Breaking the Sorting Barrier for Directed Single-Source Shortest Paths

Ein Blick über die wissenschaftliche Arbeit

B. Tascan, B. Durie, S. Lackner

January 20, 2026

Verzeichnis

SSSP Algorithmen

- Single-Source Shortest Path Algorithmen (SSSP's)

SSSP Algorithmen

- Single-Source Shortest Path Algorithmen (SSSP's)
  - Input: ein Graph G und ein Startknoten s
  - Output: Der kürzeste Pfad von s nach alle anderen Knoten

SSSP Algorithmen

- Single-Source Shortest Path Algorithmen (SSSP's)
  - Input: ein Graph G und ein Startknoten s
  - Output: Der kürzeste Pfad von s nach alle anderen Knoten
- Verwendet meistens für:
  - GPS-Navigation
  - Network Routing
  - Video-Spiele

SSSP Algorithmen

- Single-Source Shortest Path Algorithmen (SSSP's)
  - Input: ein Graph G und ein Startknoten s
  - Output: Der kürzeste Pfad von s nach alle anderen Knoten
- Verwendet meistens für:
  - GPS-Navigation
  - Network Routing
  - Video-Spiele
- Es gibt zwei SSSP Algorithmen, die uns interessieren:
  - Dijkstra's Algorithmus,
  - Bellman-Ford's Algorithmus

kleiner Exkurs

- Greedy Algorithmen:
  Algorithmen, die in jedem Schritt die lokalbeste Entscheidung treffen.

kleiner Exkurs

- Greedy Algorithmen:
  Algorithmen, die in jedem Schritt die lokalbeste Entscheidung treffen.
- Frontier:
  Die Menge bereits abgeschlossener Knoten, durch die jeder noch nicht gefundene kürzeste Pfad zwingend hindurchführen muss.

kleiner Exkurs

- Greedy Algorithmen:
  Algorithmen, die in jedem Schritt die lokalbeste Entscheidung treffen.
- Frontier:
  Die Menge bereits abgeschlossener Knoten, durch die jeder noch nicht gefundene kürzeste Pfad zwingend hindurchführen muss.
- Ein Knoten relaxieren:
  Alle Kanten von einem Knoten checken und die Distanzen verbessern, falls ein kürzerer Pfad entstanden ist.

Dijkstra

- Von Edsger W. Dijkstra im Jahr 1959 publiziert.

Dijkstra

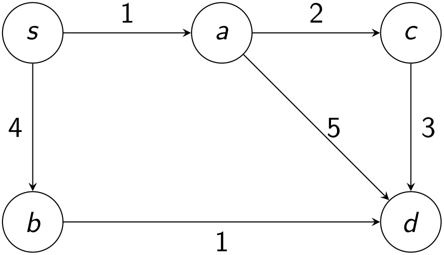
- Von Edsger W. Dijkstra im Jahr 1959 publiziert.
- 66 Jahre lang der asymptotisch schnellste Algorithmus

Dijkstra

- Von Edsger W. Dijkstra im Jahr 1959 publiziert.
- 66 Jahre lang der asymptotisch schnellste Algorithmus
- Dijkstra's ist ein *Greedy Algorithmus*.

Dijkstra

- min-Prioritätswarteschlange zur Auswahl des nächsten Knotens
- $\mathcal{O}(m + n \log n)$
-



Bellman-Ford

- Unabhängig von Bellman (1958) und Ford (1956) entwickelt

Bellman-Ford

- Unabhängig von Bellman (1958) und Ford (1956) entwickelt
- Klassischer SSSP-Algorithmus für gerichtete Graphen

Bellman-Ford

- Unabhängig von Bellman (1958) und Ford (1956) entwickelt
- Klassischer SSSP-Algorithmus für gerichtete Graphen
- Funktioniert mit *negativen Kantengewichten*

