

Kapitel 6: Dynamic Shortest Path

6.3 APSP von Demetrescu & Italiano, TEIL 2

VO Algorithm Engineering

Professor Dr. Petra Mutzel

Lehrstuhl für Algorithm Engineering, LS11

15./16./17. VO 29./31. Mai / 5. Juni 2007

Literatur für diese VO

- C. Demetrescu und G.F. Italiano: A new approach to dynamic all pairs shortest paths, Proc. 35th Annual ACM Symposium on Theory of Computing (STOC '03), San Diego, 2003, 159-166.
- Zeitschriftenversion: Journal of the Association for Computing Machinery (JACM), vol. 51 (6), 2004, 968-992.

Überblick

Eigenschaften von LSPs

Algorithmus Increase-Only

Analyse: Korrektheit + Laufzeit

Probleme im voll-dynamischen Fall

Eigenschaften von LHPs

Algorithmus Fully-Dynamic

Analyse: Korrektheit + Laufzeit



Historisches

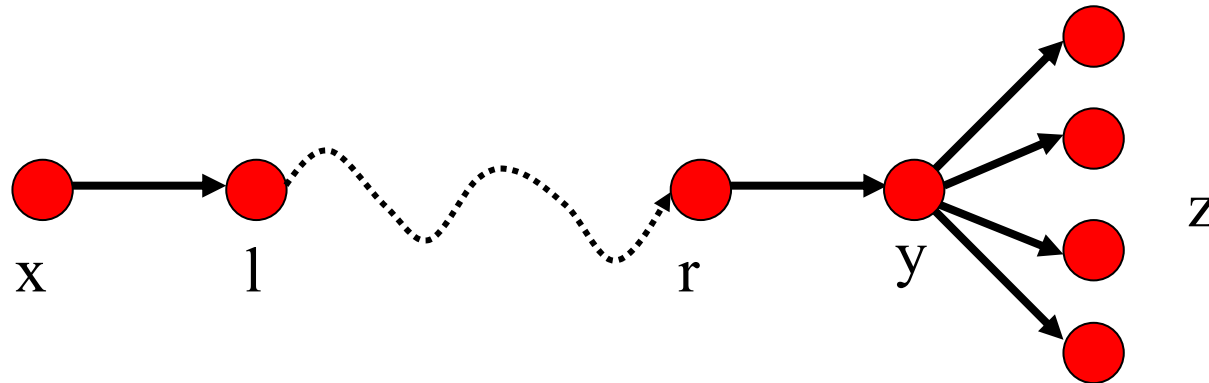
- Seit 1967 studiert:
 - „Highway Research“, „Transport Network Theory“
- **King 1999:**
 - erster voll-dynamischer Algorithmus, der im worst case schneller als der statische Algorithmus war;
 - allerdings nur für ganzzahlige $\text{Kosten} \leq C$;
 - Laufzeit: $O(n^{2.5}(C \log n)^{0.5})$ per update & optimal per Query
- **Demetrescu und Italiano 2003:**
 - erster voll-dynamischer Algorithmus ohne Einschränkungen, der im worst case schneller als der statische Algorithmus ist
 - Laufzeit: $O(n^2 \log^3 n)$ amort. per update & optimal (look-up) per Query

Idee: mittels LSP

Dynamische APSP

- Geg.: Gerichteter Graph $G=(V,E)$ mit nicht-negativen Kantenkosten $w(e) \in \mathbb{R}$, sowie eine Sequenz der folgenden Operationen:
 - **update(v,w')**: ändere die Kosten aller zu Knoten v inzidenten Kanten zur neuen Kostenfunktion w' (verallgemeinerte Version von dyn. APSP)
 - **distance(x,y)**: gib die Distanz zwischen Knoten x und y zurück
 - **path(x,y)**: gib den kürzesten Weg von x nach y aus, falls einer existiert.

Locally Shortest Paths



Nach einem $\text{update}(v, w')$ sind

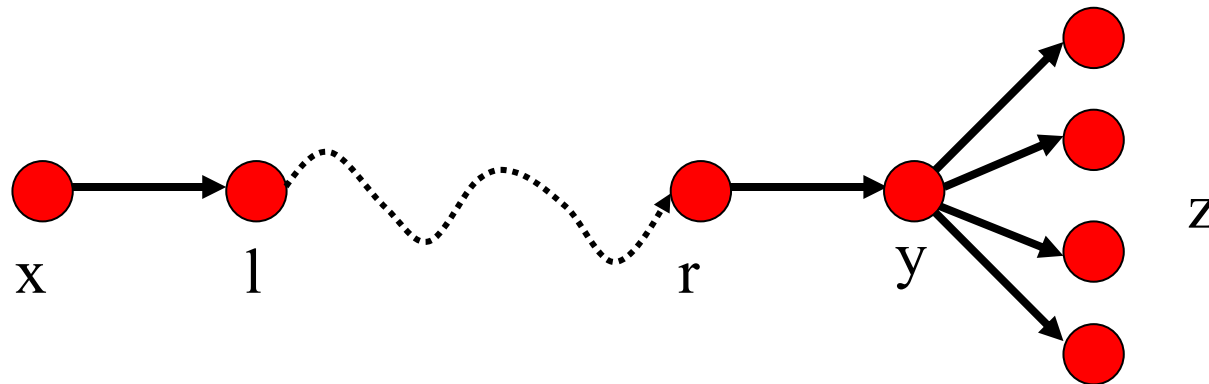
- einige kürzeste Wege vor dem Update nicht mehr die kürzesten nach dem Update
- andere Wege, die vorher nicht kürzeste Wege waren, werden jetzt die kürzesten

- **ZIEL:** Finde die neuen kürzesten Wege
- Natürliche Kandidaten hierfür: LSP

Algorithmus-Idee

- Halte alle LSPs in einer Datenstruktur DS, so dass die Ersatzpfade schnell gefunden werden können.
- Frage: Wie teuer ist die Verwaltung einer solchen DS?
 - Wieviele LSPs können per Update neu dazu kommen?
 - Wieviele LSPs können per Update herausfallen?

