



openHPI

Einführung in die Mathematik der Algorithmik – Dijkstras Algorithmus II

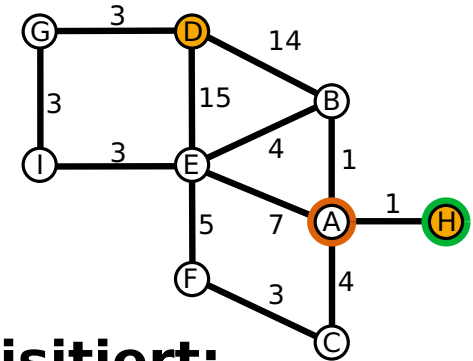
Thomas Bläsius und Pascal Lenzner

Hasso-Plattner-Institut

Universität Potsdam

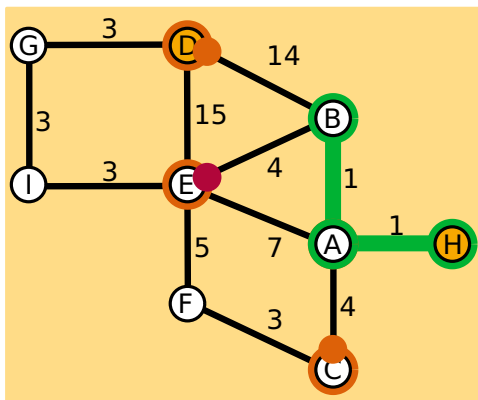
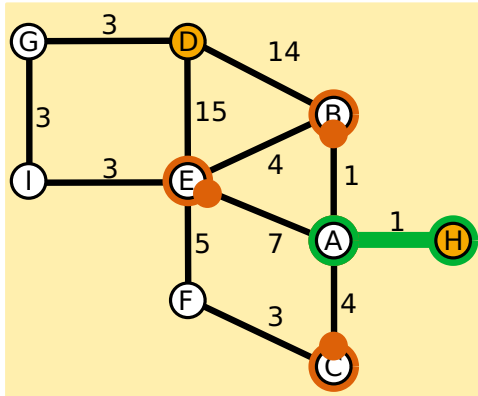
Initialisierung:

- nur Knoten H markiert; Distanz zum Startknoten $H = 0$
- alle Nachbarn X des Startknotens H sind Randknoten mit $dist(X, H) = \text{Länge}(X, H)$; Vorgänger ist H



Solange unmarkierter Randknoten existiert:

- X sei unmarkierter Randknoten mit geringster Distanz zu H
- entferne X aus der Menge der Randknoten
- mache für alle unmarkierten Nachbarn Y von X folgendes:
 - falls Y kein Randknoten:
 - füge Y in die Menge der Randknoten ein
 - $dist(Y, H) = \text{Länge}(Y, X) + dist(X, H)$; Vorgänger von Y ist X
 - sonst:
 - falls $dist(Y, H) > \text{Länge}(Y, X) + dist(X, H)$:
 - aktualisiere $dist(Y, H)$ zu $\text{Länge}(Y, X) + dist(X, H)$
 - X wird Vorgänger von Y
- markiere X



Wie verwalten wir clever die Randknoten und deren Distanz zum Startknoten?

Aufgaben:

Finden und Entfernen des Randknotens mit geringster Distanz

Einfügen eines neuen Randknotens mit gewisser Distanz

Aktualisieren der Distanz eines Randknotens

ExtractMin Randknoten existiert:

- X sei unmarkierter Randknoten mit geringster Distanz zu H
- entferne X aus der Menge der Randknoten
- mache für alle unmarkierten Nachbarn Y von X folgendes:

Insert falls Y kein Randknoten:

- füge Y in die Menge der Randknoten ein
- $dist(Y, H) = \text{Länge}(Y, X) + dist(X, H)$; Vorgänger von Y ist X

DecreaseKey

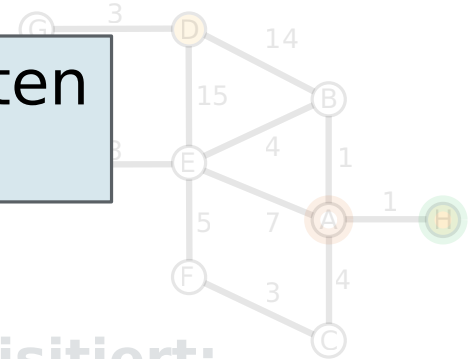
sonst:

■ falls $dist(Y, H) > dist(Y, X) + dist(X, H)$:

- aktualisiere $dist(Y, H)$ zu $\text{Länge}(Y, X) + dist(X, H)$

■ X wird Vorgänger von Y

■ markiere X



Wie verwalten wir clever die Randknoten und deren Distanz zum Startknoten?

Aufgaben:

ExtractMin

Insert

DecreaseKey

- haben: Randknoten, jeweils mit Distanz zum Startknoten
- wollen: alle drei Operationen möglichst effizient unterstützen
- benötigen geeignete Datenstruktur!

Datenstrukturen, die die Operationen **ExtractMin** **Insert** **DecreaseKey** unterstützen, heißen **Prioritätswarteschlange (Priority Queue)**

Idee:

- Elemente warten in einer Warteschlange
- jedes Element hat eine Priorität, die definiert, wie wichtig das Element ist
- das Element mit der höchsten Priorität kommt als nächstes an die Reihe

Wie verwalten wir clever die Randknoten und deren Distanz zum Startknoten?