Bellman-Ford

- Unabhängig von Bellman (1958) und Ford (1956) entwickelt
- Klassischer SSSP-Algorithmus für gerichtete Graphen
- Funktioniert mit *negativen Kantengewichten*
- Kein Greedy-Algorithmus

Bellman-Ford

- Initialisiere Distanzen

Bellman-Ford

- Initialisiere Distanzen
- Wiederhole $n - 1$ Mal:

Bellman-Ford

- Initialisiere Distanzen
- Wiederhole $n - 1$ Mal:
  - Relaxiere **alle Kanten**

Bellman-Ford

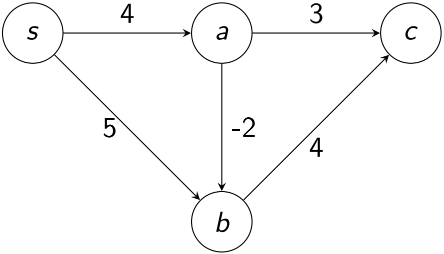
- Initialisiere Distanzen
- Wiederhole $n - 1$ Mal:
  - Relaxiere **alle Kanten**
- Optional: zusätzliche Runde erkennt negative Zyklen

Bellman-Ford

- Initialisiere Distanzen
- Wiederhole $n - 1$ Mal:
  - Relaxiere **alle Kanten**
- Optional: zusätzliche Runde erkennt negative Zyklen
- Jede Iteration erweitert bekannte kürzeste Pfade um genau eine Kante

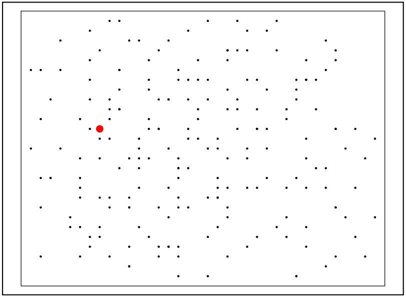
Bellman-Ford

- Laufzeit: $\mathcal{O}(n \cdot m)$
- Deutlich langsamer als Dijkstra
- Dafür sehr robust



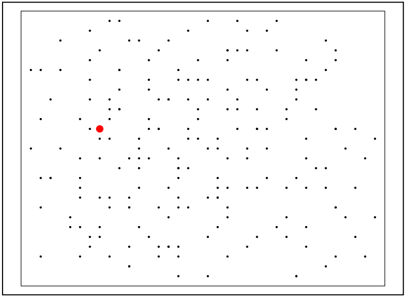
Der Algorithmus - Einleitung

- Dijkstra in $\mathcal{O}(m + n \log n)$



Der Algorithmus - Einleitung

- Dijkstra in $\mathcal{O}(m + n \log n)$
- Sortierbarriere $\Omega(n \log n)$



Der Algorithmus - Lösungsansatz

Sortierbarriere umgehen durch:

- **Eigene Datenstruktur**
  Ermöglicht effizientes *Bucketing*, verhindert Sortierung

Der Algorithmus - Lösungsansatz

Sortierbarriere umgehen durch:

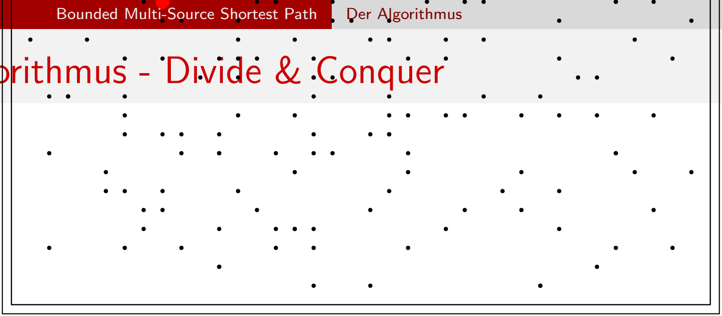
- **Eigene Datenstruktur**
  Ermöglicht effizientes *Bucketing*, verhindert Sortierung
- **Pivoting**
  Reduziert den Rechenaufwand