Definitionen



- Notation für einen Weg: $P(x \rightarrow z)$
- $l(P(x \rightarrow z))$: „linker“ Teilweg, d.h. $P(x \rightarrow z)$ ohne Kante (y, z)
- $r(P(x \rightarrow z))$: „rechter“ Teilweg, d.h. $P(x \rightarrow z)$ ohne Kante (x, l)
- LSP: Locally shortest path
- SP: Shortest path

Eigenschaften von LSPs

- **Beobachtung 1:** Falls $l(P(x \rightarrow y))$ und $r(P(x \rightarrow y))$ SP sind, dann ist $P(x \rightarrow y)$ ein LSP

- **Beobachtung 2:** Bezeichnen \mathcal{SP} bzw. \mathcal{LSP} die Menge aller SPs bzw. LSPs in G . Dann ist $\mathcal{SP} \subseteq \mathcal{LSP}$.

- **Lemma 2:** Falls SP eindeutig in G sind, dann: Für alle Knotenpaare (x, y) gilt: die LSP-Wege von x nach y sind knotendisjunkt (außer Anfangs- und Endknoten x, y).

- Beweis Indirekt: Annahme: Es existieren 2 LSP Pfade P^1 und P^2 durch den gleichen Knoten $v \neq x, y$.
- Da jeder echte Teilweg von $P(x, y)$ ist SP, also auch $P^1(x \rightarrow v)$ und $P^2(x \rightarrow v)$
- wegen Eindeutigkeit: $P^1(x \rightarrow v) = P^2(x \rightarrow v)$.

Eigenschaften von LSPs

- **Lemma 3:** Falls SP eindeutig in G sind, dann kann es höchstens mn LSPs in G geben.

- **Lemma 4:** Sei G Graph, der eine Sequenz Σ von Knoten Updates durchlaufen hat. Falls SP eindeutig in G sind, dann können anlässlich eines „increase“ updates höchstens $O(n^2)$ Wege aus der LSP Menge herausfallen („stop being LSP“)

- Beweis: Ein Weg kann aus LSP herausfallen, nur wenn durch den Update einer seiner Teilwege nicht mehr in SP ist.
- Dies kann nur passieren, wenn dieser Teilweg den Knoten v enthält.
- Es gibt aber höchstens $O(n^2)$ LSPs, die v als internen Knoten enthalten und höchstens $O(n^2)$ LSPs mit v als Endknoten.

Eigenschaften von LSP's

- **Theorem 1:** Sei G Graph mit einer Sequenz Σ von increase-only update Operationen und sei m größte Kantenanzahl in G unter Σ . Falls SP eindeutig sind, dann ist die Anzahl der Pfade, die durch eine increase-Operation neue LSPs werden:
 - $O(mn)$ im Worst Case
 - $O(n^2)$ amortisiert über $\Omega(m/n)$ Update Operationen.
- Beweis: $O(mn)$ gilt, weil $O(mn)$ die maximale Anzahl in G ist.
- Amortisierte Analyse (Sketch): zu jedem Zeitpunkt können zwar theoretisch $O(mn)$ LSPs neu entstehen; andererseits können höchstens $O(n^2)$ LSPs herausfallen; insgesamt können zu jedem Zeitpunkt höchstens $O(mn)$ LSPs existieren. Nach m/n Operationen können also höchstens per Operation $O(mn/(m/n))=O(n^2)$ LSPs neu entstanden sein.

Algorithmus Idee für Increase-Only

- Halte alle LSPs $P(x \rightarrow y)$ in Datenstruktur $DS(x, y)$
- Wg. Theorem 1 hat man $O(n^2)$ Änderungen per Update in DS bei $\Omega(m/n)$ Operationen
- Als DS wähle Prioritätsschlange; Priorität= $\text{dist}()$
- $O(\log n)$ Kosten pro Weg-Änderung
- Insgesamt Laufzeit: $O(n^2 \log n)$ amortisiert ($\Omega(m/n)$ Oper.)

Speicherung der Wege:

- $P(x \rightarrow y)$ Zeiger zu $l(x, y)$ und $r(x, y)$ (konstant viel Speicherplatz)
- Das geht, wegen
 - Eindeutigkeit von SP und
 - optimaler Teilweg Eigenschaften

Datenstrukturen

- $\text{dist}[x,y]$: Matrix: Achtung: nicht mehr notwendig, da: $w(P())$
- $w(P(x \rightarrow y))$: Kosten des Weges $P(x \rightarrow y)$ // bei SP-Wegen = $\text{dist}[]$
- $P(x,y)$: Menge aller LSPs von x nach y (in PrioQ)
- $P^*(x,y)$: Menge aller SPs von x nach y (in PrioQ)
- $L(P(x \rightarrow y))$: Liste der möglichen $P(x \rightarrow y)$ path extension Wege nach vorn (*left*), die LSP sind
- $L^*(P(x \rightarrow y))$: Liste der möglichen $P(x \rightarrow y)$ path extension Wege nach vorn (*left*), die SP sind
- $R(P(x \rightarrow y))$: Liste der möglichen $P(x \rightarrow y)$ path extension Wege nach hinten (*right*), die LSP sind
- $R^*(P(x \rightarrow y))$: Liste der möglichen $P(x \rightarrow y)$ path extension Wege nach hinten (*right*), die SP sind

Achtung: L und R Listen gehören hier zu den Wegen (nicht zu Knotenpaaren)

Algorithmus Update()

Algorithmus Update(v, w'):

1. Cleanup(v): entfernt alle Wege, die v enthalten
2. Fixup(v, w'): Addiere alle neuen SP's und LSP's

Def.: Nach einer Operation Update(v, w') heißt ein SP (bzw. LSP) **NEU**, wenn er vorher nicht SP (bzw. LSP) war oder er den Knoten v enthält.

Prozedur Cleanup(v)

Prozedur Cleanup(v):

Entfernt alle Wege aus DS, die in $G \setminus \{v\}$ nicht mehr LSP wären

=== Entfernt alle Wege $P(x \rightarrow y)$, die v enthalten aus

$P(x,y), P^*(x,y), L(r(P(x \rightarrow y))), L^*(r(P(x \rightarrow y))), R(l(P(x \rightarrow y))),$
 $R^*(l(P(x \rightarrow y)))$.

Cleanup(v) kann realisiert werden durch:

1. Entferne alle Wege der Form $P(u \rightarrow v)$ und $P(v \rightarrow u)$
2. Entferne rekursiv die dazugehörigen $L()$ und $R()$ -Wege

Prozedur Cleanup(v)

- InsertQ ← {(v)}
- **Solange** $Q \neq \emptyset$:
 - $P \leftarrow \text{ExtractElem}(Q)$
 - **Für** jeden Weg $P(x \rightarrow y) \in L(P) \cup R(P)$
 - InsertQ ← $P(x \rightarrow y)$
 - Entferne $P(x \rightarrow y)$ aus $P(x, y)$, $L(r(P(x \rightarrow y)))$ und $R(l(P(x \rightarrow y)))$
 - **Falls** $P(x \rightarrow y) \in P^*(x, y)$, **dann**
 - Entferne $P(x \rightarrow y)$ aus $P^*(x, y)$, $L^*(r(P(x \rightarrow y)))$ und $R^*(l(P(x \rightarrow y)))$.

