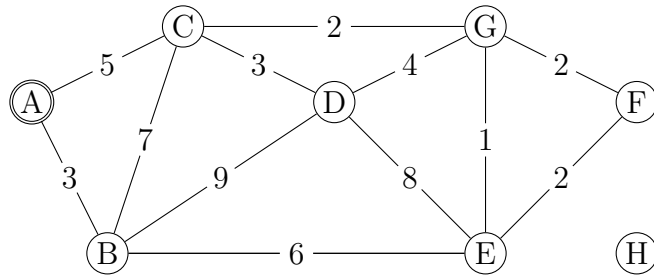


Der Algorithmus von Dijkstra

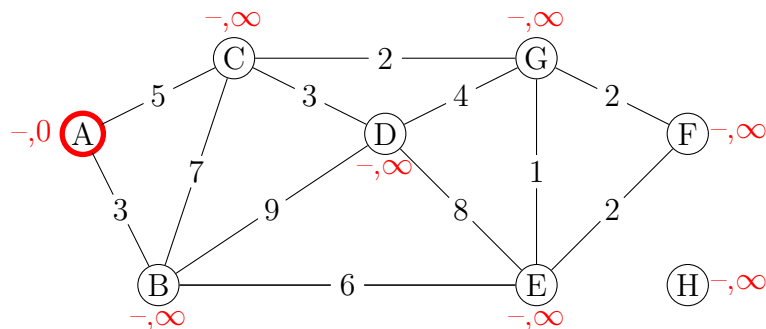
Die Ausgangslage

Gegeben: Kantengewichteter Graph G und Startknoten A

Gesucht: Kürzester Weg von A zu jedem Knoten im Graph

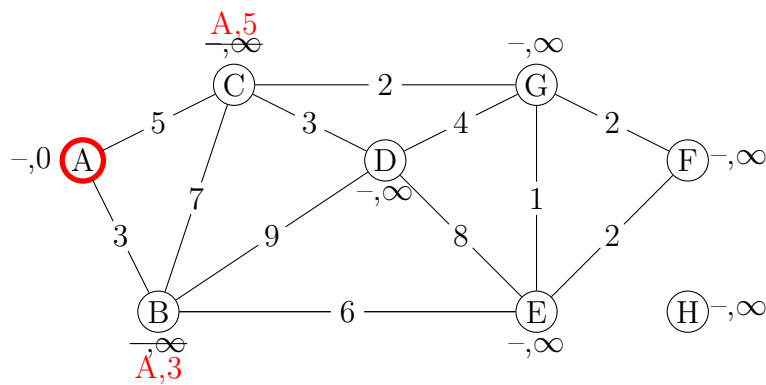


Schritt 1



Mache den Startknoten A zum ersten Arbeitsknoten (rot) und initialisiere jeden von A verschiedenen Knoten provisorisch mit einer leeren Herkunftsangabe (-) und der Distanz ∞ , was sinnvoll ist, wenn ein Knoten von A aus nicht erreichbar ist.

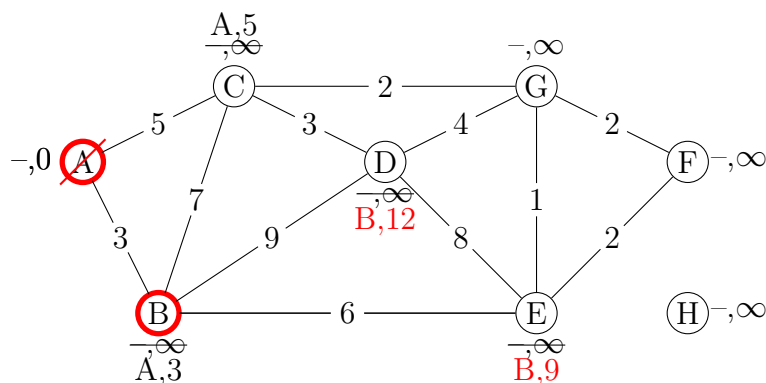
Schritt 2



Gehe vom Arbeitsknoten A (rot) zu jedem Nachbarn, der noch nicht Arbeitsknoten war und untersuche, ob die Distanz zum Zielknoten verkleinert (*relaxiert*) werden kann. Falls ja, ersetze den alten Herkunftsknoten und die alte Distanz zum Startknoten.

