4 Kürzeste Wege

Ash Ketchum möchte die kürzeste Strecke von der Universität Trier (Campus 2) zur Pokémon-Arena Porta Nigra bestimmen. Er hat dafür eine Straßenkarte von Trier, auf der die Abstände zwischen allen Paaren benachbarter Kreuzungen eingezeichnet sind.
In diesem Szenario stellen die Kosten einer Kante die Distanz zweier Kreuzungen dar. In anderen Szenarien können die Kosten einer Kante aber auch andere Bedeutungen haben. Sie können z.B. tatsächliche Kosten darstellen, wenn es um Energieverbrauch oder das Herstellen von Produkten geht. Treten in diesen Szenarien negative Kosten auf, können diese Gewinnen entsprechen.

4.1 Allgemeines Problem

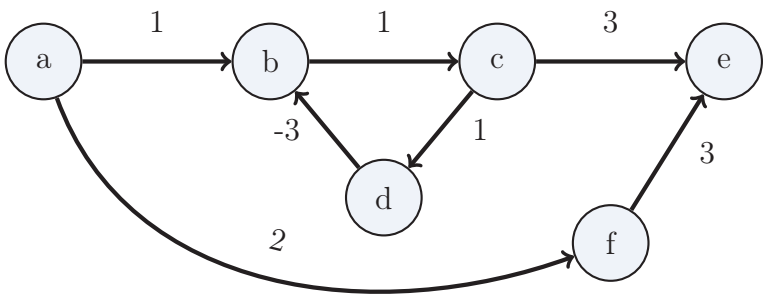
Gegeben:

Gerichteter Graph: $G = (V, E)$

Kostenfunktion: $cost : E \to \mathbb{R}$

$cost(e)$ = Kosten der Kante e
Anstatt $cost((v, w))$ *schreiben wir nur* $cost(v, w)$.

Beispiel 4.1



4.2 Definitionen

4.2.1 Pfad

Ein Pfad kann sowohl über die Knoten, als auch über die Kanten definiert werden, die er enthält.

Definition über Knoten

Ein Pfad von $s \in V$ nach $w \in V$ in einem gerichteten Graphen $G = (V, E)$ ist eine Folge $P = v_0, \ldots, v_k$ mit $v_0 = s$ und $v_k = w$.

Definition über Kanten

Ein Pfad von $s \in V$ nach $w \in V$ in einem gerichteten Graphen $G = (V, E)$ ist eine Folge $P = (e_1, e_2, \ldots, e_n)$ von Kanten $e_i \in E$.

Länge eines Pfades

Die Länge eines Pfades wird oft über die Anzahl der enthaltenen Kanten definiert. Hier allerdings **nicht**!
Trotzdem sprechen wir von kürzesten Pfaden, da das Problem im Englischen „shortest path“ genannt wird und es oft mit „kürzestem Weg“ übersetzt wird. Da oft von billigsten Pfaden gesprochen wird, wenn es um die Kosten eines Pfades geht, werden in diesem Skript die Bezeichnungen „kürzeste Wege“ und „billigste Pfade“ (oder Mischformen daraus) synonym verwendet.

Einfacher Pfad

Ein einfacher Pfad enthält keinen Knoten mehrfach.
Erfüllt ein Pfad diese Eigenschaft nicht nennt man ihn „nicht einfachen Pfad“.

4.2.2 Kosten eines Pfades

Die Kosten eines Pfades ist die Summe der Kosten der Kanten entlang des Pfades.

$$\boldsymbol{cost}(\boldsymbol{P}) := \sum_{\boldsymbol{e} \in \boldsymbol{P}} \boldsymbol{cost}(\boldsymbol{e})$$

Achtung: Eine Kante kann mehrfach vorkommen.

Im Beispiel 4.1:

$$cost(a \to f \to e) = 5$$
$$cost(a \to b \to c \to e) = 5$$
$$cost(a \to b \to c \to d \to b \to c \to e) = 4$$

4.2.3 Distanz zweier Knoten

Die Distanz zweier Knoten entspricht dem kürzesten Weg zwischen diesen beiden Knoten.
Da es durch negative Zyklen unendlich viele Wege geben kann, wird hier nicht das Minimum, sondern das Infimum (s. 1.2.2) genutzt.

$$\boldsymbol{dist(v,w) := inf\{cost(P) \mid P \text{ ist Pfad von v nach w}\}}$$

dh. $dist(v,w) = -\infty$, falls negativer Zyklus auf P existiert.
$dist(v,w) = +\infty$, falls kein Pfad von v nach w existiert.