Prozedur Fixup(v, w'): Phase 1

Für alle $u \neq v$:

- $w(u, v) \leftarrow w'(u, v); w(v, u) \leftarrow w'(v, u)$
- **Falls** $w(u, v) < M$ **dann:**
 - setze $w(P(u \rightarrow v)), l(P(u \rightarrow v)) \leftarrow \{u\}, r(P(u \rightarrow v)) \leftarrow \{v\}$
 - addiere $P(u \rightarrow v) = \langle u, v \rangle$ zu $P(u, v), L(P(v)), R(P(u))$
- **Falls** $w(v, u) < M$ **dann:**
 - setze $w(P(v \rightarrow u)), l(P(v \rightarrow u)) \leftarrow \{v\}, r(P(v \rightarrow u)) \leftarrow \{u\}$
 - addiere $P(v, u) = \langle v, u \rangle$ zu $P(v \rightarrow u), L(P(u)), R(P(v))$

Alle neuen LSP's der Länge 1 werden zu DS hinzuaddiert

Prozedur Fixup(v, w'): Phase 2

$H \leftarrow \emptyset$

Für jedes Knotenpaar (x, y) :

- addiere den Weg $P(x \rightarrow y) \in P(x, y)$ mit den kleinsten Kosten zu H

Prozedur Fixup(v, w'): Phase 3

Solange $H \neq \emptyset$: ExtractMinH \leftarrow P($x \rightarrow y$)

Falls P($x \rightarrow y$) der erste extrahierte Weg für das Paar (x, y) ist:

Falls P($x \rightarrow y$) \notin P*(x, y)

– Addiere P($x \rightarrow y$) zu P*(x, y), $L^*(r(P(x \rightarrow y)))$ und $R^*(l(P(x \rightarrow y)))$

– **Für** jeden Weg P($x' \rightarrow b$) $\in L^*(l(P(x \rightarrow y)))$ //kombiniere alle LSPs

• P($x' \rightarrow y$) \leftarrow (x', x)P($x \rightarrow y$) // bilde neuen LSP

• $w(P(x' \rightarrow y)) \leftarrow w(x', x) + w(P(x \rightarrow y))$ //setze Kosten

• $l(P(x' \rightarrow y)) \leftarrow P(x' \rightarrow b)$; $r(P(x' \rightarrow y)) \leftarrow P(x \rightarrow y)$

• Addiere P($x' \rightarrow y$) zu P(x', y), L(P($x \rightarrow y$)), R(P($x' \rightarrow b$)) und H

– **Für** jeden Weg P($a \rightarrow y'$) $\in R^*(r(P(x \rightarrow y)))$

• P($x \rightarrow y'$) \leftarrow P($x \rightarrow y$)(y, y')

• $w(P(x \rightarrow y')) \leftarrow w(P(x \rightarrow y)) + w(y, y')$

• $l(P(x \rightarrow y')) \leftarrow P(x \rightarrow y)$; $r(P(x \rightarrow y')) \leftarrow P(a \rightarrow y')$;

• Addiere P($x \rightarrow y'$) zu P(x, y'), L(P($a \rightarrow y'$)), R(P($x \rightarrow y$)) und H

Bemerkungen

- Achtung: hier wird bei „edge scanning“ jeder neu entdeckte Weg in H eingefügt, nicht wie bisher: update

Analyse Korrektheit

- **Invariante:** Falls die kürzesten Wege eindeutig sind, dann ist für jedes Knotenpaar x und y in G der erste (x,y) -Weg, der aus H in Phase 3 von `Fixup()` extrahiert wird ein kürzester Weg.
- **Annahme:** Invariante ist (erstes Mal) verletzt bei Pfad $P'(x \rightarrow y)$. Sei $P(x \rightarrow y)$ SP zwischen x und y .
- Offensichtlich $P(x \rightarrow y) \notin H$ und $\notin P(x,y)$ (sonst Phase 2 in H)
- Also ist $P(x \rightarrow y)$ ein neuer LSP mit Länge >1 (wg. Phase 1)
- $l(P(x \rightarrow y))$ und $r(P(x \rightarrow y))$ müssen neue kürzeste Wege sein; einer davon war zu Beginn von `fixup()` nicht in P^*
- $l(P(x \rightarrow y))$ oder $r(P(x \rightarrow y))$ wurden vor der falschen Extraktion (s.o.) aus H extrahiert (weil geringere Kosten)
- dort wurde dann auch $P(x \rightarrow y)$ gebildet und zu H addiert



Analyse Korrektheit

- Falls die Operation ein **Increase** war und die kürzesten Wege eindeutig sind, dann werden die Mengen $P(x,y)$ und $P^*(x,y)$ für jedes Knotenpaar x und y korrekt aktualisiert.
- Cleanup(): o.k.
- Wird jeder neue LSP gebildet? Länge 1: o.k. Länge >1 :
- $P(x \rightarrow y)$ muss neu gebildet werden, wenn beide, $l(P(x \rightarrow y))$ und $r(P(x \rightarrow y))$ SP sind aber mindestens einer davon nicht im alten P^* war.
- Irgendwann wird (o.E.) $l(P(x \rightarrow y))$ extrahiert; dann wird $l(P(x \rightarrow y))$ kombiniert mit $r(P(x \rightarrow y)) \in R^*(r(l(P(x \rightarrow y))))$ um $P(x \rightarrow y)$ zu bilden und zu H zu addieren.