Der Algorithmus - Lösungsansatz

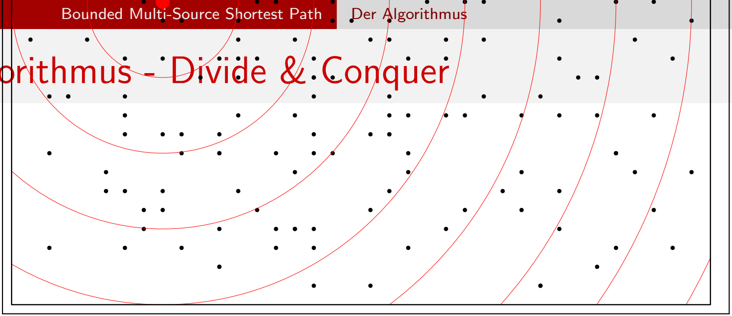
Sortierbarriere umgehen durch:

- **Eigene Datenstruktur**
  Ermöglicht effizientes *Bucketing*, verhindert Sortierung
- **Pivoting**
  Reduziert den Rechenaufwand
- **Divide & Conquer**
  Mindert die Problemgröße durch Rekursion

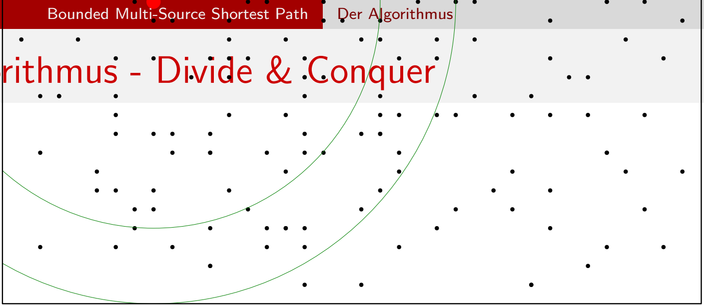
Der Algorithmus - Divide & Conquer



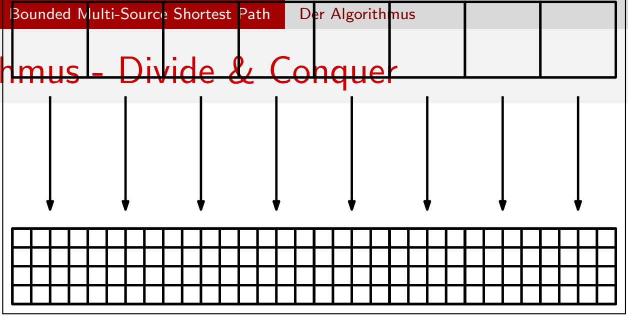
Der Algorithmus - Divide & Conquer



Der Algorithmus - Divide & Conquer



Der Algorithmus - Divide & Conquer

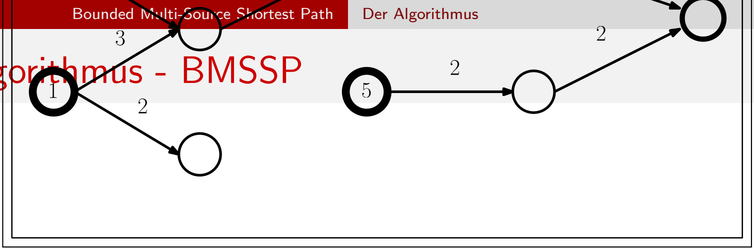


Der Algorithmus - Divide & Conquer

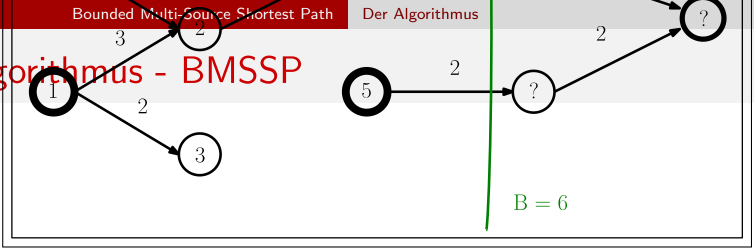
Wozu das ganze? ist notwendiges Hilfsmittel für:

