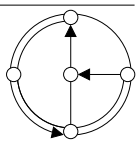
ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich



SS 2007 Prof. R. Wattenhofer / Michael Kuhn / Remo Meier

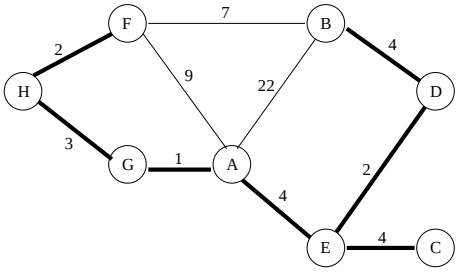
Vernetzte Systeme
Lösungsvorschlag 7

1 Algorithmus von Dijkstra

a)

Step	visited	Set of blue nodes B (with distance, predecessor)
0	A	G(1,A), E(4,A), F(9,A) , B(22,A)
1	A, G(1,A)	H(4,G), E(4,A), F(9,A) , B(22,A)
2	AG, H(4,G)	E(4,A), F(6,H) , B(22,A)
3	AGH, E(4,A)	D(6,E), F(6,H), C(8,E), B(22,A)
4	AGHE, D(6,E)	F(6,H), C(8,E), B(10,D)
5	AGHED, F(6,H)	C(8,E), B(10,D)
6	AGHEDF, C(8,E)	B(10,D)
7	AGHEDFC, B(10,D)	-

b) Spannbaum und Routingtabelle können bestimmt werden, indem die ermittelten Werte aus a) rückwärts bis Knoten A verfolgt werden. Dann ergibt sich für Eintrag B z.B. der Pfad `B -> D -> E -> A`, wobei Nachrichten an B in diesem Fall nach E geschickt werden.



Destination	Outgoing link to use, cost
B	E, 10
C	E, 8
D	E, 6
E	E, 4
F	G, 6
G	G, 1
H	G, 4

2 Distance Vector Routing

a) Die gesuchte Tabelle kann aus den vorgegebenen Werten und den Kosten von A zu den benachbarten Knoten ermittelt werden. Zum Beispiel ergibt sich der Wert für die Strecke von A über B nach G aus dem **niedrigsten** Wert für die Strecke von B nach G (hier 11) zuzüglich den Kosten von A nach B (hier 22), addiert also 33.

		Cost to destination via			
	$D^A()$	B	E	F	G
Destination	B	22	**10**	16	12
	C	32	**8**	23	10
	D	26	**6**	20	8
	E	28	**4**	19	6
	F	29	14	9	**6**
	G	33	9	14	**1**
	H	31	12	11	**4**

b) Ausgangspunkt ist die Änderung der Kosten für die Verbindung von B nach D auf den Wert 6. Dadurch ändern sich in der jeweiligen *Entfernungstabelle* für B alle Werte in der D-Spalte und für D alle Werte in der B-Spalte. Destinationen, für die sich der niedrigste Wert ändert, müssen mittels Nachricht an alle Nachbarn bekannt gemacht werden. Erhält ein Knoten eine solche Nachricht, so muss dieser seine eigene Tabelle aktualisieren. Ändert sich dabei wiederum einer der bisher niedrigsten Werte für einen Zielknoten, so muss auch hier eine Nachricht generiert werden. Entsprechend werden *wichtige* Änderungen im gesamten Netz propagiert.

From	To	Message
B	A, D, F	(A,12), (C,12), (D,6), (E,8), (G,12)
D	B, E	(B,6), (F,12)
F	A, B, H	(D,12)
E	A, C, D	(B,8)
A	B, E, F, G	(B,12)
C	E	(B,12)
G	A, H	(B,12)

c) Die vollständige Routingtabelle lässt sich leicht aus den Entfernungstabellen bestimmen: Die Einträge entsprechen den jeweils niedrigsten Werten zu einer Destination. **Fett** dargestellt sind die Werte, die sich gegenüber der ursprünglichen Routingtabelle geändert haben.