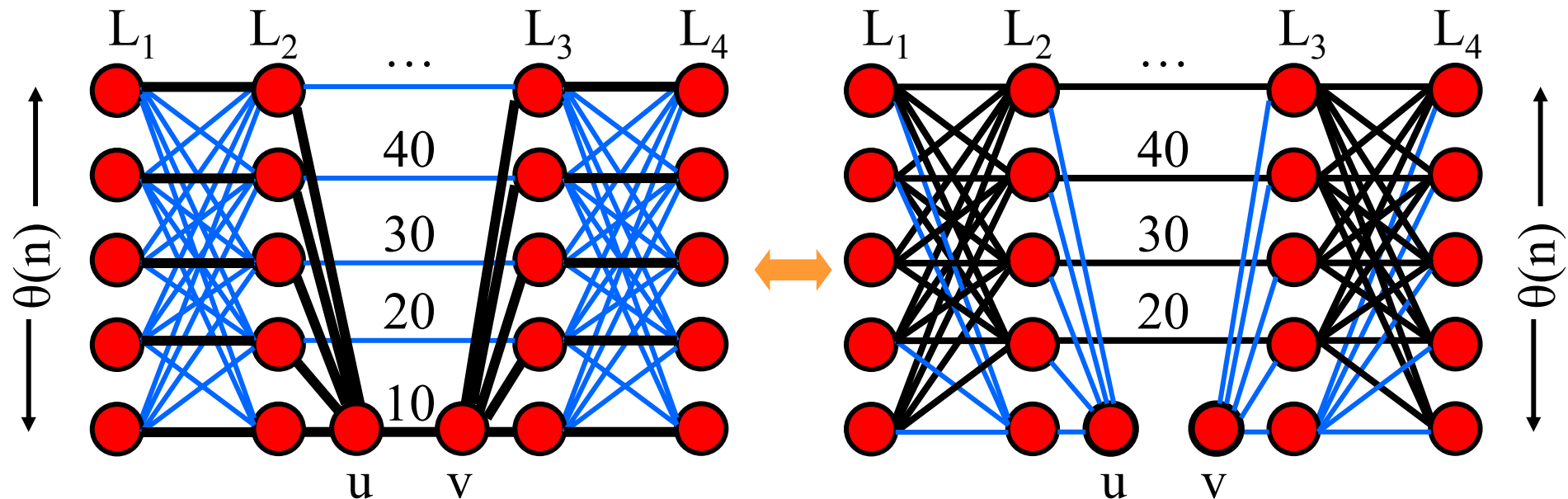
Analyse Laufzeit

- In einer **Increase-only** Sequenz von $\Omega(m/n)$ Operationen wird jede Update Operation in $O(n^2 \log n)$ amortisierter Zeit, und jede Distanz und Weg-Anfrage in optimaler Zeit durchgeführt.
- Da $P(x,y)$ für alle Paare in eigener Priority Queue gehalten wird: Abfrage nach $\text{dist}(x,y)$ in $O(1)$
- Cleanup(v): maximal $O(n^2)$ LSPs können durch v laufen; pro Iteration $O(\log n)$: $O(n^2 \log n)$
- Fixup():
 - Phase 1: $O(n \log n)$
 - Phase 2: $O(n^2 \log n)$
 - Phase 3: $O(n^2 \log n)$

Decrease-only APSP

- **Beobachtung:** Algorithmus **Increase-only APSP** funktioniert auch für Decrease-only Sequenzen mit gleicher Laufzeit $O(n^2 \log n)$, denn
 - ein neuer LSP wg. Decrease in Vorwärtsrichtung entspricht einem Herausfallen eines LSP Weges bei Increase in Rückwärtsrichtung (rückwärts Durchlaufen der Sequenz)
-
- **Decrease-only** ist jedoch einfacher lösbar, denn:
 - alle neuen SPs nach $\text{update}(v, w')$ müssen über v laufen
 - löse ein SSSP von v zu allen anderen Knoten und
 - ein SSSP zu v von allen Knoten
 - Falls für ein Knotenpaar die verschmolzenen Teilwege kürzer sind als die vorherigen Wege vor Änderung, dann: NEUER Weg gefunden!

Probleme bei fully-dynamic



Kanten sind von links nach rechts gerichtet und besitzen kleine Kantengewichte $\epsilon > 0$ so dass SP eindeutig und wie in Abb.

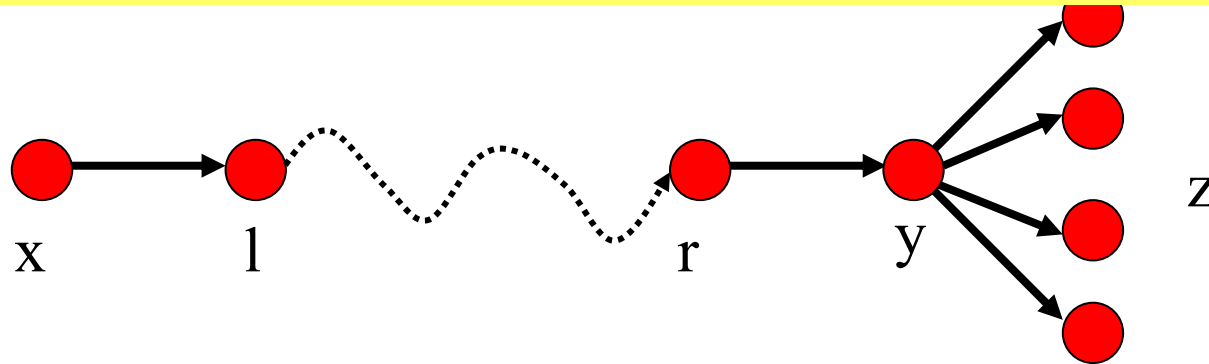
Update-Sequenz: Iteratives Einfügen und Entfernen von (u,v)
Betrachte: LSPs zwischen Knoten in L_1 und L_4

→ führt zu $\theta(n^3)$ Änderungen in der Menge \mathcal{LSP}

→ Aktualisierung von \mathcal{LSP} nach jedem Update zu teuer

Locally Historical Paths

Idee: Aktualisierung von \mathcal{LSP} nicht nach jedem Update



Betrachte vertex update auf v zur Zeit t' :

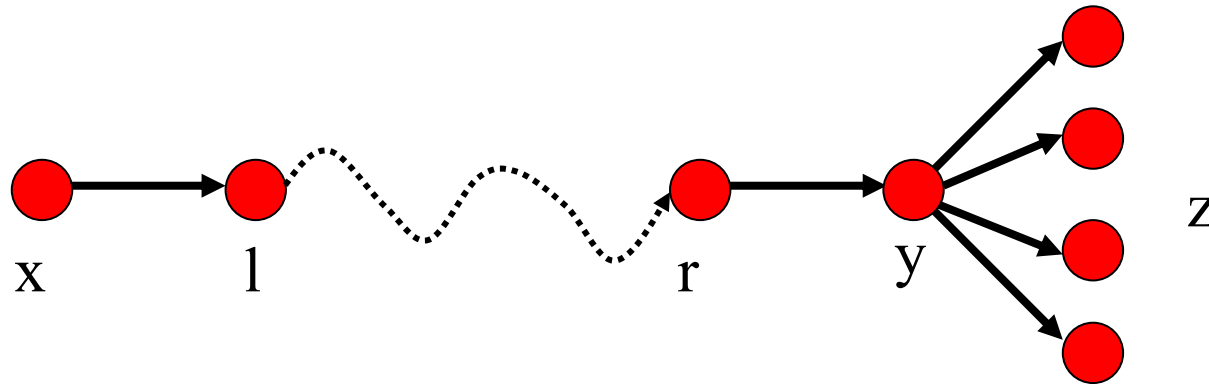
- Laut Algorithmus **Increase-only APSP** werden in `Cleanup()` alle Wege aus DS entfernt, die v enthalten. In `Fixup()` werden diese wieder neu (korrekt) eingefügt.
- Betrachte nun einen neuen SP-Weg $P(x \rightarrow y)$