Im Beispiel 4.1:

$$dist(a,f) = 2$$
$$dist(e,a) = +\infty$$
$$dist(a,e) = -\infty \qquad \text{Pfad: } a \to b \to (c \to d \to b \to c)^i \to e$$
Kosten: $4 - i$ dh. beliebig klein.

4.3 Varianten des Problems

Es existieren verschiedene Varianten des Kürzesten-Wege Problems. Dabei wird unterschieden, wie viele source und target Knoten es gibt.

i) **Single-Source-Single-Target**
$dist(s,t) \quad s,t \in V$
Zwischen zwei gegebenen, festen Knoten wird die Distanz gesucht.

ii) **Single-Source-Shortest-Path**
$dist(s,v) \quad \forall\, v \in V$
Von einem gegebenen Knoten s werden die kürzesten Distanzen zu allen anderen Knoten gesucht.

iii) **All-Pairs-Problem**
$dist(v, w) \quad (v, w) \in VxV$
Gesucht ist die Distanz zwischen allen Knotenpaaren. ($\rightarrow$ Entfernungstabelle)

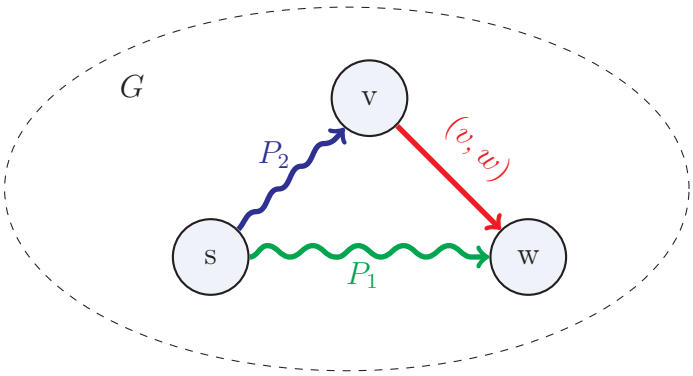
4.4 Single-Source-Shortest-Path

Ab hier wird das Single-Source-Shortest-Path Problem betrachtet.

Eingabe: $G = (V, E), \quad s \in V, \quad cost : E \rightarrow \mathbb{R}$
Ausgabe: $\forall\ v \in V \quad dist(s, v)$ (und entsprechende Pfade)

Beobachtung:
Die *dist*-Funktion erfüllt die Δ-Ungleichung.
Für jede Kante $(v, w) \in E$ gilt:

$$dist(s, w) \leq dist(s, v) + cost(v, w)$$



P_1 ist ein kürzester Pfad von s nach w.
P_2 ist ein kürzester Pfad von s nach v.

Herleitung der Δ-Ungleichung:
$dist(s, w) = cost(P_1)$
$dist(s, v) = cost(P_2)$

$dist(s, w) \leq cost(P_2 + (v, w))$ da P_1 kürzester Pfad von s nach w
$\Leftrightarrow dist(s, w) \leq cost(P_2) + cost(v, w)$
$\Leftrightarrow dist(s, w) \leq dist(s, v) + cost(v, w$

4.4.1 Allgemeiner Algorithmus (Label correcting)

Aus diesen Beobachtungen kann leicht ein Algorithmus abgeleitet werden.

Ideen:

- Überschätze die *dist*-Werte ($DIST = \infty$)
- Solange eine Kante (u, v) die Δ-Ungleichung verletzt: Korrigiere $DIST[v]$ dh. $DIST[v] \leftarrow DIST[u] + cost(u, v)$

Algorithmus 1 : Allgemeiner Algorithmus

```
1 foreach v ∈ V do
2 |   DIST[v] ← ∞
3 end
4 DIST[s] ← 0
5 while ∃ Kante (u, v) ∈ E mit DIST[v] > DIST[u] + cost((u, v)) do
6 |   DIST[v] ← DIST[u] + cost((u, v))
7 end
```

Lemma 1

a) Es gilt immer $DIST[v] \geq dist(s, v)$

b) Wenn $DIST[v] < \infty$, dann existiert ein Pfad von s nach v mit Kosten $DIST[v]$

c) Die kürzesten Pfade bilden einen Baum T mit Wurzel s.

