Informatik II

Übung 10

Giuseppe Accaputo, Felix Friedrich, Patrick Gruntz, Tobias Klenze, Max Rossmannek, David Sidler, Thilo Weghorn

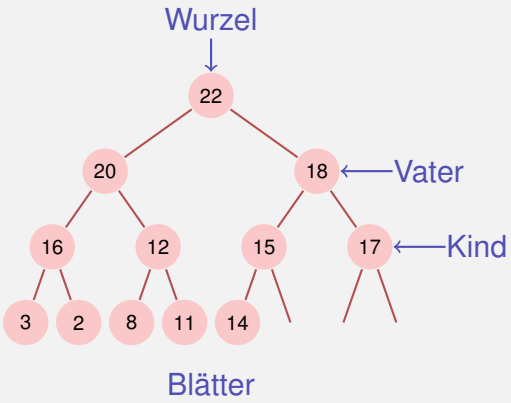
FS 2017

Heutiges Programm

1. Heaps
2. Graphen Traversieren

[Max-]Heap[1]

Binärer Baum mit folgenden Eigenschaften

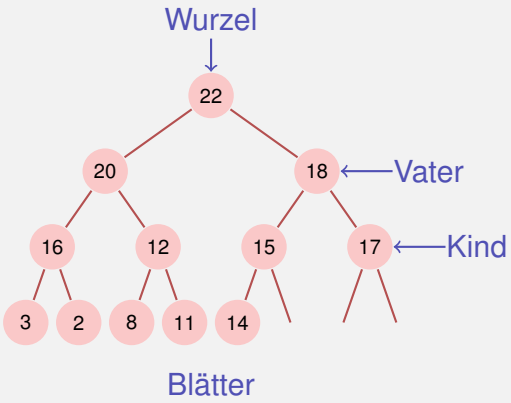


[1] Heap (Datenstruktur), nicht: wie in “Heap und Stack” (Speicherallokation)

[Max-]Heap[1]

Binärer Baum mit folgenden Eigenschaften

1. vollständig, bis auf die letzte Ebene

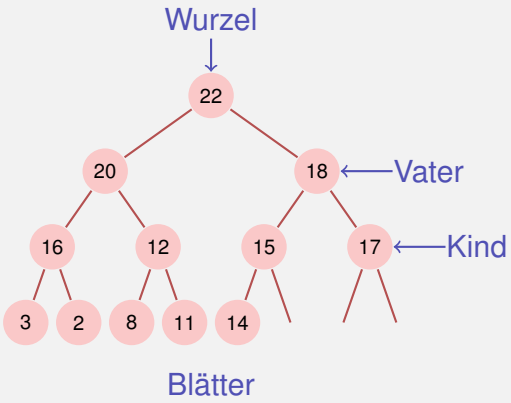


[1] Heap (Datenstruktur), nicht: wie in “Heap und Stack” (Speicherallokation)

[Max-]Heap[1]

Binärer Baum mit folgenden Eigenschaften

1. vollständig, bis auf die letzte Ebene
2. Lücken des Baumes in der letzten Ebene höchstens rechts.

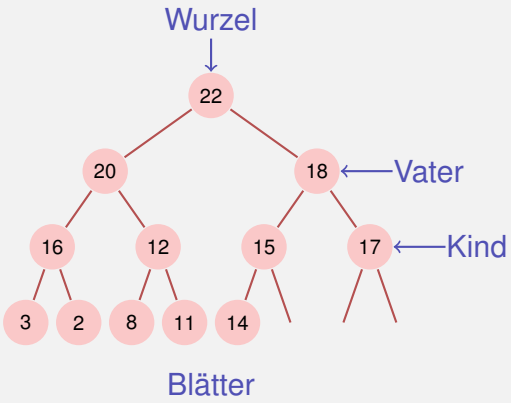


[1] Heap (Datenstruktur), nicht: wie in “Heap und Stack” (Speicherallokation)

[Max-]Heap[1]

