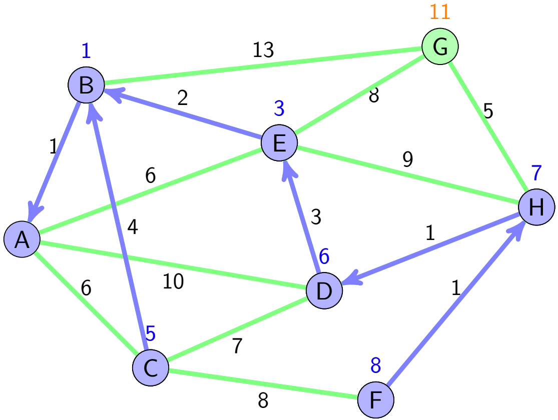
H wird fertiggestellt.

Algorithmus von Dijkstra: Beispiel



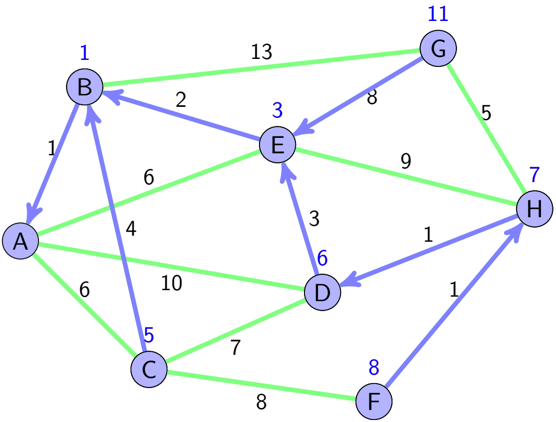
Update von δ_F und δ_G, wobei nur δ_F tatsächlich geändert wird.

Algorithmus von Dijkstra: Beispiel



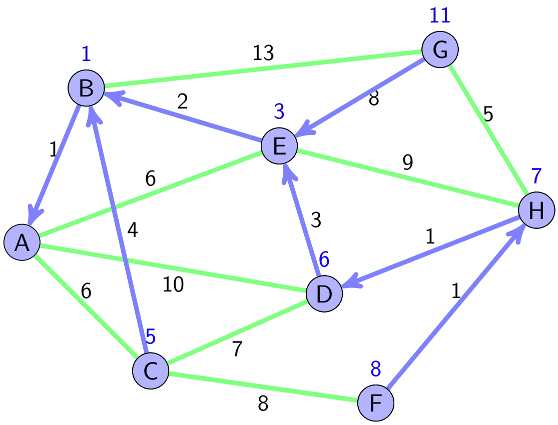
F wird fertiggestellt.

Algorithmus von Dijkstra: Beispiel



G wird fertiggestellt (Vorgänger *E*).

Algorithmus von Dijkstra: Beispiel



Die `blauen Kanten` **bilden einen Wurzelbaum, der die kürzesten Wege vom Startknoten zu jedem Knoten enthält.**

Algorithmus von Dijkstra

- Die `blauen Kanten` bilden einen Wurzelbaum, der die kürzesten Wege vom Startknoten zu jedem Knoten enthält.
- Die Berechnung des kürzesten Weges von einem Startknoten zu einem Zielknoten beinhaltet also die Berechnung der kürzesten Wege vom Startknoten zu *allen* anderen Knoten.

Algorithmus von Dijkstra – Laufzeitanalyse

Mit $n = |V|$ und $m = |E|$ können wir die Laufzeiteigenschaften angeben. Diese hängen von der konkreten Umsetzung der Prioritätswarteschlange Q ab.

Operation		Queue Implementierung		
Name	Anzahl	Liste	Heap	Fibonacci Heap[1]
decreaseKey [10]	m	$O(1)$	$O(\log n)$	$O(1)$
getMin [5]	n	$O(n)$	$O(\log n)$	$O(\log n)$
create [3]	1	$O(n)$	$O(n)$	$O(n)$
Gesamt		$O(n^2 + m) = O(n^2)$	$O((n+m)\log n)$	$O(n \log n + m)$

Anmerkung: In der Spalte ganz links ist in eckigen Klammern auf die Zeile der jeweiligen Operation im Pseudocode verwiesen!

[1] amortisierte Laufzeit