

# Algorithm Engineering - Mitschrift vom 28.11.2005

## Decrease-Only ASPS

Unter der Annahme, dass man alle decrease-Operationen bis zu einem bestimmten Zeitpunkt kennt, kann man die Situation rückwärts betrachten. In der Rückrichtung hat man dann nur increase-Operationen und kann den Algorithmus vom letzten Mal anwenden. Eigentlich gehen wir jedoch von einem Online-Problem aus.

### Beobachtung

Bei einem update laufen alle neuen kürzesten Wege über  $v$ .

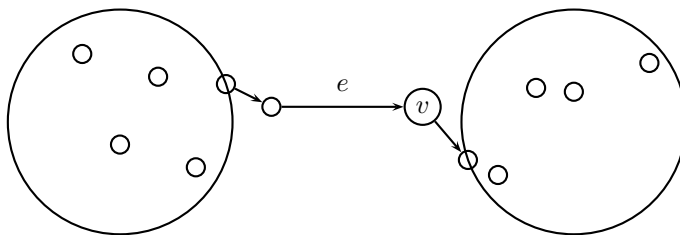


Abbildung 1: SSSP über  $v$

### Erinnerung

Laufzeit von Dijkstra mit Heap:

Binärheap	extractMin	$n \log n$	(für alle Knoten)
	update	$\frac{m \log n}{(n + m) \log n}$	(für alle Kanten)
Fibonacci		$(m + n) \log n$	

Tabelle 1: Laufzeiten von Dijkstra je nach verwendetem Heap

### Vorgehensweise bei decrease-only

Starte von  $v$  aus SSSP und von allen Knoten aus SSSP nach  $v$  (hierfür müssen die Kanten umgedreht werden). Betrachte dann alle Knotenpaare und vergleiche, ob der Weg über  $v$  kürzer ist.

### Decrease / Increase

Im Worst Case gibt es sehr große Änderungen der Menge der Locally Shortest Paths, wenn increase und decrease beide vorkommen können. Siehe dazu Folie 5, hier sollen die dicken, horizontalen, schwarzen Linien die Kanten mit den geringsten Gewichten sein.

## Locally Historical Paths

Idee: Aktualisierung nicht nach jedem Update, sondern nur, wenn auf dem Knoten ein vertex update durchgeführt wird.

### Eigenschaften von LHPs

Zeichnung zu Folie 11 über die maximale Anzahl von LHPs:

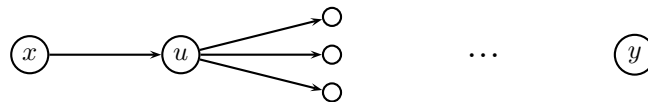


Abbildung 2:  $m$  mögliche Kanten,  $n$  mögliche Zielknoten

### Hinweis

Historical Paths sind nicht notwendigerweise disjunkt.

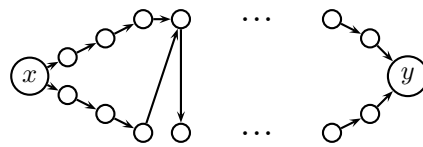


Abbildung 3: verschiedene historische Pfade von  $x$  nach  $y$