Binärer Baum mit folgenden Eigenschaften

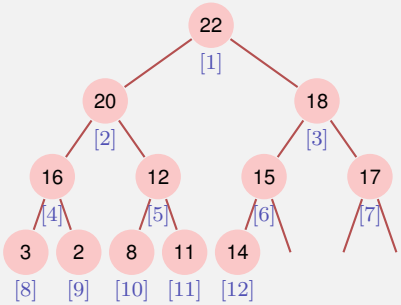
1. vollständig, bis auf die letzte Ebene
2. Lücken des Baumes in der letzten Ebene höchstens rechts.
3. Heap-Bedingung:
   Max-(Min-)Heap: Schlüssel eines Kindes kleiner (grösser) als der des Vaters



[1] Heap (Datenstruktur), nicht: wie in “Heap und Stack” (Speicherallokation)

Heap und Array

Ein Heap lässt sich sehr gut in einem Array speichern:



Es gilt

- Kinder$(i) = \{2i, 2i+1\}$
- Vater$(i) = \lfloor i/2 \rfloor$

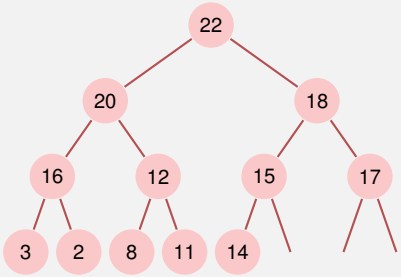
Abhängig von Startindex![2]

[2] Für Arrays, die bei 0 beginnen: $\{2i, 2i+1\} \to \{2i+1, 2i+2\}$, $\lfloor i/2 \rfloor \to \lfloor (i-1)/2 \rfloor$

Datenstruktur ArrayHeap

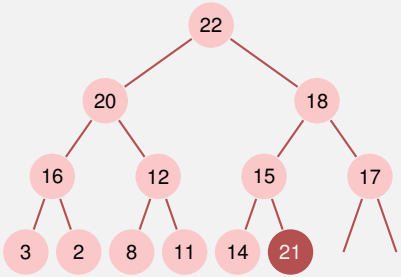
```
public class ArrayHeap {
    float[] data; // Array zum Speichern der Daten
    int used;    // Anzahl belegter Knoten

    ArrayHeap () {
        data = new float[16];
        used = 0;
    }

    int Parent(int of) {
        return (of − 1)/2;
    }

    void Grow() { ... } // Binaeres Vergroessern von data, wenn noetig
}
```

Einfügen



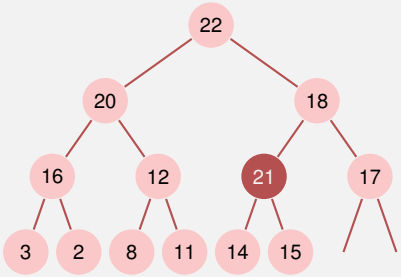
Einfügen

- Füge neues Element an erste freie Stelle ein. Verletzt Heap Eigenschaft potentiell.



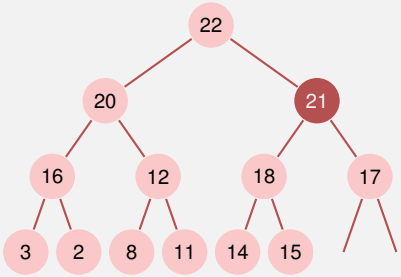
Einfügen

- Füge neues Element an erste freie Stelle ein. Verletzt Heap Eigenschaft potentiell.
- Stelle Heap Eigenschaft wieder her: Sukzessives Aufsteigen.



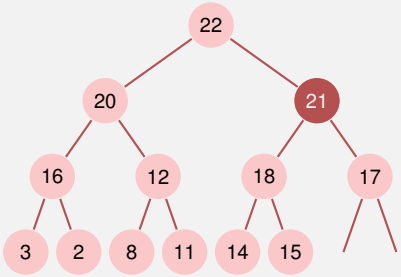
Einfügen

- Füge neues Element an erste freie Stelle ein. Verletzt Heap Eigenschaft potentiell.
- Stelle Heap Eigenschaft wieder her: Sukzessives Aufsteigen.



