Breitensuche (Breadth First Search - kurz BFS)

- Möglichkeit zur Sortierung eines generellen Graphen
- Idee: G wird in einer Reihenfolge abgesucht, die immer zuerst in die Breite geht
- Jeder Knoten u von G bekommt
  - $u.d$: Zeitstempel der ersten Prüfung des Knotens
  - $u.predecessor$: Vorgänger von u in der Suche
- Außerdem: first-in first-out Queue Q

BFS

Algorithm 1: Breitensuche Algorithmus von Quelle s:

```
1  for jeden Knoten u von G do
2  |   u.predecessor = NULL
3  |   u.d = ∞
4  end
5  s.d = 0
6  ENQUEUE(Q, s)
7  while Q ≠ ∅ do
8  |   u = DEQUEUE(Q)
9  |   for jeden Knoten v mit (u, v) ∈ G do
10 |   |   if v.d == ∞ then
11 |   |   |   v.d = u.d + 1
12 |   |   |   v.predecessor = u
13 |   |   |   ENQUEUE(Q, v)
14 |   |   end
15 |   end
16 end
```

Laufzeitanalyse

- Initialisierung $O(n)$
- Jeder Knoten wird nur ein Mal in Q eingefügt
- Operationen ENQUEUE und DEQUEUE in $O(1)$, alle Operationen insgesamt in $O(n)$
- Die Adjazenzliste jedes Knotens wird ein Mal untersucht, Gesamtlänge $O(m)$
- Insgesamte Laufzeit: $O(n + m)$

Zusammenhang zu kürzesten Wegen

Definition

Die *Länge eines kürzesten Weges* $\delta(s, v)$ zwischen s und v in G ist die minimale Anzahl der Kanten in allen Pfaden von s nach v in G. Falls es keinen solchen Pfad gibt ist $\delta(s, v) = \infty$. Ein *kürzester Weg* von s und v in G ist ein Pfad von s nach v in G mit $\delta(s, v)$ Kanten.

Theorem

Bei Ende der Breitensuche enthält die Variable $v.d$ die Länge eines kürzesten Weges $\delta(s, v)$. Ein kürzester Weg kann gefunden werden, indem man den predecessor Knoten folgt bis s erreicht wird.

Kürzeste Wege in gewichteten Graphen

Definition

Gewichteter Graph: Ein Graph heißt gewichtet, wenn jeder Kante ein reelles Gewicht zugeordnet wird. Ansonsten heißt der Graph ungewichtet.

Definition

Sei G ein gewichteter Graph, und sei ω_e das Gewicht der Kante e. Die *Länge des Pfads* $p = (e_1, e_2, \ldots, e_k)$ ist $\omega_p = \sum_{i=1}^{k} \omega_{e_i}$.

Wir möchten kürzeste Wege von einem Knoten s aus berechnen.

Grundoperationen für die Berechnung

Algorithm 2: Initialisierung s:

```
1 for jeden Knoten u von G do
2 |   u.predecessor = NULL
3 |   u.d = ∞
4 end
5 s.d = 0
```

Algorithm 3: Kantenrelaxierung $(u, v), \omega_{(u,v)}$:

```
1 if v.d > u.d + ω(u,v) then
2 |   v.d = u.d + ω(u,v)
3 |   v.predecessor = u
4 end
```

Gewichtete kürzeste Wege

Lemma

Falls nach einer Sequenz von Kantenrelaxierungen $v.d < \infty$, dann gibt es einen Weg der Länge $v.d$ von s nach v.

Frage: In welcher Reihenfolge sollten die Kanten relaxiert werden?

Kürzeste Wege

Zur Berechnung kürzester Wege hilft folgendes Lemma

Lemma

Nach einer Folge R von Kantenrelaxierungen ist $v.d$ die Länge eines kürzesten Weges von s nach v, falls es einen kürzesten Weg $p = (e_1, \ldots, e_k)$ von s nach v gibt, sodass R als Teilfoge die Relaxation der Kanten $e_1, \ldots, e_k$ (in dieser Reihenfolge) enthält.
In diesen Fall enthält die predecessor Information einen kürzesten Weg von s nach v.