Vom Arbeitsknoten A sind die Knoten B und C erreichbar und noch nicht erledigt (durchgestrichen). Da der Knoten B von A die Distanz 3 hat und dies kleiner als ∞ ist, wird die alte Information (Vorgängerknoten, Gesamtdistanz zum Startknoten) ersetzt. Das gleiche mit Knoten C.

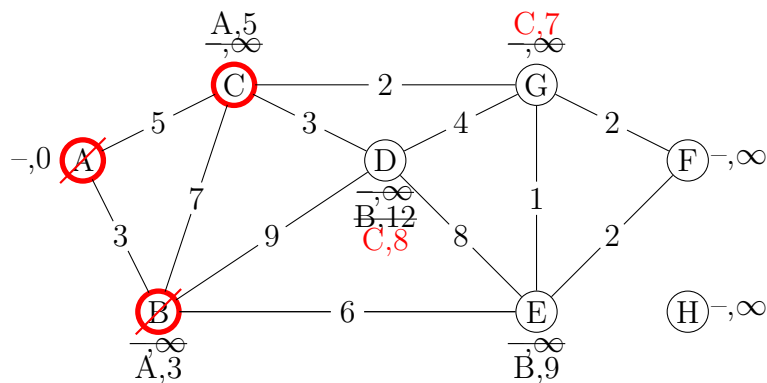
Schritt 3



Markiere den Arbeitsknoten A als erledigt und wähle unter den unerledigten Knoten den mit der kleinsten Gesamtentfernung vom Startknoten (\rightarrow B). Wiederhole Schritt 2 mit Knoten B.

Von Arbeitsknoten B sind die Knoten C, D und E erreichbar und noch nicht erledigt. Geht man von Knoten B zum Knoten C, erhält man die Gesamtdistanz $3 + 7 = 10$, was weiter ist als die Distanz 5 des Pfades von A nach C. Daher lässt sich der Weg nach C nicht über B verkürzen (*relaxieren*). Bei den anderen Knoten D und E ist dies jedoch der Fall. Beachte, dass die Distanzen *kumuliert* werden, da immer die Länge des Gesamtwegs zum Startknoten (A) berechnet wird.

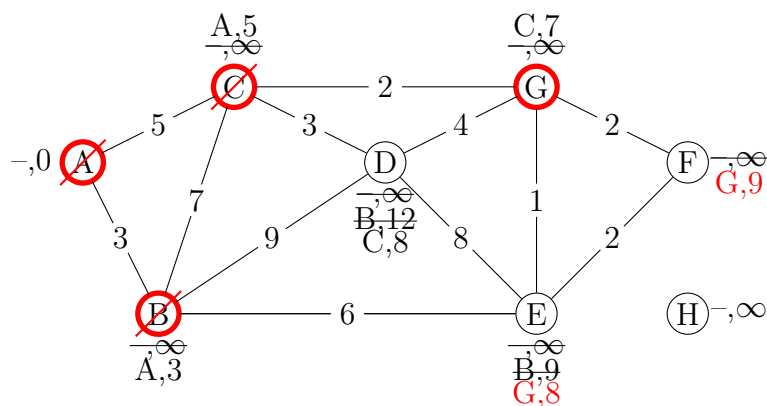
Schritt 4



Markiere Arbeitsknoten B als erledigt und wähle unter den unerledigten Knoten den mit der kleinsten Gesamtdistanz vom Startknoten (\rightarrow C). Wiederhole Schritt 2 mit Knoten C.