- **BMSSP**
  funktioniert nur dank Abgrenzungen
- **Pivots**
  erlaubt schnelle Auswahl von wichtigen Knoten

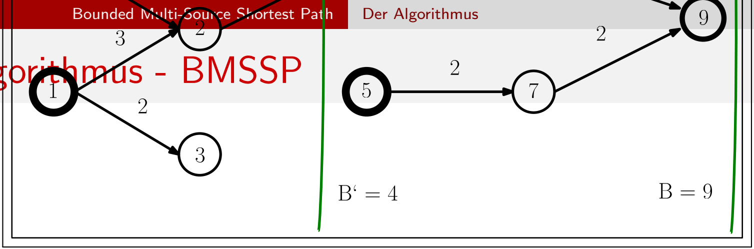
Der Algorithmus - BMSSP



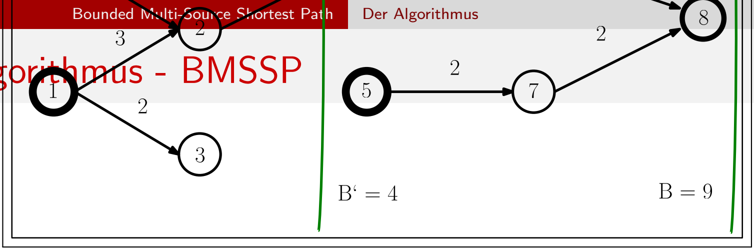
Der Algorithmus - BMSSP



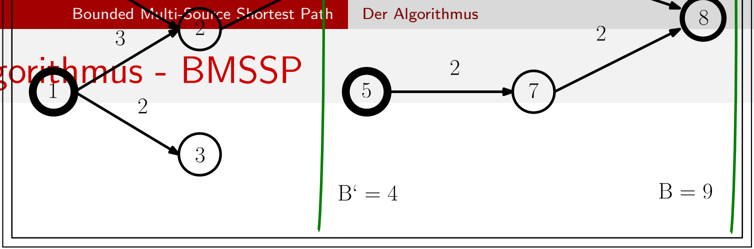
Der Algorithmus - BMSSP



Der Algorithmus - BMSSP



Der Algorithmus - BMSSP

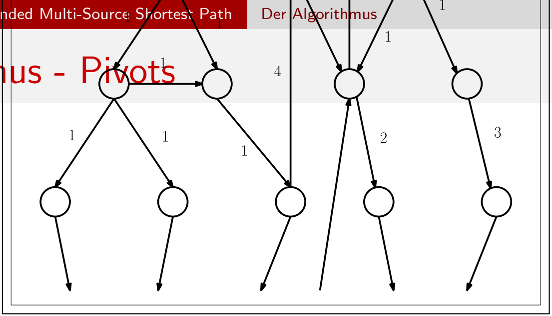


Der Algorithmus - BMSSP

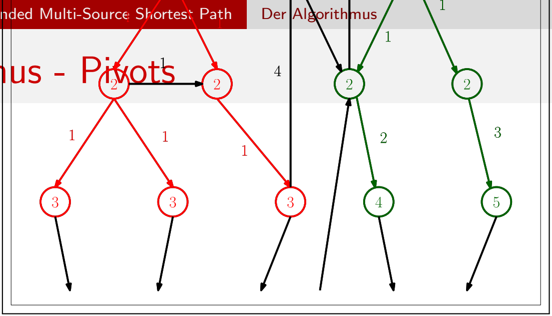
Wozu das ganze?