Einfügen

- Füge neues Element an erste freie Stelle ein. Verletzt Heap Eigenschaft potentiell.
- Stelle Heap Eigenschaft wieder her: Sukzessives Aufsteigen.
- Anzahl Operationen im schlechtesten Fall: $\mathcal{O}(\log n)$



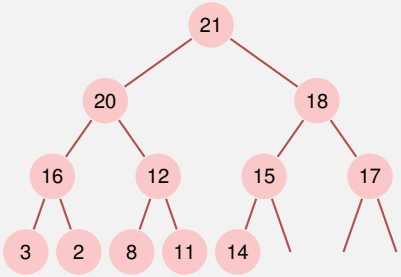
Insert

```
public void Insert(double value) {
    if (used == data.length) {
        Grow();
    }
    int current = used;
    while (current > 0 && value < data[Parent(current)]) {
        data[current] = data[Parent(current)];
        current = Parent(current);
    }
    data[current] = value;
    used++;
}
```

GetMax

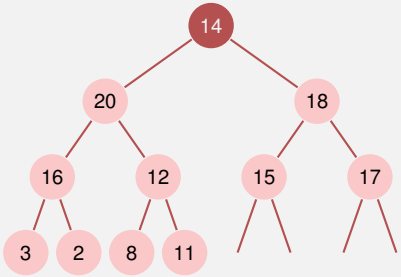
Das grösste Element ist immer an der Wurzel im Baum. Somit kann es sehr schnell ausgelesen werden $\mathcal{O}(1)$.
Wie verhält es sich aber mit Auslesen und Entfernen?
Wiederholtes Entfernen der Wurzel ergibt Schlüssel in absteigender Reihenfolge: das kann z.B. auch zum Sortieren verwendet werden (Heap-Sort Algorithmus.)

Maximum entfernen



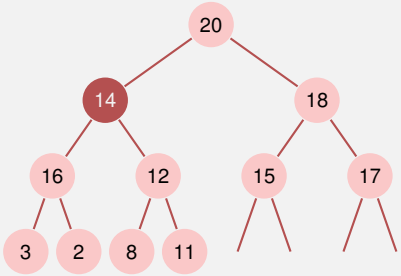
Maximum entfernen

- Ersetze das Maximum durch das unterste rechte Element.



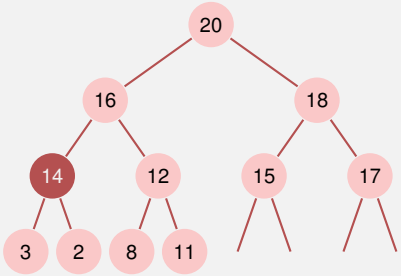
Maximum entfernen

- Ersetze das Maximum durch das unterste rechte Element.
- Stelle Heap Eigenschaft wieder her: Sukzessives Absinken (in Richtung des grösseren Kindes).



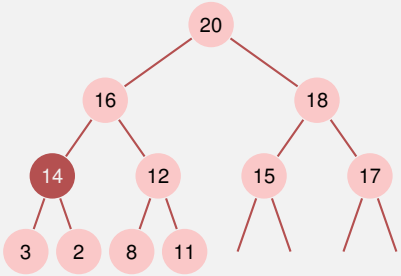
Maximum entfernen

- Ersetze das Maximum durch das unterste rechte Element.
- Stelle Heap Eigenschaft wieder her: Sukzessives Absinken (in Richtung des grösseren Kindes).



Maximum entfernen

- Ersetze das Maximum durch das unterste rechte Element.
- Stelle Heap Eigenschaft wieder her: Sukzessives Absinken (in Richtung des grösseren Kindes).
- Anzahl Operationen im schlechtesten Fall: $\mathcal{O}(\log n)$



Verwendunsbeispiel Heap

Wir können das schnelle Auslesen, Extrahieren, Einfügen von Maximum (bzw. Minimum) im Heap für einen online-Algorithmus des Median nutzen. Wie?
Beobachtung: Der Median bildet sich aus Minimum der oberen Hälfte Daten und/oder Maximum der unteren Hälfte der Daten.

Online Median

Verwende Max-Heap H_{max} und Min-Heap H_{min}.
Bezeichne Anzahl Elemente jeweils mit $|H_{max}|$ und $|H_{min}|$

- Einfügen neuen Wertes v
  - H_{max}, wenn $v \leq max(H_{max})$
  - H_{min}, sonst
- Rebalancieren der beiden Heaps
  - Falls $|H_{max}| > \lfloor n/2 \rfloor$, dann extrahiere Wurzel von H_{max} und füge den Wert bei H_{min} ein.
  - Falls $|H_{max}| < \lfloor n/2 \rfloor$, dann extrahiere Wurzel von H_{min} und füge den Wert bei H_{max} ein.