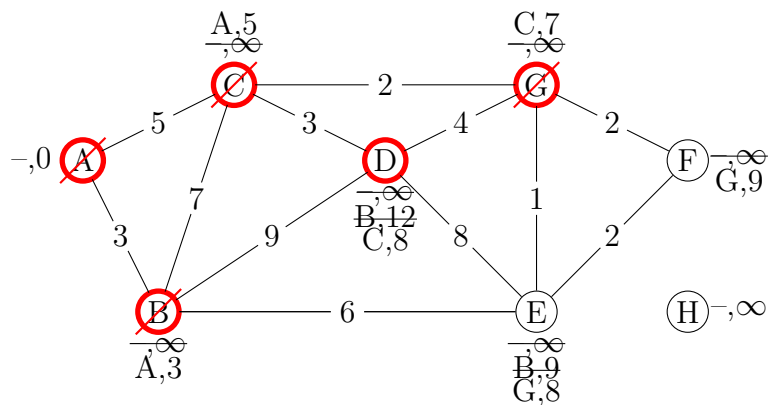
Vom Arbeitsknoten C sind die Knoten D und G erreichbar und noch nicht durchgestrichen. Geht man von Knoten C zum Knoten D, so erhält man die Gesamtdistanz $5 + 3 = 8$, was kürzer ist, als wenn man den vorher bestimmten Weg über B geht. Daher müssen die Weginformationen im Knoten D aktualisiert werden. Ebenso wenn man von C nach G geht.

Schritt 5



Markiere Arbeitsknoten C als erledigt und wähle unter den unerledigten Knoten den mit der kleinsten Gesamtdistanz vom Startknoten (\rightarrow G). Wiederhole Schritt 2 mit Knoten G.

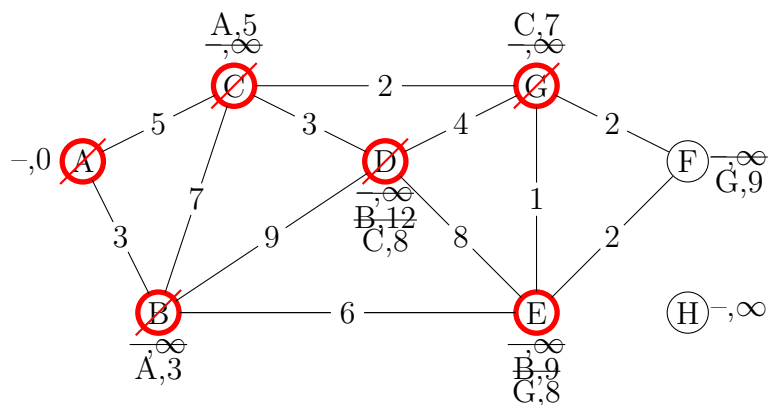
Schritt 6



Markiere Arbeitsknoten C als erledigt und wähle unter den unerledigten Knoten den mit der kleinsten Gesamtentfernung vom Startknoten. Da es zwei solche Knoten (D und E) gibt, wählen wir willkürlich Knoten D und wiederholen damit Schritt 2.

E ist der einzige noch nicht erledigte Nachbar von D. Er kann nicht relaxiert werden, weil der bisher kürzeste Weg von A nach E (via G) nicht durch den Weg über D verkürzt werden kann.

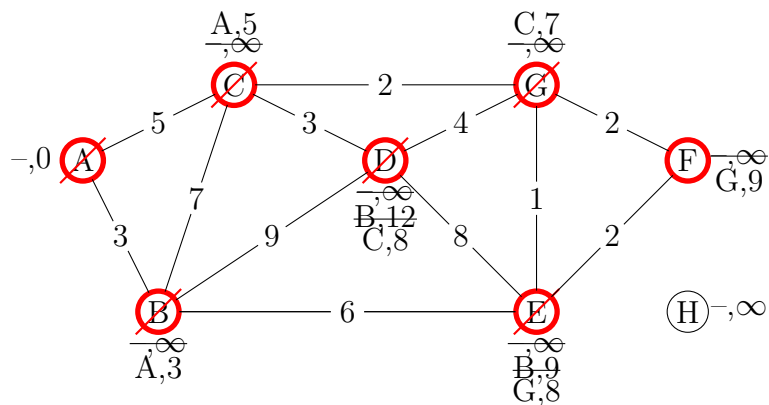
Schritt 7



Markiere Arbeitsknoten D als erledigt und wähle unter den unerledigten Knoten den mit der kleinsten Gesamtentfernung vom Startknoten (\rightarrow E). Wiederhole damit Schritt 2.