- **Sortierbarriere**
  Ermöglicht Umgehung der $\Omega(n \log n)$ Schranke dank Bucketing

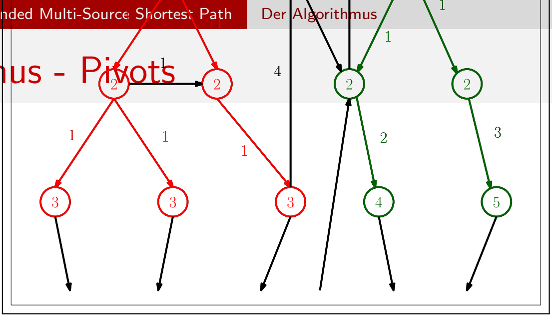
Der Algorithmus - Pivots



Der Algorithmus - Pivots

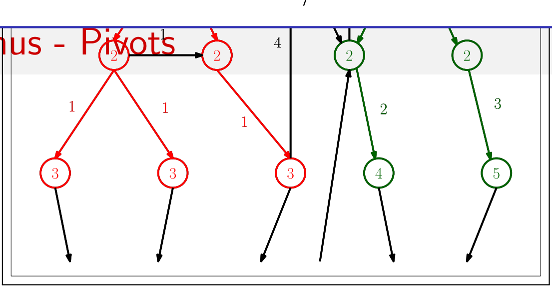


Der Algorithmus - Pivots

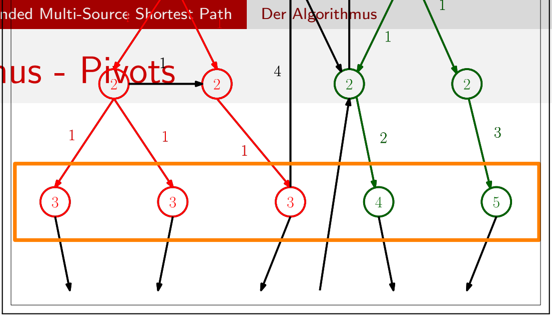


$\Rightarrow$ Maximal n/k Pivots

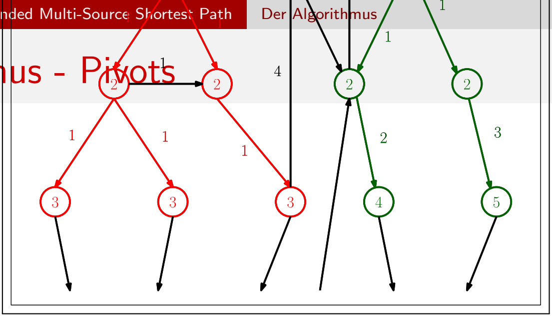
Der Algorithmus - Pivots



Der Algorithmus - Pivots

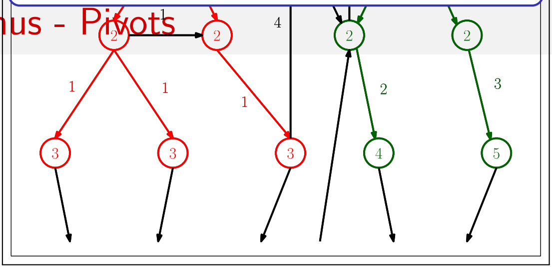


Der Algorithmus - Pivots



⇒ Nur Pivots werden weiterverfolgt

Der Algorithmus - Pivots

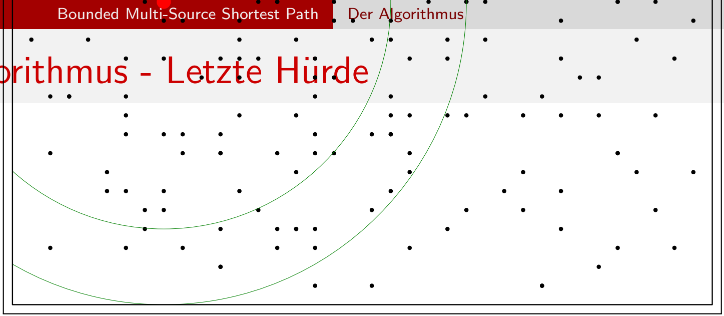


Der Algorithmus - Pivots

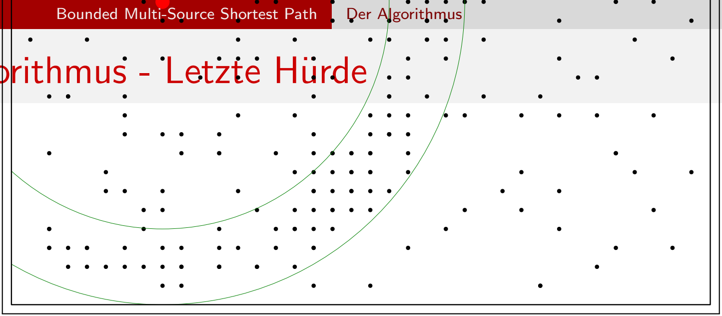
Wozu das ganze? Hauptgrund für Geschwindigkeit:

- **Kostenreduktion**
  reduziert Aufwand durch gezielte Knotenwahl
- **Faktor**
  reduziert BMSSP Knotenanzahl um $\log^{1/3} n$

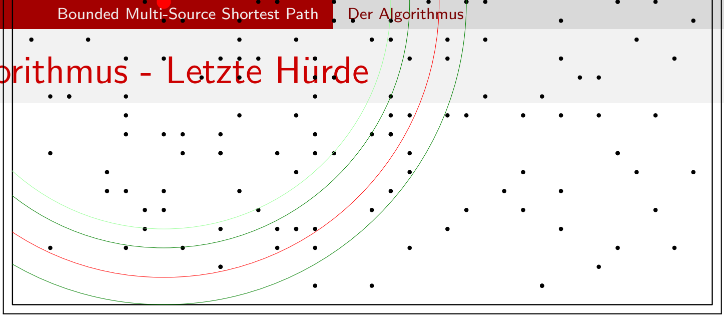
Der Algorithmus - Letzte Hürde



Der Algorithmus - Letzte Hürde