Gesamt worst-case Komplexität des Einfügens: $\mathcal{O}(\log n)$

Berechnung Median

Berechnung Median

- Wenn n ungerade, dann

$$median = min(H_{min})$$

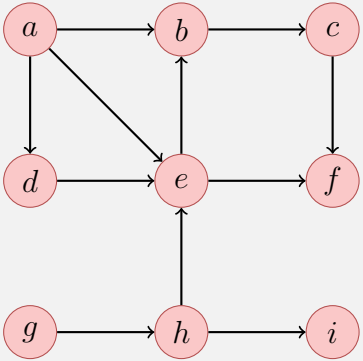
- Wenn n gerade, dann

$$median = \frac{max(H_{max}) + min(H_{min})}{2}$$

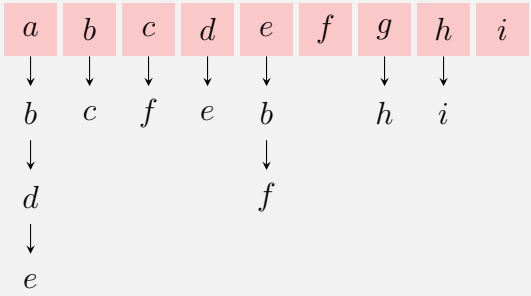
$\Rightarrow$ Worst-case Komplexität $\mathcal{O}(1)$

Tiefensuche

Verfolge zuerst Pfad in die Tiefe, bis nichts mehr besucht werden kann.

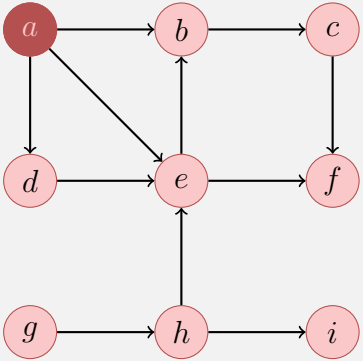


Adjazenzliste

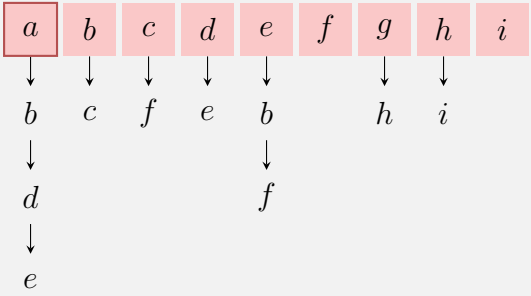


Tiefensuche

Verfolge zuerst Pfad in die Tiefe, bis nichts mehr besucht werden kann.

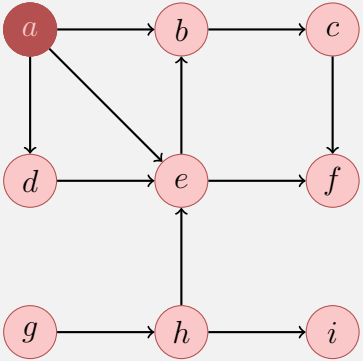


Adjazenzliste

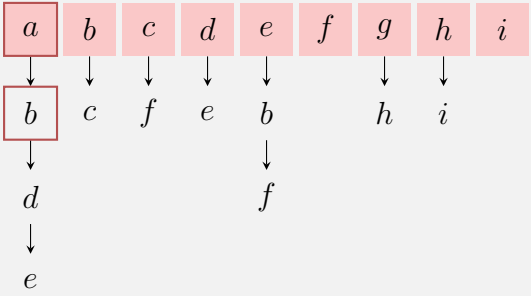


Tiefensuche

Verfolge zuerst Pfad in die Tiefe, bis nichts mehr besucht werden kann.