F ist der einzige noch nicht erledigte Nachbar von E. Auch er kann nicht relaxiert werden, weil der bisher kürzeste Weg von A nach F (via G) nicht durch den Weg über E verkürzt werden kann.

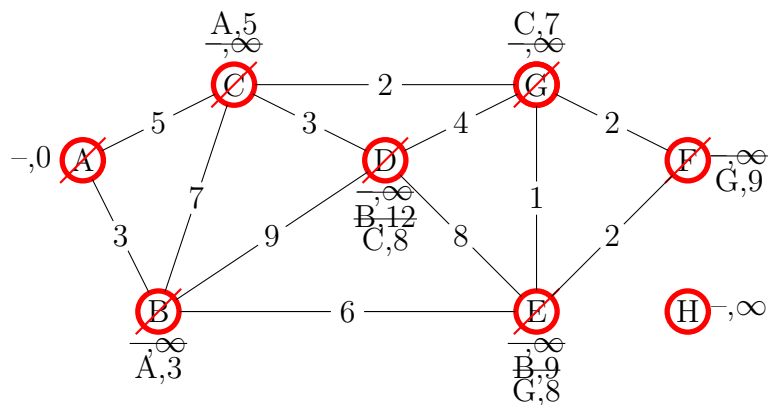
Schritt 8



Markiere Arbeitsknoten E als erledigt und wähle unter den noch unerledigten Knoten den mit der kleinsten Gesamtentfernung vom Startknoten (\rightarrow F). Wiederhole damit Schritt 2.

Hier gibt es nichts zu tun, da von F aus keine unmarkierten Nachbarn mehr erreichbar sind.

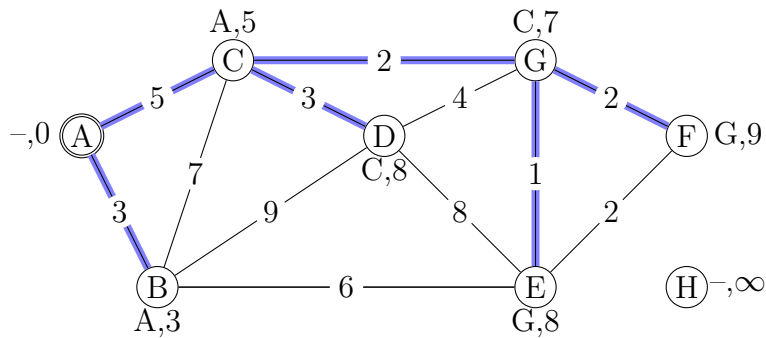
Schritt 9



Markiere Arbeitsknoten F als erledigt und wähle unter den noch unerledigten Knoten den mit der kleinsten Gesamtentfernung vom Startknoten (\rightarrow H). Da es keine weiteren unerledigten Knoten mehr gibt endet hier der Algorithmus von Dijkstra.

Grundsätzlich können wir jetzt noch den Knoten H als erledigt markieren.

Der Spannbaum



Geht man von einem Knoten v_1 zu seinem Vorgängerknoten v_2 und von dort wieder zu dessen Vorgängerknoten v_3 usw., erhält man einen kürzesten Pfad zum Startknoten A. Zusammen bilden diese Pfade einen *Spannbaum*, der im Allgemeinen aber nicht minimal sein muss, d. h. bei dem die Summe aller Kanten minimal ist.

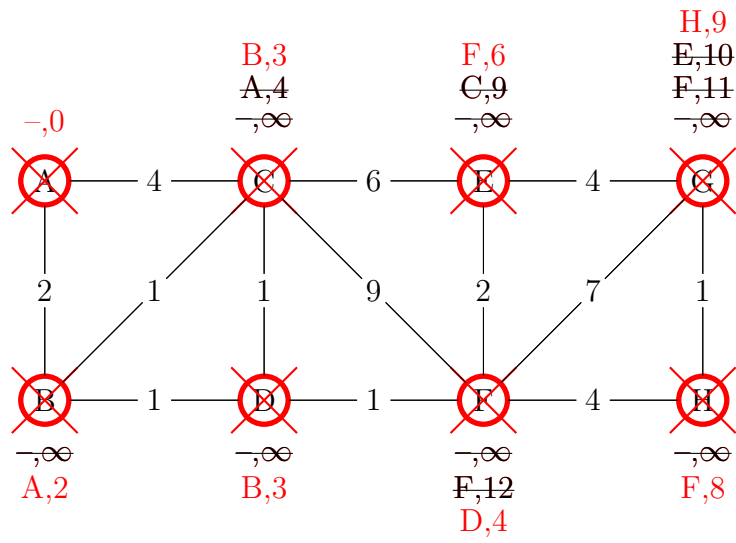
Beschreibung des Algorithmus von Dijkstra

