Shortest Path Algorithmus[1]

```
// Berechnung des Sink Tree für den Knoten v
algorithm shortestPath(v)
set  grün := ∅,          // Menge der von v aus erreichbaren Knoten
     gelb := { v };      // noch zu bearbeitende Knoten
var  dist (v) := 0;      // Abstand des Knotens v zum Startknoten (v selbst!)

while gelb ≠ ∅ do                                   // weitere Knoten analys.?
   wähle w ∈ gelb, sodass ∀ w' ∈ gelb : dist (w) ≤ dist (w');
   färbe w grün;
   for each  w_i ∈ succ (w)  do                     // für alle Nachbarn von w
      if  w_i ∉ (gelb ∪ grün)                       // w_i bisher nicht besucht
      then färbe Kante (w, w_i) rot;                // vorl. kürzest. Pfad zu w_i
           färbe w_i gelb;                          // w_i weiter zu analysieren
           dist (w_i) := dist (w) + cost (w, w_i)   // vorl. Distanz berechnen
      elsif  w_i ∈ gelb                             // w_i erneut erreicht
      then if  dist (w_i) > dist (w) + cost (w, w_i)  // kürzerer Pfad entdeckt
           then färbe Kante (w, w_i) rot;           // kürzesten Pfad aktualis.
                färbe bisher rote Kante zu w_i gelb; // bisher kürzesten entfern.
                dist (w_i) := dist (w) + cost (w, w_i); // vorl. Distanz aktualisier.
           else  färbe Kante (w, w_i) gelb          // nicht im kürzesten Pfad
           endif
      endif
   end for
end while.                                          // rote Kanten → Sink Tree
```

1) entnommen Güting, R.H.; Dieker, S.: Datenstrukturen und Algorithmen. B.G.Teuber, Wiesbaden, 2003