Der Algorithmus - Letzte Hürde



Die Datenstruktur

- Dijkstra hat eine asymptotische Laufzeit von $\mathcal{O}(m \log n)$

Die Datenstruktur

- Dijkstra hat eine asymptotische Laufzeit von $\mathcal{O}(m \log n)$
- Um diese Laufzeit zu verbessern, wird eine spezielle Datenstruktur benötigt

Die Datenstruktur

- Dijkstra hat eine asymptotische Laufzeit von $\mathcal{O}(m \log n)$
- Um diese Laufzeit zu verbessern, wird eine spezielle Datenstruktur benötigt
- Diese Struktur ist eine sogenannte Block-based linked List

Die Datenstruktur

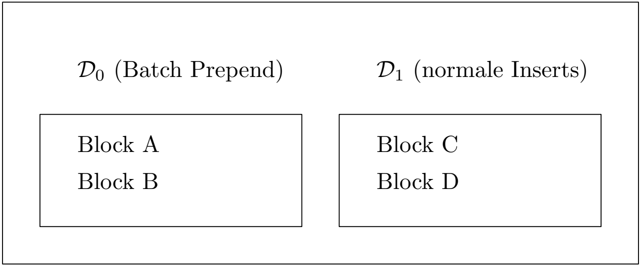
- Es gibt zwei Sequenzen an Blöcken, $\mathcal{D}_0$ und $\mathcal{D}_1$, welche beide Linked Lists sind mit maximal M Key/Value Paaren und einem Upperbound von B

Die Datenstruktur

- Es gibt zwei Sequenzen an Blöcken, $\mathcal{D}_0$ und $\mathcal{D}_1$, welche beide Linked Lists sind mit maximal M Key/Value Paaren und einem Upperbound von B
- $\mathcal{D}_0$ enthält Batch Prepend Elemente, unbounded