Betrachte nun die Operation `Decrease(u)` von $u \notin P(x \rightarrow y)$ zum Zeitpunkt $t > t'$.

- Evtl. ist $P(x \rightarrow y)$ nun kein SP-Weg mehr

Locally Historical Paths

Idee: Aktualisierung von \mathcal{LSP} nicht nach jedem Update

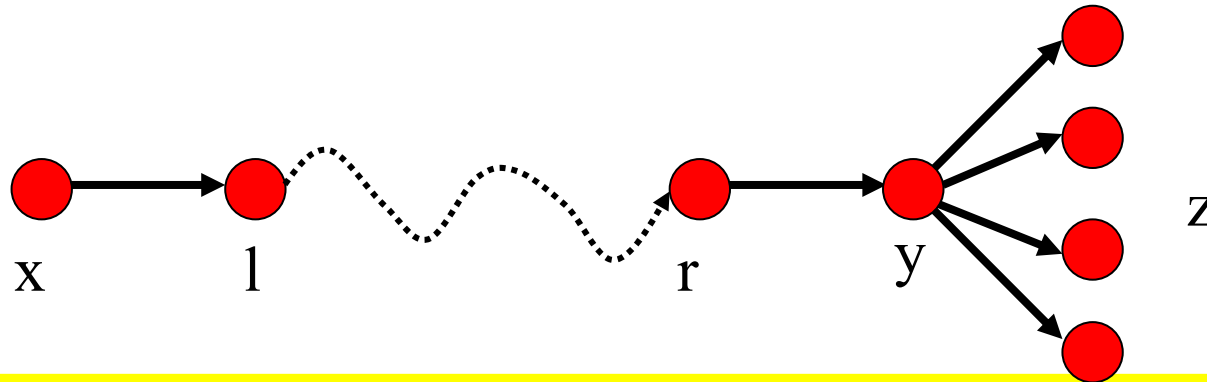


Sei $P(x \rightarrow y)$ ein Weg in G zur Zeit t , und sei $t' \leq t$ der Zeitpunkt des letzten vertex updates auf Knoten in $P(x \rightarrow y)$.

Wir bezeichnen $P(x \rightarrow y)$ als **historisch (historical)** zur Zeit t , wenn er im Zeitintervall $[t', t]$ mindestens einmal kürzester Weg gewesen ist.

ENDE

Locally Historical Paths



- Ein Weg $P(x \rightarrow z)$ heißt „Lokal Historischer Weg“ („Locally Historical Path“, LHP) in G zur Zeit t , falls entweder
 - $P(x \rightarrow z)$ aus einem einzigen Knoten besteht, oder
 - jeder echte Teilweg von $P(x \rightarrow z)$ ein historischer Weg in G zur Zeit t ist.
- Achtung: LHP Wege müssen nicht unbedingt irgendwann LSP Wege gewesen sein.
- Denn: Im Gegensatz zu LSP, müssen die Teilwege nicht zur gleichen Zeit kürzeste Wege sein.

Eigenschaften von LHPs (1)

Lemma 1: Bezeichnen SP , \mathcal{HP} bzw. \mathcal{LHP} die Menge aller SPs, HPs bzw. LHPs in G zur Zeit t in einer Sequenz von Updates. Dann gilt: $SP \subseteq \mathcal{HP} \subseteq \mathcal{LHP}$.

Beweis: $SP \subseteq \mathcal{HP}$: aus Definition.

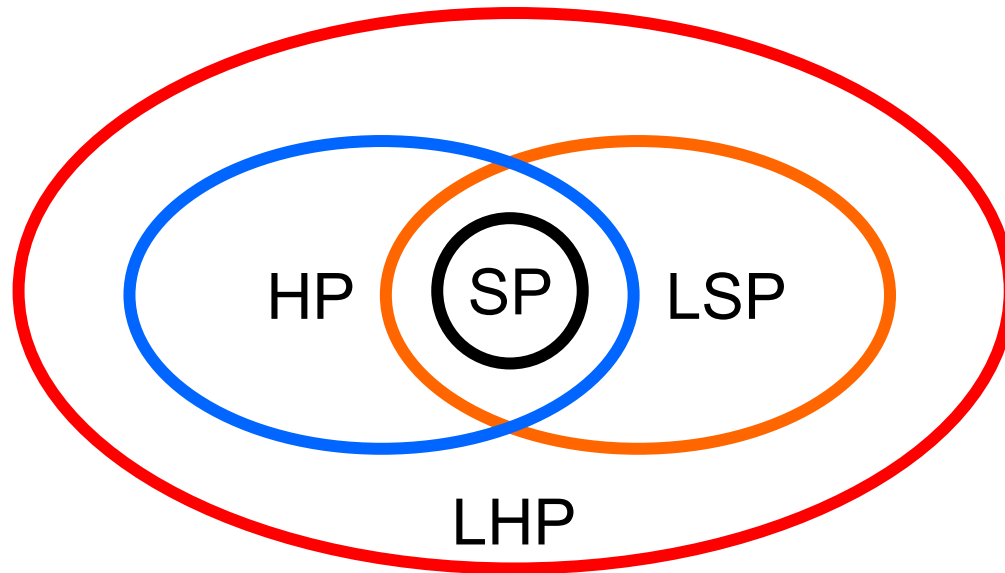
$\mathcal{HP} \subseteq \mathcal{LHP}$: Sei P ein HP zur Zeit t und sei $t' \leq t$ der Zeitpunkt des letzten vertex updates auf P . Für jeden echten Teilweg $P^i \subset P$, sei $t^i \leq t' \leq t$ der Zeitpunkt des letzten vertex updates von P^i .