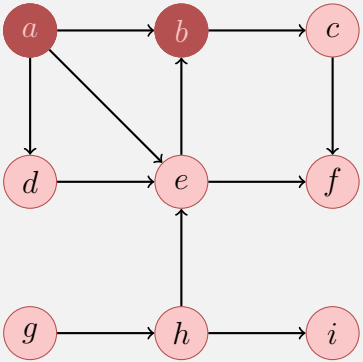


Adjazenzliste

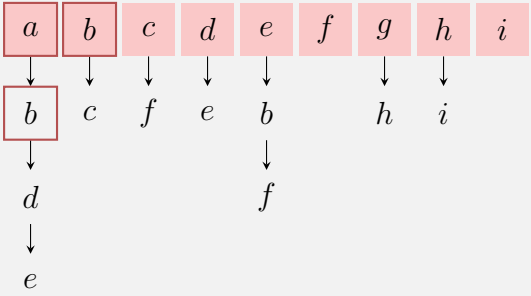


Tiefensuche

Verfolge zuerst Pfad in die Tiefe, bis nichts mehr besucht werden kann.

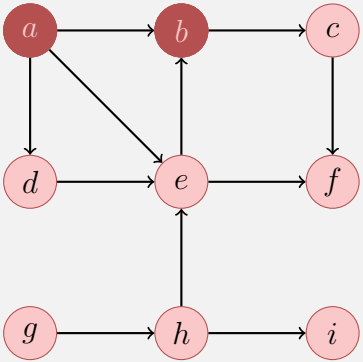


Adjazenzliste

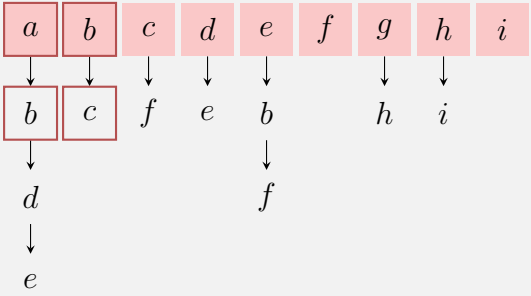


Tiefensuche

Verfolge zuerst Pfad in die Tiefe, bis nichts mehr besucht werden kann.

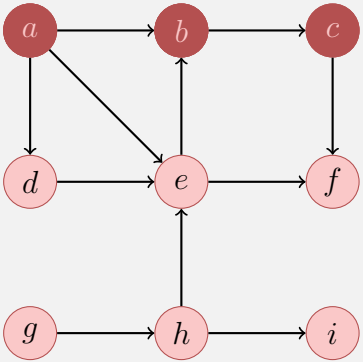


Adjazenzliste

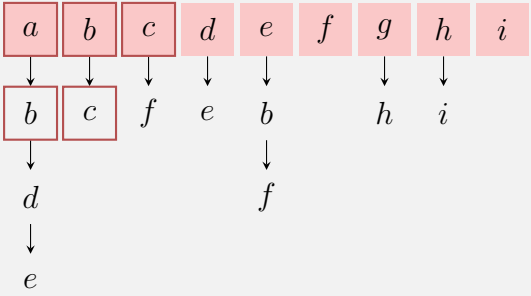


Tiefensuche

Verfolge zuerst Pfad in die Tiefe, bis nichts mehr besucht werden kann.

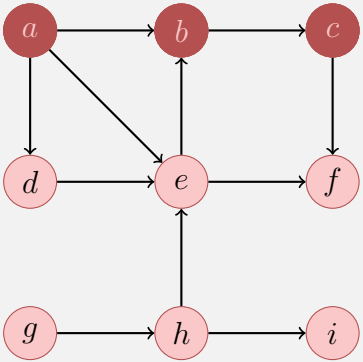


Adjazenzliste

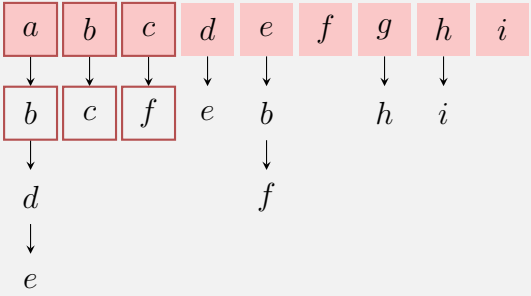


Tiefensuche

Verfolge zuerst Pfad in die Tiefe, bis nichts mehr besucht werden kann.