Einfacher Algorithmus zur Berechnung kürzester Wege

Für Graphen mit nicht-negativen Gewichten

Algorithm 4: Simple SSSP von Startpunkt s:

```
1 Initialisierung(s)
2 for i = 1 ⟶ n − 1 do
3 |   for alle Kanten (u, v) ∈ G do
4 |   |   Kantenrelaxierung((u, v), ω_(u,v))
5 |   end
6 end
```

Einfacher Algorithmus zur Berechnung kürzester Wege

Korrektheit folgt aus Eigenschaft, dass jeder kürzeste Weg maximal $n - 1$ Kanten enthält.
Laufzeit ist $O(mn)$ (folgt direkt aus Laufzeit von Initialisierung und Kantenrelaxierung).

Berechnung kürzester Wege in Graphen mit negativen Gewichten

Algorithm 5: Bellman-Ford Algorithmus von Startpunkt s:

```
1 Simple SSSP(s)
2 for alle Kanten (u, v) ∈ G do
3 |   if v.d > u.d + ω_(u,v) then
4 |   |   Infiziere(v)
5 |   end
6 end
```

Algorithm 6: Infiziere einen Knoten v:

```
1 if v.d > −∞ then
2 |   v.d = −∞
3 |   for Kanten (v, w) ausgehend von v do
4 |   |   Infiziere(w)
5 |   end
6 end
```

Bellman-Ford Algorithmus

- Fall von negativem Zyklus: nach Ausführung von Simple SSSP(s) gibt es einen Knoten v mit $v.d > u.d + \omega_{(u,v)}$
- Bellman-Ford Algorithmus setzt die Distanz $v.d$ auf $-\infty$
- Anschließend werden Gewichte der von v erreichbaren Knoten auf $-\infty$ gesetzt
- Methode *Infiziere*(v) ist der Tiefensuche ähnlich
- Laufzeit: $O(mn)$

Berechnung kürzester Wege in Graphen mit nicht-negativen Gewichten

Frage Kann man im Fall nicht-negativer Gewichte kürzeste Wege auch schneller berechnen?

Idee Reihenfolge der Kantenrelaxierungen gut auswählen

DS PQ: Min-Priority Queue, die Knoten v mit Prioritäten $v.d$ enthält

S: Menge an Knoten, für die bereits ein kürzester Weg gefunden wurde

Dijkstra Algorithmus

Für Graphen mit nicht-negativen Gewichten

Algorithm 7: Dijkstra Algorithmus von Startpunkt s:

```
1 Initialisierung(s)
2 S = ∅
3 while PQ ≠ ∅ do
4 |   u = EXTRACT-MIN(PQ)
5 |   S = S ∪ u
6 |   for Kanten (u, v) ausgehend von u do
7 |   |   Kantenrelaxierung((u, v), ω_(u,v))
8 |   end
9 end
```

Grundoperationen für die Berechnung

Algorithm 8: Initialisierung s:

```
1 for jeden Knoten u von G do
2 |   u.predecessor = NULL
3 |   u.d = ∞
4 |   Füge u in Prioritätenschlange PQ ein
5 end
6 s.d = 0
7 Füge s in Prioritätenschlange PQ ein
```

Algorithm 9: Kantenrelaxierung $(u, v), \omega_{(u,v)}$:

```
1 if v.d > u.d + ω(u,v) then
2 |   v.d = u.d + ω(u,v)
3 |   Decrease-Key von v in Prioritätenschlange PQ
4 |   v.predecessor = u
5 end
```

Dijkstra Algorithmus

Theorem

Für einen gewichteten Graphen G mit nicht-negativen Gewichten berechnet der Dijkstra Algorithmus alle kürzesten Wege vom Startpunkt s. Bei der Terminierung des Algorithmus sind die Längen der kürzesten Wege $v.d = \delta(s, v)$, und die kürzesten Wege sind in den predecessor Informationen enthalten.

Beweisidee verwendet die Invariante, dass bei Beginn der While-Schleife (in Zeile 5) gilt, dass $u.d = \delta(s, u)$.

Dijkstra Algorithmus

Laufzeit

- Initialisierung: $O(n)$
- Min-Priority Queue Operationen
  - $O(n)$ Mal Kosten für Einfügen
  - $O(n)$ Mal Kosten für EXTRACT-MIN
  - $O(m)$ Mal Kosten für DECREASE-KEY (in Kantenrelaxierung)