Laut Def. war P mind. einmal kürzester Weg im Intervall $[t', t]$.

Wegen der optimalen Teilweg Eigenschaft war P^i auch mind. einmal kürzester Weg im Intervall $[t^i, t] \supseteq [t', t]$, also historisch zur Zeit t . Daraus folgt $\mathcal{HP} \subseteq \mathcal{LHP}$.

Eigenschaften von LHPs (1)

Lemma: Bezeichnen SP , HP bzw. LHP die Menge aller SPs, HPs bzw. LSPs in G zur Zeit t in einer Sequenz von Updates. Dann gilt: $SP \subseteq HP \subseteq LHP$.



Eigenschaften von LHPs (2)

Lemma 2: Sei G ein Graph mit Update-Sequenz Σ . Wenn zu jeder Zeit t höchstens z HPs zwischen jedem Knotenpaar existieren und $G(t)$ m Kanten besitzt, dann können zur Zeit t höchstens zmn LHPs existieren.

Beweis: Fixiere Kante (x,u) und einen Knoten y . Da jeder echte Teilweg eines LHPs historisch sein muss, existieren höchstens z historische Wege zwischen u und y .

Es gibt m Möglichkeiten zur Wahl der ersten Kante und n Möglichkeiten zur Wahl des letzten Knotens.

Eigenschaften von LHPs (3)

Lemma 3: Sei G Graph mit einer Update-Sequenz Σ und seien x, y zwei Knoten in G und sei $\sigma(P)$ der letzte Knoten-Update auf Weg P und $v(\sigma)$ der dazugehörige Knoten des Updates σ .

Falls SP eindeutig sind, gilt für je zwei verschiedene historische Wege $P=P(x \rightarrow y)$ und $P'=P'(x \rightarrow y)$ in G zu jeder Zeit t : $v(\sigma(P)) \neq v(\sigma(P'))$.

- Beweis Indirekt: Annahme: Es gilt Gleichheit.
- Beide Wege waren mind. einmal SP im Intervall $[t(\sigma(P)), t]$.
- Allerdings wird keiner im Intervall $(t(\sigma(P)), t]$ berührt
- Widerspruch zu: Eindeutigkeit von SP.

Eigenschaften von LHPs (4)

Lemma 4: Sei G Graph mit einer Update-Sequenz Σ und sei v ein Knoten in G . Falls SP eindeutig sind und es zu jeder Zeit höchstens z HPs zwischen jedem Knotenpaar gibt, dann:

- gibt es höchstens $O(zn^2)$ LHPs mit v als Endknoten.

Beweis: Betrachte Wege der Form (v, x, \dots, y) : Es gibt $O(n^2)$ Möglichkeiten für x und y ; es gibt $\leq z$ verschiedene $(x \rightarrow y)$ HP Wege.


Eigenschaften von LHPs (4)

Lemma 5: Sei G Graph mit einer Update-Sequenz Σ und sei v ein Knoten in G . Falls SP eindeutig sind und es zu jeder Zeit höchstens z HPs zwischen jedem Knotenpaar gibt, dann:

- gibt es höchstens $2z$ LHPs mit v als internen Knoten.

Beweis: Sei \mathcal{P} Menge der LHP Wege der Form (x, \dots, v, \dots, y) und $\mathcal{Q} \subseteq \mathcal{P}$ der Form $Q_i = (x, \dots, v_i, \dots, v, \dots, y)$, wobei v_i ist der letzte update vertex von Q_i (außer x, y).

Betrachte \mathcal{Q}' : Menge der Teilwege der Form $(x, \dots, v_i, \dots, v)$ in \mathcal{Q} . Beh.: $|\mathcal{Q}| = |\mathcal{Q}'|$. **NEU:**

Denn z.z.: je 2 versch. Wege in \mathcal{Q} sind auch verschieden in \mathcal{Q}' :
Sonst (falls „=“): letzter update vertex v_i ist gleich. Betrachte Teilwege der Form $(v_i, \dots, v, \dots, y)$: diese sind HP und besitzen gleichen letzten update vertex (v_i oder y). Wg. Lemma 3: „=“ 

Eigenschaften von LHPs (4)

Lemma 5: Sei G Graph mit einer Update-Sequenz Σ und sei v ein Knoten in G . Falls SP eindeutig sind und es zu jeder Zeit höchstens z HPs zwischen jedem Knotenpaar gibt, dann:

- gibt es höchstens $2z$ LHPs mit v als internen Knoten.

Beweis: Sei \mathcal{P} Menge der LHP Wege der Form (x, \dots, v, \dots, y) und $\mathcal{Q} \subseteq \mathcal{P}$ der Form $Q_i = (x, \dots, v_i, \dots, v, \dots, y)$, wobei v_i ist der letzte update vertex von Q_i (außer x, y).

Betrachte \mathcal{Q}' : Menge der Teilwege der Form $(x, \dots, v_i, \dots, v)$ in \mathcal{Q} . Beh.: $|\mathcal{Q}| = |\mathcal{Q}'|$. (gezeigt)

Ind. Annahme: $|\mathcal{P}| > 2z$, dann o.E. $|\mathcal{Q}| \geq |\mathcal{P}|/2$, also $|\mathcal{Q}| > z$.

Alle Wege in Menge \mathcal{Q}' sind historisch zur Zeit t (s. Def. \mathcal{P}).

Weil aber $|\mathcal{Q}| = |\mathcal{Q}'|$ gibt es zur Zeit t mehr als z HPs zwischen x und v in G . Widerspruch.

Eigenschaften von LHPs (5)

Lemma 6: Sei G Graph mit einer Update-Sequenz Σ . Falls SP eindeutig sind und es zu jeder Zeit höchstens z HPs zwischen jedem Knotenpaar gibt, dann ist die Anzahl der Wege, die aus der Menge der LHPs per Update herausfallen höchstens $O(zn^2)$.

Beweis: Ein Weg P kann nur durch vertex update an einem Knoten in P aus LHP-Menge herausfallen.

Dann folgt die Behauptung direkt aus den Definitionen und Eigenschaften von LHPs (4).