Die Datenstruktur

- Es gibt zwei Sequenzen an Blöcken, $\mathcal{D}_0$ und $\mathcal{D}_1$, welche beide Linked Lists sind mit maximal M Key/Value Paaren und einem Upperbound von B
- $\mathcal{D}_0$ enthält Batch Prepend Elemente, unbounded
- $\mathcal{D}_1$ enthält mit Insert eingefügte Elemente, bounded mit $O(\max\{1, N/M\})$



Die Datenstruktur

- Dijkstra Laufzeit $\mathcal{O}(m \log n)$

Die Datenstruktur

- Dijkstra Laufzeit $\mathcal{O}(m \log n)$
- Verbesserung mit spezieller Datenstruktur

Die Datenstruktur

- Dijkstra Laufzeit $\mathcal{O}(m \log n)$
- Verbesserung mit spezieller Datenstruktur
- Block-based Linked List

Die Datenstruktur

- Zwei Sequenzen $\mathcal{D}_0$ und $\mathcal{D}_1$

Die Datenstruktur

- Zwei Sequenzen $\mathcal{D}_0$ und $\mathcal{D}_1$
- Maximal M Key/Value Paaren

Die Datenstruktur

- Zwei Sequenzen $\mathcal{D}_0$ und $\mathcal{D}_1$
- Maximal M Key/Value Paaren
- Upperbound von B

$\mathcal{D}_0$ (Batch Prepend)

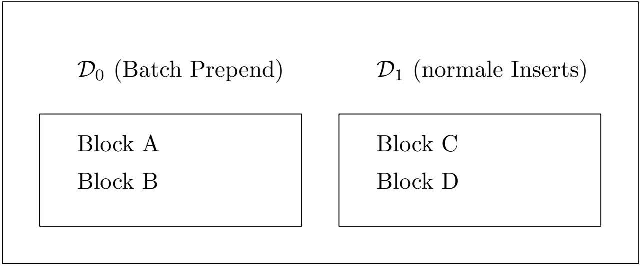
Block A
Block B

$\mathcal{D}_1$ (normale Inserts)

Block C
Block D

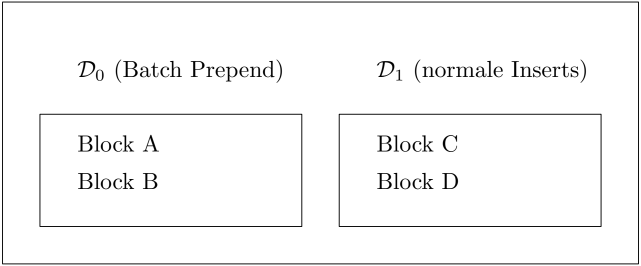
Die Datenstruktur

- Zwei Sequenzen $\mathcal{D}_0$ und $\mathcal{D}_1$
- Maximal M Key/Value Paaren
- Upperbound von B
- $\mathcal{D}_0$ enthält Batch Prepend Elemente, unbounded



Die Datenstruktur

- Zwei Sequenzen $\mathcal{D}_0$ und $\mathcal{D}_1$
- Maximal M Key/Value Paaren
- Upperbound von B
- $\mathcal{D}_0$ enthält Batch Prepend Elemente, unbounded
- $\mathcal{D}_1$ enthält mit Insert Elemente, bounded mit $O(\max\{N/M\})$