Aufgaben:

ExtractMin

Insert

DecreaseKey

Prioritätswarteschlange (Priority Queue)

- Elemente warten in einer Warteschlange
- jedes Element hat eine Priorität, die definiert, wie wichtig das Element ist
- das Element mit der höchsten Priorität kommt als nächstes an die Reihe

Verwaltung der Randknoten mit einer Priority Queue:

- benutze Distanz zum Startknoten als Priorität
- je kleiner die Distanz, desto höher die Priorität

Wie verwalten wir clever die Randknoten und deren Distanz zum Startknoten?

Aufgaben:

ExtractMin

Insert

DecreaseKey

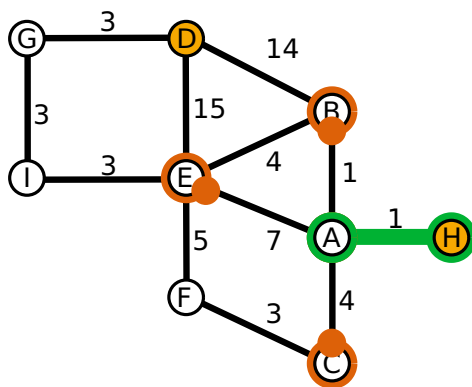
Priority Queue:

- Elemente warten in Warteschlange
- jedes Element hat Prioritätswert
- das Element mit der höchsten Priorität kommt als nächstes an die Reihe

Verwaltung der Randknoten mit einer Priority Queue:

- benutze Distanz zum Startknoten als Priorität
- je kleiner die Distanz, desto höher die Priorität

1. Idee: Verwalte Randknoten als nach Priorität sortierte Liste



Priority Queue: $(B, 2) \rightarrow (C, 5) \rightarrow (E, 8)$

ExtractMin

$(C, 5) \rightarrow (E, 8)$ (B,2) wird ausgegeben

Insert

$(C, 5) \rightarrow (E, 8) \rightarrow (D, 16)$ (D,16) wurde eingefügt

Wie verwalten wir clever die Randknoten und deren Distanz zum Startknoten?

Aufgaben:

ExtractMin

Insert

DecreaseKey

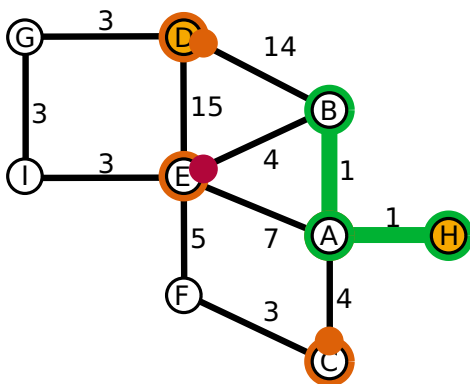
Priority Queue:

- Elemente warten in Warteschlange
- jedes Element hat Prioritätswert
- das Element mit der höchsten Priorität kommt als nächstes an die Reihe

Verwaltung der Randknoten mit einer Priority Queue:

- benutze Distanz zum Startknoten als Priorität
- je kleiner die Distanz, desto höher die Priorität

1. Idee: Verwalte Randknoten als nach Priorität sortierte Liste



Priority Queue: $(B, 2) \rightarrow (C, 5) \rightarrow (E, 8)$

ExtractMin

$(C, 5) \rightarrow (E, 8)$ $(B, 2)$ wird ausgegeben

Insert

$(C, 5) \rightarrow (E, 8) \rightarrow (D, 16)$ $(D, 16)$ wurde eingefügt

DecreaseKey

$(C, 5) \rightarrow (E, 6) \rightarrow (D, 16)$

Priorität von E wurde von 8 auf 6 gesenkt

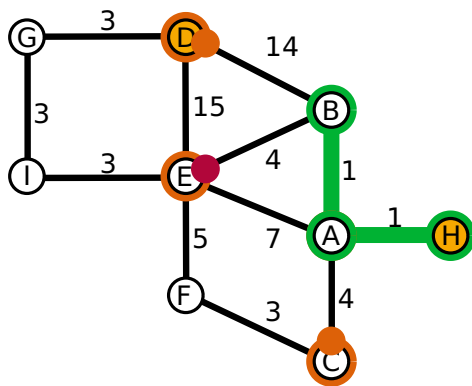
Problem:

Müssen viel Aufwand betreiben,
um die Liste sortiert zu halten!

Lösung:

Benutze spezialisierte Daten-
struktur: Binäre Min-Heaps!

1. Idee: Verwalte Randknoten als nach Priorität sortierte Liste



Priority Queue: $(B, 2) \rightarrow (C, 5) \rightarrow (E, 8)$

ExtractMin

$(C, 5) \rightarrow (E, 8)$ $(B, 2)$ wird ausgegeben

Insert

$(C, 5) \rightarrow (E, 8) \rightarrow (D, 16)$ $(D, 16)$ wurde eingefügt

DecreaseKey

$(C, 5) \rightarrow (E, 6) \rightarrow (D, 16)$

Priorität von E
wurde von 8
auf 6 gesenkt