Input: Graph G mit Kantengewichten, Startknoten S

1. *Initialisierung:* Weise jedem Knoten provisorisch seinen *Vorgänger* (-) und die Distanz zum Startknoten (0 bzw. ∞) zu. Markiere alle Knoten als unbesucht.
2. Solange es noch unbesuchte (unerledigte) Knoten gibt:
 - 2.1 Wähle darunter den Knoten v mit der kürzesten Distanz zum Startknoten und markiere ihn als besucht. Falls es mehrere solche Knoten gibt, wähle einen beliebigen aus.
 - 2.2 Berechne für alle Nachbarn w von v die Distanz $d(S, w) = d(S, v) + d(v, w)$.
 - 2.3 *Relaxation:* Ist diese Distanz kleiner als die, welche vorher im Knoten w gespeichert wurde, ersetze die alte Distanz durch die neue und speichere den neuen Vorgänger v . Andernfalls ändere nichts.

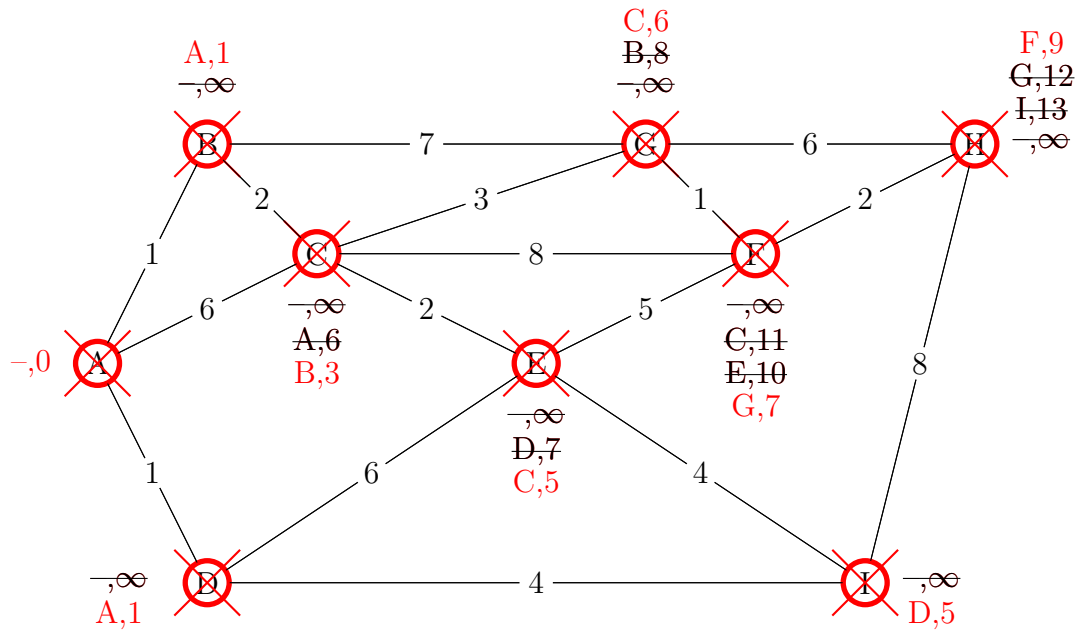
Output: Alle Knoten von G mit ihren Vorgängern und Distanzen

Aufgabe 1



Reihenfolge der Arbeitsknoten: A, B, C, D, F, E, H, G

Aufgabe 2



Reihenfolge der Arbeitsknoten: A, B, D, C, E, I, G, F, H