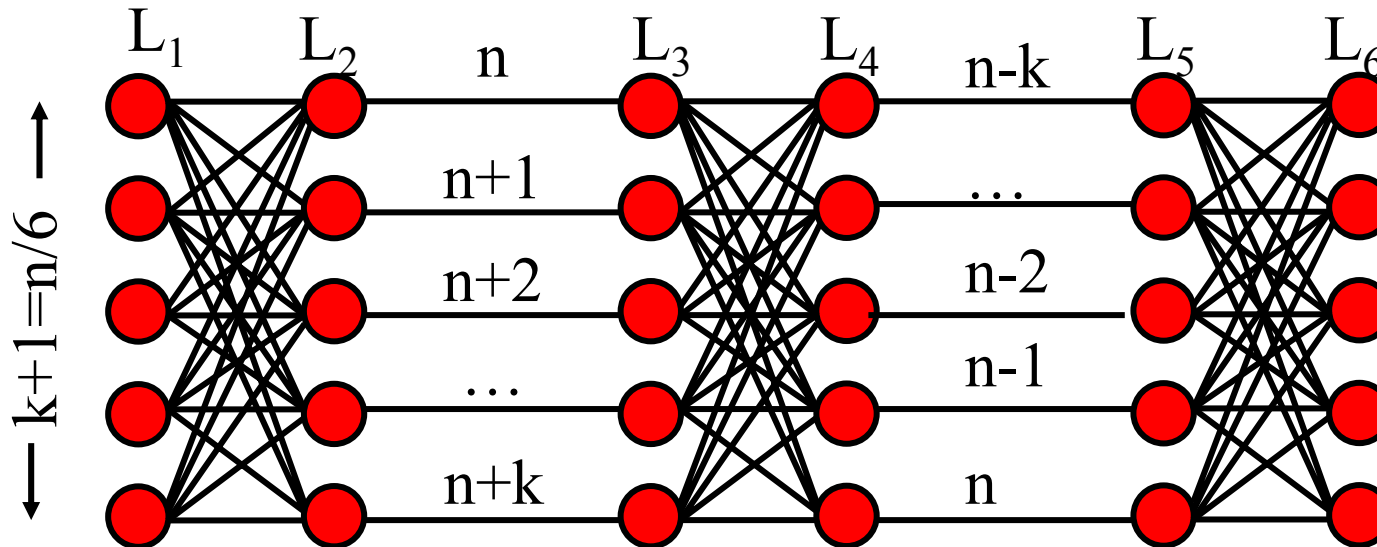
Eigenschaften von LHPs (6)

Theorem: Sei G Graph mit einer Update-Sequenz Σ und sei m die maximale Kantenanzahl in G . Falls SP eindeutig sind und es zu jeder Zeit höchstens z HPs zwischen jedem Knotenpaar gibt, dann ist die Anzahl der Wege, die neu in die Menge der LHPs per Update hineinkommt

- $O(zmn)$ im Worst Case
- $O(zn^2)$ amortisiert über $\Omega(m/n)$ Update Operationen

Beweis: wie bei LSPs

Beispiel mit $\Omega(n^3)$ HPs



Kanten von links nach rechts gerichtet und Kosten 1 bzw. Bild.

Phase 1 (L_2, L_3): Phase 2 (L_4, L_5): Update-Sequenz: Ph1, Ph2,

(1) $n+1 \rightarrow n-1$

(1) $n-k \rightarrow n+k$

(2) $n+2 \rightarrow n-2$

(2) $n-(k-1) \rightarrow n+(k-1)$

... ..

... ..

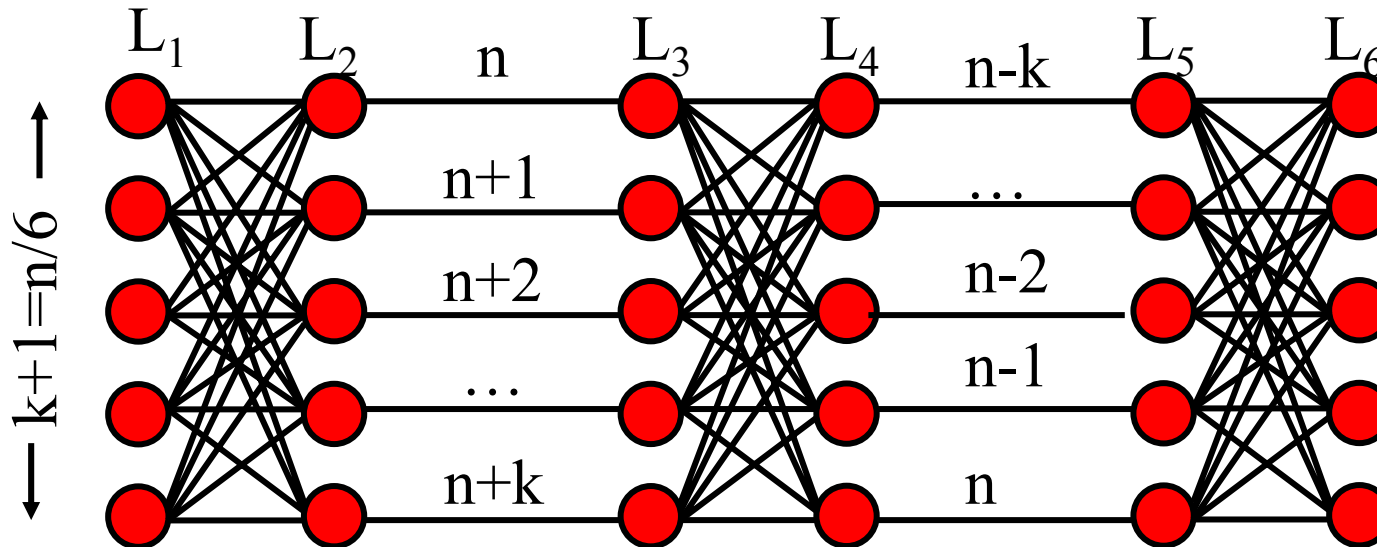
(k) $n+k \rightarrow n-k$

(k) $n-1 \rightarrow n+1$

Phase 1: $(k+1)^2$ Wege zwischen L_1 und L_5 werden HP je
Decrease \rightarrow führt zu $\theta(k^3)$ Wegen in der Menge \mathcal{HP}

ENDE

Beispiel mit $\Omega(n^3)$ HPs



Kanten von links nach rechts gerichtet und Kosten 1 bzw. Bild.

Phase 1 (L_2, L_3): Phase 2 (L_4, L_5): Update-Sequenz: Ph1, Ph2,

(1) $n+1 \rightarrow n-1$

(1) $n-k \rightarrow n+k$

(2) $n+2 \rightarrow n-2$

(2) $n-(k-1) \rightarrow n+(k-1)$

... ..

... ..

