Algorithmus von Bellman-Ford

Vorbereitung *l_{ij}: Länge der Kante $v_i v_j$. l_{ij}:=∞, falls es eine solche Kante nicht gibt. Für jede Ecke $v_i \in V$ werden zwei Variable angelegt:*

1. **Entf$_i$**: *die bisher kürzeste Entfernung von v_1 nach v_i an. Startwert 0 für i=1 und ∞ sonst.*
2. **Vorg$_i$** *Vorgänger von v_i auf dem bisher kürzesten Weg von v_1 nach v_i an. Startwert v_1 für i=1 und undefiniert sonst.*

Iteration *Wiederhole |V|-1 mal*

*Für **alle** Kanten ($v_i v_j$) aus E*

Falls gilt $Entf_j > Entf_i + l_{ij}$ dann

Setze $Entf_j := Entf_i + l_{ij}$

Setze $Vorg_j := i$

Für alle Kanten ($v_i v_j$) aus E

Falls gilt $Entf_j > Entf_i + l_{ij}$ dann

STOP mit Ausgabe „Zyklus negativer Länge gefunden“

Hinweis: Da **über alle Kanten iteriert wird**, ist bei einem ungerichteten Graphen darauf zu achten, dass die Kanten in beide Richtungen untersucht werden!

FiFo-Algorithmus

Vorbereitung *l_{ij}: Länge der Kante $v_i v_j$. l_{ij}:=∞, falls es eine solche Kante nicht gibt. Zur Verwaltung wird eine Warteschlange* **queue** *eingesetzt. Am Anfang ist v_1 in queue enthalten. Für jede Ecke $v_i \in V$ werden zwei Variable angelegt:*

1. **Entf$_i$**: *die bisher kürzeste Entfernung von v_1 nach v_i an. Startwert 0 für i=1 und ∞ sonst.*
2. **Vorg$_i$** *Vorgänger von v_i auf dem bisher kürzesten Weg von v_1 nach v_i an. Startwert v_1 für i=1 und undefiniert sonst.*

Iteration *Wiederhole (i,j seien dabei die Laufvariablen, h ein fester Wert)*

v_h := pop(queue).

Für alle Ecken v_j für die eine Kante $v_h v_j$ existiert:

Falls gilt $Entf_j > Entf_h + l_{hj}$ dann

Setze $Entf_j := Entf_h + l_{hj}$

Setze $Vorg_j := h$

push(v_j,queue)

solange queue nicht leer ist.

Floyd-Warshall-Algorithmus

Setze $d_{ij} := l_{ij}$ *für* $v_i v_j \in E$ *und* $i \neq j$ | 0 *für* $i = j$ | ∞ *sonst*

$t_{ij} := 0$

Für $j = 1, \dots, |V|$:

Für $i = 1, \dots, |V|;\ i \neq j$:

Für $k = 1, \dots, |V|;\ k \neq j$:

Setze $d_{ik} := \min\{d_{ik},\ d_{ij} + d_{jk}\}$.

Falls d_{ik} *verändert wurde, setze* $t_{ik} := j$.

Falls $d_{ii} < 0$ *ist, brich den Algorithmus vorzeitig ab.*

(Es wurde ein Kreis negativer Länge gefunden.)