d) Für jede Kante (v, w) auf einem kürzesten Pfad gilt:
$dist(s, w) = dist(s, v) + cost((v, w))$ (Δ-Ungleichung mit Gleichheit)

Erklärungen zu Lemma 1:

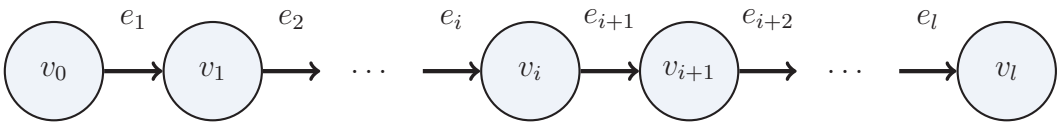
a) $dist(s, v)$ sind die Kosten des kürzesten Pfades von s nach v. $DIST[v]$ ist zu Beginn $+\infty$ und wird im Laufe des Algorithmus, falls mindestens ein Pfad von s nach v existiert, auf die Kosten dieses Pfades verringert. Somit kann

der Wert von $DIST[v]$ auch nie kleiner als die Kosten des kürzesten Pfades ($dist(s, v)$) werden. Ab dem Zeitpunkt, an dem der Algorithmus den kürzesten Pfad findet, gilt: $DIST[v] = dist(s, v)$

b) Sobald der Algorithmus einen Pfad von s nach v findet, berechnet er die Kosten dieses Pfades und setzt dieses Ergebnis als neuen $DIST$-Wert von v. Da der Pfad und die Kosten der einzelnen Kanten endlich sind, ist das Ergebnis und somit $DIST[v]$ echt kleiner als $+\infty$.

c) In jedem Knoten mit $DIST[v] < \infty$ mündet genau ein kürzester Pfad. Bei Korrektur des $DIST$-Wertes wird die eingehende Kante von T definiert.

d) Die Δ-Ungleichung mit Gleichheit ergibt sich aus der Definition von $dist(s, w)$.



$$dist(s, v_i) = \sum_{k=1}^{i} e_k$$

$$dist(s, v_{i+1}) = \sum_{k=1}^{i} e_k + cost(v_i, v_{i+1})$$

$$dist(s, v_{i+1}) = \sum_{k=1}^{i} e_k + e_{i+1}$$

$$dist(s, v_{i+1}) = \sum_{k=1}^{i+1} e_k$$

(v_i, v_{i+1}) verbindet den Knoten v_{i+1} mit dem auf dem kürzesten Weg davor liegenden Knoten.

4.4.2 Verfeinerter Algorithmus

Der allgemeine Algorithmus lässt viel Freiheit bei der Wahl des Knotens. Dadurch kann der Knoten sehr ungeschickt gewählt werden. Liegt der Knoten sehr weit hinten im Graph, müssen die $DIST$-Werte der hinteren Knoten immer wieder angepasst werden, wenn bei einem auf dem Pfad davor liegenden Knoten der $DIST$-Wert verringert wurde. Daher beginnt der folgende Algorithmus beim Startknoten, sodass

die $DIST$-Werte wie eine Welle durch den Graphen vom Startknoten aus verändert werden.

Damit der Algorithmus nicht mehr jedes mal alle Kanten überprüfen muss, wird eine Kandidatenmenge gebildet. In dieser Kandidatenmenge U werden alle Knoten, aus denen Kanten ausgehen, die die Δ-Ungleichung verletzen können, gespeichert. Dadurch muss der Algorithmus immer nur die Kandidatenmenge abarbeiten und arbeitet somit effizienter.
Am Anfang besteht U nur aus dem Startknoten s. $U = \{s\}$
Immer, wenn $DIST[v]$ vermindert wird, wird v in die Menge U aufgenommen, da die Δ-Ungleichung verletzt worden sein könnte.

Algorithmus 2 : Verfeinerter Algorithmus

```
1  foreach v ∈ V do
2  |  DIST[v] ← ∞
3  end
4  DIST[s] ← 0
5  U = {s}
6  while U ≠ ∅ do
7  |  wähle und entferne ein u ∈ U
8  |  foreach v ∈ V mit (u, v) ∈ E do
9  |  |  c = DIST[u] + cost((u, v))
10 |  |  if c < DIST[v] then
11 |  |  |  DIST[v] = c
12 |  |  |  U = U ∪ {v}
13 |  |  end
14 |  end
15 end
```