(k) $n+k \rightarrow n-k$

(k) $n-1 \rightarrow n+1$

Alle LHPs zu aktualisieren
kann zu lange dauern:
 $\Omega(n^3)$ per Update

Phase 2: je 1 neuer HP zwischen jedem Knotenpaar aus L_2
und $L_6 \rightarrow \Omega(k^3)$ neue LHP-Wege von L_1 nach L_6 per Increase

Idee: Glättung (Smoothing)

- Alle LHPs in DS zu halten kann zu lange dauern
- Aber auch: keine Information über die Historie zu haben (LSPs) kann auch zu lange dauern.

- Idee: Finde eine Lösung, die dazwischen liegt:
- Reduziere die Anzahl der LHPs
- Dies gelingt durch „Schein“-Updates (Cleanup Updates) von Wegen.
- Idee: Transformiere die gegebene Update Sequenz Σ in eine geglättete Sequenz $F(\Sigma)$, die Cleanup-Updates enthält.

- Ein Cleanup-Update ändert die Kosten nicht.
- Es werden zur Zeit t immer ein Update und mehrere Cleanup-Updates gemacht

Datenstrukturen

- $w(P(x \rightarrow y))$: Kosten des Weges $P(x \rightarrow y)$ // bei SP-Wegen = $\text{dist}[]$
- $P(x, y)$: Menge aller LHPs von x nach y (in PrioQ)
- $P^*(x, y)$: Menge aller HPs von x nach y (in PrioQ)
- $L(P(x \rightarrow y))$: Liste der möglichen $P(x \rightarrow y)$ path extension Wege nach vorn (*left*), die LHP sind
- $L^*(P(x \rightarrow y))$: Liste der möglichen $P(x \rightarrow y)$ path extension Wege nach vorn (*left*), die HP sind
- $R(P(x \rightarrow y))$: Liste der möglichen $P(x \rightarrow y)$ path extension Wege nach hinten (*right*), die LHP sind
- $R^*(P(x \rightarrow y))$: Liste der möglichen $P(x \rightarrow y)$ path extension Wege nach hinten (*right*), die HP sind
- $t(v)$: Speichert für jeden Knoten Zeitpunkt des letzten Knoten-Updates

Algorithmus Fully-Dynamic APSP

Fully-Update(v, w')

- $\text{time} \leftarrow \text{time} + 1$
- $t(v) \leftarrow \text{time}$
- $\text{Update}(v, w')$
- Für jeden Knoten $u \in V$:
 - Falls $\log_2(\text{time} - t(u))$ ganzzahlig ist: $\text{Update}(u, w)$

Wenn Knoten v zur Zeit $t = t(v)$ einen echten Knoten-Update hatte, dann erhält er zu den folgenden Zeiten einen Cleanup-Update:

$t+1, t+2, t+4, t+8, \dots, t+2^p$

Zur Erinnerung: Algorithmus Update()

Algorithmus Update(v, w'):

1. Cleanup(v): entfernt alle Wege, die v enthalten
2. Fixup(v, w'): Addiere alle neuen SP's und LSP's

Analyse Korrektheit

- Falls die kürzeste Wege eindeutig sind, dann werden die Mengen $P(x,y)$ und $P^*(x,y)$ für jedes Knotenpaar x und y korrekt aktualisiert.

Beweis: genau wie für LSP letztes Mal.

Analyse Laufzeit

- Sei Σ eine Update-Sequenz der Länge k . Die geglättete Sequenz $F(\Sigma)$ hat die folgenden Eigenschaften:
 - Es gibt pro Original-Update höchstens $O(\log k)$ von ihm initiierte Schein-Updates (Cleanup-Updates).
 - Die Graphen, die durch Σ und durch $F(\Sigma)$ produziert werden sind gleich.
 - Es gibt zu jedem Zeitpunkt höchstens $O(\log k)$ HPs zwischen jedem Knotenpaar im Graphen mit Sequenz $F(\Sigma)$.

Beweis: Die ersten beiden Behauptungen sind klar.

Dritte Behauptung: z.z. ist: Seien P^1, \dots, P^z HPs zur Zeit t , dann müssen z Updates $\sigma^1, \dots, \sigma^z \in \Sigma$ existieren mit $t(\sigma^i) + 2 \lfloor \log(t - t(\sigma^i)) \rfloor \leq t(\sigma^{i+1}) \leq t$ für $1 \leq i \leq z-1$. (o.Bw.)

Daraus folgt $z = O(\log t) = O(\log k)$

Analyse Laufzeit

- In einer Update-Sequenz von $\Omega(m/n)$ Operationen wird, falls es zu jedem Zeitpunkt höchstens z HPs zwischen jedem Knotenpaar gibt, jede Update-Operation in $O(zn^2 \log n)$ amortisierter Zeit durchgeführt.
- Der Speicherplatz beträgt $O(zmn)$.

Beweis: wie letztes Mal bei LSPs:

- Cleanup(v): maximal $O(zn^2)$ LHPs können durch v laufen; pro Iteration $O(\log n)$: $O(zn^2 \log n)$
- Fixup():
 - Phase 1: $O(n \log n)$
 - Phase 2: $O(n^2 \log n)$
 - Phase 3: $O(zn^2 \log n)$

Analyse Laufzeit

- In einer Update-Sequenz von $\Omega(m/n)$ Operationen wird jede Update-Operation in $O(n^2 \log^3 n)$ amortisierter Zeit und jede Distanz und Weg-Nachfrage in optimaler Zeit durchgeführt. Platzverbrauch ist $O(mn \log n)$
- Wegen der Glättung existieren bei einer Sequenz von k Updates höchstens $z=O(\log k)$ HPs zwischen jedem Knotenpaar zu jedem Zeitpunkt.
- Jeder Update Aufruf kostet $O(n^2 \log n \log k)$ amortisierte Zeit
- Jeder Fully-Update Aufruf zieht $O(\log k)$ Update-Aufrufe nach sich
- Das sind insgesamt $O(n^2 \log n \log^2 k)=O(n^2 \log^3 n)$ für jede Sequenz der Länge k , falls k polynomiell in n ist.
- Falls dies nicht der Fall ist, dann: Restart Algorithmus alle $\theta(n^2)$ Operationen.

Neuere Entwicklungen

- Mikkel Thorup erreicht mit einer neuen Glättungs-Strategie eine Laufzeit von $O(n^2(\log n + \log^2(m/n)))$

Offene Probleme

- Speicherplatzreduktion
- Fully-Dynamic Algorithms für Single-Source Shortest Path?

und jetzt: 6.4 Experimenteller Vergleich