		Routing (outgoing link to use, cost)							
	Source →	A	B	C	D	E	F	G	H
Destination	A	--	**D, 12**	E, 8	E, 6	A, 4	H, 6	A, 1	G, 4
	B	**E, 12**	--	**E, 12**	**B, 6**	**D, 8**	B, 7	**H, 12**	F, 9
	C	E, 8	**D, 12**	--	E, 6	C, 4	H, 14	A, 9	G, 12
	D	E, 6	**D, 6**	E, 6	--	D, 2	**H, 12**	A, 7	G, 10
	E	E, 4	**D, 8**	E, 4	E, 2	--	H, 10	A, 5	G, 8
	F	G, 6	F, 7	E, 14	**E, 12**	A, 10	--	H, 5	F, 2
	G	G, 1	**F, 12**	E, 9	E, 7	A, 5	H, 5	--	G, 3
	H	G, 4	F, 9	E, 12	E, 10	A, 8	H, 2	H, 3	--

3 Count-to-Infinity Solution?

a) Die folgenden Entfernungstabellen zeigen die Ausgangslage vor dem Unterbruch der Verbindung zwischen b und c. Die neu gegebene Information über den ersten Knoten auf dem Weg zum Ziel ist dabei ebenfalls eingetragen. Aus der Entfernungstabelle für b können wir also

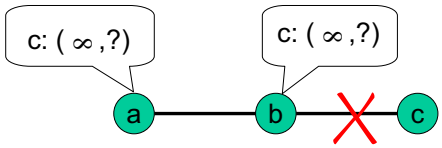
zum Beispiel entnehmen, dass es von b zu c via a einen Pfad mit Kosten 3 gibt. Dieser führt via a und zurück zu b (als erstem Knoten auf dem Weg von a zu c) zu c.

$D^a()$	b
b	**(1,b)**
c	**(2,c)**

$D^b()$	a	c
a	**(1,a)**	(3,b)
c	(3,b)	**(1,c)**

$D^c()$	b
a	**(2,a)**
b	**(1,b)**

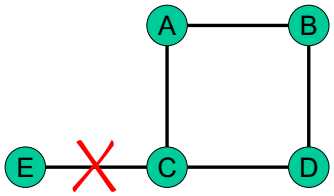
Nun wird die Verbindung zwischen b und c unterbrochen. Der Knoten b merkt dies und kann somit in seiner Entfernungstabelle die Spalte 'via c' komplett löschen. Als Alternativverbindung zu c bleibt ein Pfad via a, denn laut Eintrag gibt es einen Pfad von b nach c via a. Aber a würde als ersten Knoten auf diesem Pfad wiederum b benutzen. b hat aber keine Verbindung mehr zu c. Also kann b daraus schliessen, dass zur Zeit[1] kein Pfad zu c bekannt ist und die Nachricht **c:(∞,?)** propagieren.



Der Knoten a empfängt diese Nachricht. Sein einziger Pfad zu c ging via b. Da b jetzt keinen Pfad mehr zu c kennt, kennt a auch keinen Pfad mehr zu c. Dies wird festgehalten und propagiert.

Für dieses Beispiel kann also mit der vorgeschlagenen Verbesserung das Count-to-Infinity-Problem verhindert werden.

b) Das Count-to-Infinity-Problem lässt sich durch die vorgeschlagene Verbesserung aber nicht immer verhindern. Anhand des folgenden Beispiels lässt sich zeigen, dass das Problem weiterhin besteht.