Floyd-Warshall-Algorithmus

		j		k
	0	1	∞	3
j	∞	0	2	1
i	∞	2	0	∞
	2	∞	2	0

$d_{ik} = \infty$

$d_{ij} = 2$

$d_{jk} = 1$

Floyd-Warshall-Algorithmus

0	1	∞	3
∞	0	2	1
∞	2	0	∞
2	∞	2	0

d_{ik}, = 0
d_{ij} = ∞
d_{jk} = 1

j	i	k
1	2	2

Floyd-Warshall-Algorithmus

0	1	∞	3
∞	0	2	1
∞	2	0	∞
2	∞	2	0

d_{ik}, = 2
d_{ij} = ∞
d_{jk} = ∞

j	i	k
1	2	3

Floyd-Warshall-Algorithmus

0	1	∞	3
∞	0	2	1
∞	2	0	∞
2	∞	2	0

d_{ik}, = 1
$d_{ij} = \infty$
$d_{jk} = 3$

j	i	k
1	2	4

Informatik
Hochschule für angewandte Wissenschaften Hamburg

Floyd-Warshall-Algorithmus

0	1	∞	3
∞	0	2	1
∞	2	0	∞
2	∞	2	0

d_{ik}, = 2
$d_{ij} = \infty$
$d_{jk} = 1$

j	i	k
1	3	2

Floyd-Warshall-Algorithmus

0	1	∞	3
∞	0	2	1
∞	2	0	∞
2	∞	2	0

d_{ik} = 0
d_{ij} = ∞
d_{jk} = ∞

j i k
1 3 3

Floyd-Warshall-Algorithmus

0	1	∞	3
∞	0	2	1
∞	2	0	∞
2	∞	2	0

d_{ik} = ∞
d_{ij} = ∞
d_{jk} = 3

j i k
1 3 4

Floyd-Warshall-Algorithmus

0	1	∞	3
∞	0	2	1
∞	2	0	∞
2	∞	2	0

$d_{ik} = \infty$
$d_{ij} = 2$
$d_{jk} = 1$

j i k
1 4 2

Floyd-Warshall-Algorithmus

0	1	∞	3
∞	0	2	1
∞	2	0	∞
2	3	2	0

$d_{ik} = \infty$
$d_{ij} = 1$
$d_{jk} = 2$

j i k
2 1 3

Informatik

Floyd-Warshall-Algorithmus

0	$\boxed{1}$	3	$\boxed{3}$
∞	0	2	$\boxed{1}$
∞	2	0	∞
2	3	2	0

d_{ik} = 3
d_{ij} = 1
d_{jk} = 1

j i k
2 1 4

Informatik

Floyd-Warshall-Algorithmus

0	1	3	2
∞	0	2	$\boxed{1}$
∞	$\boxed{2}$	0	$\boxed{\infty}$
2	3	2	0

d_{ik} = ∞
d_{ij} = 1
d_{jk} = 2

j i k
2 3 4

Floyd-Warshall-Algorithmus

0	1	3	2
$\boxed{\infty}$	0	2	$\boxed{1}$
∞	2	0	3
$\boxed{2}$	3	2	0

d_{ik}, = ∞
d_{ij} = 1
d_{jk} = 2

j	i	k
4	2	1

Floyd-Warshall-Algorithmus

0	1	3	2
3	0	2	1
$\boxed{\infty}$	2	0	$\boxed{3}$
$\boxed{2}$	3	2	0

d_{ik}, = ∞
d_{ij} = 3
d_{jk} = 2

j	i	k
4	3	1

Floyd-Warshall-Algorithmus

0	1	3	2
3	0	2	1
5	2	0	3
2	3	2	0

A*-Algorithmus

Es seien

K: *Knotenmenge;* T: *Menge der Terminalknoten;* s: *Startknoten;*

y(n): *ein Pfad vom Knoten n zu einem Knoten aus T;* Succ(n): *Menge der Nachfolger von n;*

k(n, m): *billigste Kosten von n zu m; dies bleibt undefiniert, falls kein Pfad von n nach m existiert;*

g_y(n): *Kosten des Pfades y von s zu n;* g*(n): *Kosten des billigsten Pfades von s zum Knoten n;*

g(n): *Kosten des billigsten bisher bekannten Pfades von s zu n;*

h*(n): *Kosten des billigsten Pfades von n zu einem Knoten aus T;*

h(n): *Schätzung der Kosten des billigsten Pfades von n zu einem Knoten aus T;*

f(n) = g(n) + h(n) *Schätzung des billigsten Pfades durch n zu einem Knoten aus T;*

f*(n) = g*(n) + h*(n) *optimale Kosten eines Pfades durch n zu einem Knoten aus T.*

K* = h*(s): *minimale Kosten überhaupt;*

Es gilt z.B.: g(n) ≥ g*(n) und f*(s) = g*(s)+h*(s) = 0+h*(s) = K*.

|h *(n) − h(n)| möglichst klein:

1. *Pessimistische Strategie:* h*(n) ≤ h(n) *für jeden Knoten n;*
2. *Optimistische Strategie: h(n) ≤ h*(n).*

A*-Algorithmus

1. *Setze g(s) = 0, CLOSED = [], OPEN = [s] , WEG(s) = [s] und berechne h(s) sowie f(s) = g(s) + h(s).*
2. *Wenn OPEN = [], dann STOP mit negativem Ausgang (d.h. es wurde keine Lösung gefunden).*
3. *Wenn OPEN =/= [], dann wähle n aus OPEN mit f(n) minimal, setze OPEN = OPEN\(n) , CLOSED = [n] + CLOSED.*
4. *Wenn n ∈ T, dann STOP mit positivem Ausgang (mit Gesamtkosten g(n)).*
5. *Wenn nicht n∈ T, dann betrachte alle n' ∈ Succ(n):*
   a) *Wenn n' weder auf OPEN noch auf CLOSED, dann setze WEG(n') = [n'] + WEG(n), OPEN = [n'] + OPEN, g(n') = g(n) + k(n,n') und berechne f(n') = g(n') + h(n').*
   b) *Wenn n' auf OPEN oder CLOSED und g(n) + k(n,n') < g(n'), dann setze WEG(n') = [n'] + WEG[n] und g(n') = g(n) + k(n,n') und berechne f(n') = g(n') + h(n'). Falls speziell n' ∈ CLOSED, dann setze OPEN = [n'] + OPEN und CLOSED = CLOSED\(n').*
6. *Gehe zurück zu 2.*