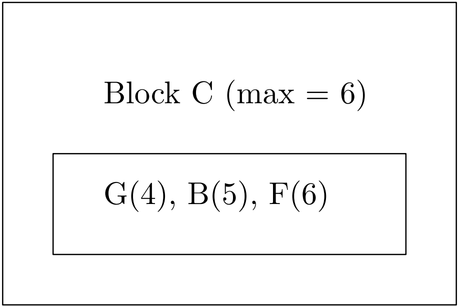


Die Datenstruktur

- Sequenzen sortiert nach Wert

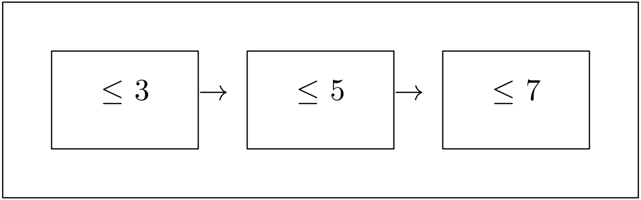
Die Datenstruktur

- Sequenzen sortiert nach Wert
- Balancing mit einem binären Suchbaum



Die Datenstruktur

- Sequenzen sortiert nach Wert
- Balancing mit einem binären Suchbaum



Die Datenstruktur

- Insert

Die Datenstruktur

- Insert
- Batch Prepend

Die Datenstruktur

- Insert
- Batch Prepend
- Pull

Die Datenstruktur

- Insert(a, b):

Die Datenstruktur

- Insert(a, b):
- Mehrere gleiche Key/Value Paare, niedrigere Value bevorzugt

Die Datenstruktur

- Insert(a, b):
- Mehrere gleiche Key/Value Paare, niedrigere Value bevorzugt
- Blocksuche mit Binärbaum

Die Datenstruktur

- Insert(a, b):
- Mehrere gleiche Key/Value Paare, niedrigere Value bevorzugt
- Blocksuche mit Binärbaum
- Insert in Block in $\mathcal{D}_1$

Die Datenstruktur

- Insert(a, b):
- Mehrere gleiche Key/Value Paare, niedrigere Value bevorzugt
- Blocksuche mit Binärbaum
- Insert in Block in $\mathcal{D}_1$
- Laufzeit $O(\max\{\log(N/M)\})$

Die Datenstruktur

- Batch Prepend(L):

Die Datenstruktur

- Batch Prepend(L):
- L Key/value Paare

Die Datenstruktur

- Batch Prepend(L):
- L Key/value Paare
- Insert, am Beginn von $\mathcal{D}_0$

Die Datenstruktur

- Batch Prepend(L):
- L Key/value Paare
- Insert, am Beginn von $\mathcal{D}_0$
- Laufzeit $O(L \cdot \max\{\log(L/M)\})$

Die Datenstruktur

- Pull:

$\rightarrow$

Die Datenstruktur

- Pull:
- Menge S' an kleinsten Werten, Upper Bound x

→

Die Datenstruktur

- Pull:
- Menge S' an kleinsten Werten, Upper Bound x
- Sortierung in Gruppen,statt einer genauen Sortierung

$\rightarrow$

Die Datenstruktur

- Pull:
- Menge S' an kleinsten Werten, Upper Bound x
- Sortierung in Gruppen,statt einer genauen Sortierung
- Laufzeit von Pull $O(|S'|)$

$\rightarrow$

Ist der Algorithmus praktisch umsetzbar?

Benchmarking

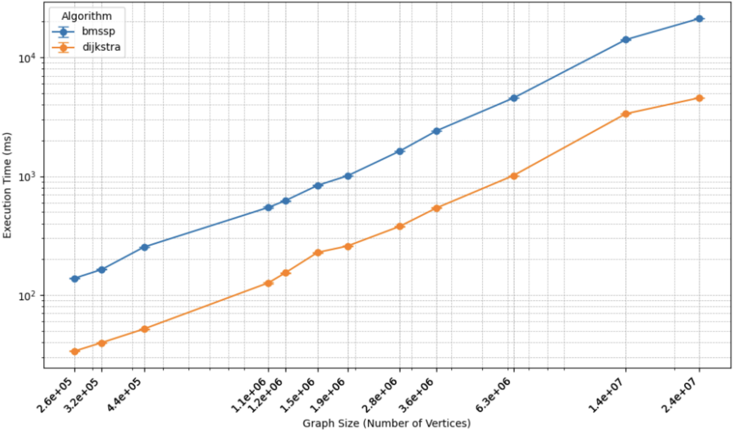


Figure 4 Execution time versus graph size for the USA road network instances, plotted on log-log scale.

Vielen Dank für Ihre Aufmerksamkeit!