Die Entfernungstabellen des Distance-Vector-Routing Algorithmus sehen zu Beginn wie folgt aus:

Cost to destination via

Destination	$D^A()$	B	C
	B	**(1,B)**	(3,D)
	C	(3,D)	**(1,C)**
	D	**(2,D)**	(2,D)
	E	(4,D)	**(2,E)**

Cost to destination via

Destination	$D^B()$	A	D
	A	**(1,A)**	(3,C)
	C	**(2,C)**	(2,C)
	D	(3,C)	**(1,D)**
	E	**(3,C)**	(3,C)

Destination	$D^C()$	A	D	E
	A	**(1,A)**	(3,B)	(∞,?)
	B	**(2,B)**	(2,B)	(∞,?)
	D	(3,B)	**(1,D)**	(∞,?)
	E	(∞,?)	(∞,?)	**(1,E)**

Destination	$D^D()$	B	C
	A	**(2,A)**	(2,A)
	B	**(1,B)**	(3,A)
	C	(3,A)	**(1,C)**
	E	(4,A)	**(2,E)**

Hier ist zu beachten, dass zusätzlich zu den Kosten auch noch der erste Knoten auf dem Weg zur Destination festgehalten wird. Ausserdem wurden die Einträge für Pfade, die im zweiten Schritt wieder über den Ausgangsknoten zurückgeroutet werden, auf (∞,?) gesetzt. Dies ist sinnvoll, da diese Pfade eine unnötige Schleife enthalten und daher ohnehin nicht verwendet werden würden. (∞,?) bedeutet, dass dieser Knoten nicht mehr erreicht werden kann, respektive mit unendlichen Kosten, und der Nachfolger auf dem Pfad ist entsprechend unbekannt.

[1] Es wäre möglich, dass a einen alternativen (schlechteren) Pfad zu c kennt und diesen dann gewissermassen als Antwort zurückliefert. Somit wäre für b lediglich eine gewisse Zeit lang kein Pfad zu c bekannt. Dies ist jedoch in unserem einfachen Beispiel nicht der Fall.

Sobald der Knoten C den Unterbruch der Verbindung C-E feststellt, teilt er dies seinen Nachbarn A und D mit der Nachricht (E,∞,?) mit. Die weiteren erzeugten Nachrichten sind in der nachfolgenden Tabelle aufgeführt.

From	To	Message
C	A, D	(E, ∞, ?)
A	B, C	(E, 4, B)
D	B, C	(E, 4, B)
B	A, D	(E, 3, D)
C	A, D	(E, 5, A)
B	A, D	(E, ∞, ?)
C	-	-
A	-	-
D	B, C	(E, ∞, ?)
A	-	-
D	B, C	(E, 6, C)
A	B, C	(E, ∞, ?)
D	-	-
B	-	-
C	-	-
B	A, D	(E, 7, D)
C	-	-
B	-	-
C	A, D	(E, ∞, ?)
A	B, C	(E, 8, B)
D	B, C	-
A	-	-
D	B, C	(E, ∞, ?)
B	-	-
C	A, D	(E, 9, A)
...	...	...

Wenn man den Ablauf durchspielt, erkennt man, wie die besten Pfade von A, B, C und D nach E iterativ um 4 inkrementiert werden, respektive dazwischen jeweils eine gewisse Zeit auf ∞ gesetzt werden. Somit versagt für dieses Beispiel die vorgeschlagene Verbesserung.

Intuitiv lässt sich das Scheitern an diesem Beispiel folgendermassen erklären: Nachdem C festgestellt hat, dass die direkte Verbindung zu E abgebrochen ist, hört C die Nachricht, dass E via A und B respektive via D und B in 5 Schritten erreichbar ist. Die von C in der Entfernungstabelle gehaltene Information reicht jedoch nicht aus, um zu erkennen, dass diese Pfade später auch wieder über C laufen. Somit wird diese Information aufgenommen und weiter propagiert, obwohl diese Pfade unsinnig sind und gar nicht existieren. Aber so beginnen sich die Knoten gegenseitig hochzuzählen und Count-to-Infinity nimmt seinen Lauf.

Mit der gegebenen Verbesserung kann der Algorithmus in jedem Knoten zwar besser 'voraussehen' und Probleme erkennen, er sieht nämlich von allen Pfaden jeweils die ersten beiden Knoten. Aber wenn genügend lange Zyklen vorkommen, reicht auch das nicht und Count-to-Infinity tritt auf.