A*-Algorithmus

- *Breitensuche:* k(n,n') = 1 und h(n) = 0
- *Tiefensuche:* k(n,n') = -1 und h(n) = 0

Wenn überhaupt eine Lösung existiert (d.h. T =/= Ø), dann terminiert der A-Algorithmus mit einem t ∈ T. Für optimistisches h liefert der Algorithmus bereits eine optimale Lösung.*

1. *Wenn ein Knoten n expandiert wird, dann gilt f(n) ≤ K*.*
2. *Jeder Knoten n ∈ OPEN mit f(n) < K* wird auch tatsächlich expandiert.*
3. *A* schaltet Knoten n mit f(n) > K* endgültig aus.*

Für zwei optimistische Schätzfunktionen h_1 und h_2 heißt h_2 ***besser informiert*** *als h_1, falls $h_1(n) < h_2(n)$ für jeden Knoten n ∈ T gilt.*

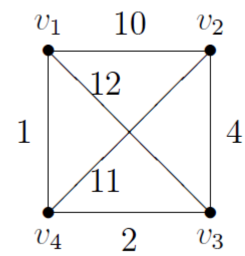
h heißt ***monoton****, falls stets h(n) ≤ k(n,n') + h(n') für n' ∈ Succ(n) gilt.*
Monotone Schätzfunktionen sind optimistisch.

A*-Algorithmus und Dijkstra-Algorithmus

- **Gegeben A*-Algorithmus**: setze Heuristik h(n) = 0. Die Variable OK wird dann durch die OPEN-Liste (OK = false) und die CLOSED-Liste (OK = true) abgebildet. So erhält man einen **Dijkstra-Algorithmus**.
- **Gegeben Dijkstra-Algorithmus**: bei einer monotonen Heuritsik wird diese in die Kostenfunktion eingerechnet und realisiert damit den zugehörigen **A*-Algorithmus**.
- **Gegeben A*-Algorithmus**: setze Kostenfunktion g(n) = 0 ergibt einen **Greedy-Algorithmus**. Greedy sucht den durch eine lokale Bewertung vorgenommene, also die Heuristik h(n), günstigsten Nachfolger im Sinne einer gierigen Vorgehensweise.

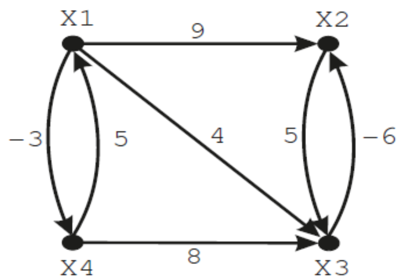
Aufgaben

1. Führen Sie den Algorithmus von Floyd-Warshall mit nachfolgendem Graphen durch (bis zum Abbruch!). Fertigen Sie eine Dokumentation an, aus der der Ablauf deutlich wird, insbesondere der Verlauf der Laufindizes. Geben Sie am Ende alle kürzesten Wege an, die keine direkten Verbindungen sind.



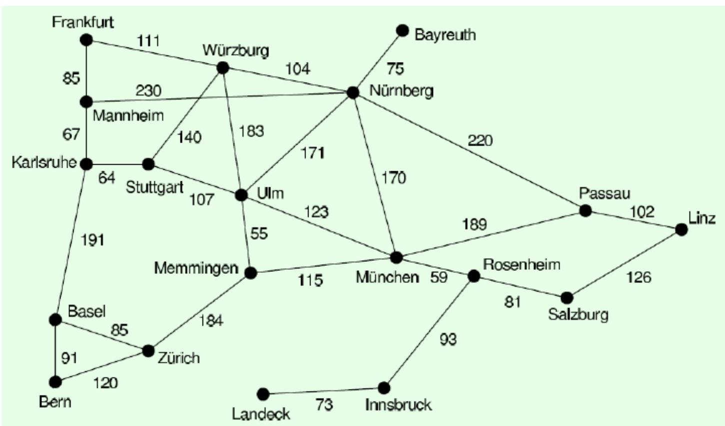
Aufgaben

2. Führen Sie den Algorithmus von Floyd-Warshall mit nachfolgendem Graphen durch (bis zum Abbruch!). Fertigen Sie eine Dokumentation an, aus der der Ablauf deutlich wird.



Aufgaben

2. Bestimmen Sie mittels A*-Algorithmus den optimalen Weg von Mannheim nach Ulm und von Linz nach Ulm. Die Kostenfunktion k ist links abgebildet und die Schätzfunktion h rechts



Luftlinie nach Ulm:

Basel	204
Bayreuth	207
Bern	247
Frankfurt	215
Innsbruck	163
Karlsruhe	137
Landeck	143
Linz	318
München	120
Mannheim	164
Memmingen	47
Nürnberg	132
Passau	257
Rosenheim	168
Stuttgart	75
Salzburg	236
Würzburg	153
Zürich	157