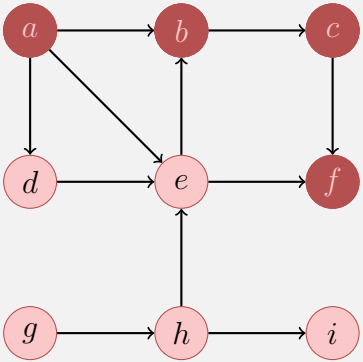


Adjazenzliste

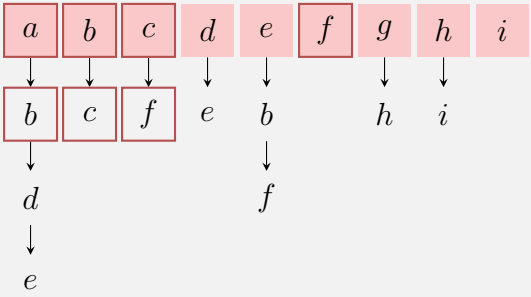


Tiefensuche

Verfolge zuerst Pfad in die Tiefe, bis nichts mehr besucht werden kann.

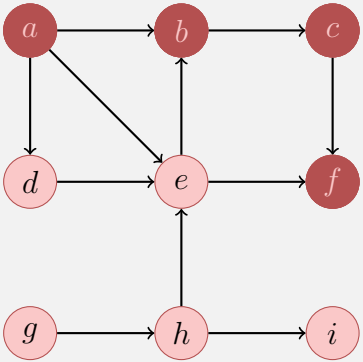


Adjazenzliste

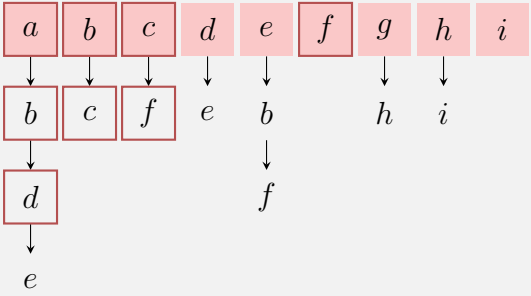


Tiefensuche

Verfolge zuerst Pfad in die Tiefe, bis nichts mehr besucht werden kann.

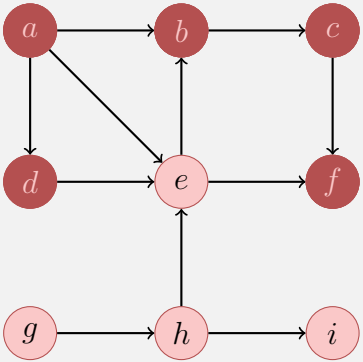


Adjazenzliste

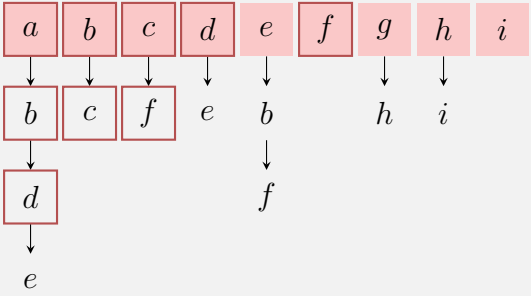


Tiefensuche

Verfolge zuerst Pfad in die Tiefe, bis nichts mehr besucht werden kann.

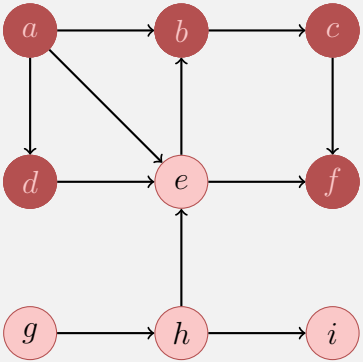


Adjazenzliste

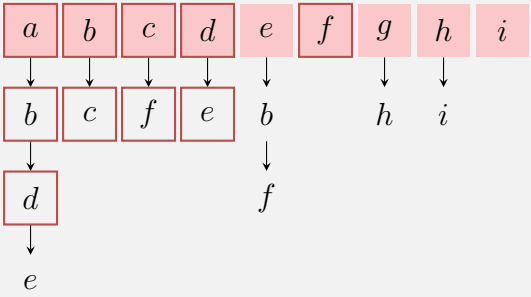


Tiefensuche

Verfolge zuerst Pfad in die Tiefe, bis nichts mehr besucht werden kann.

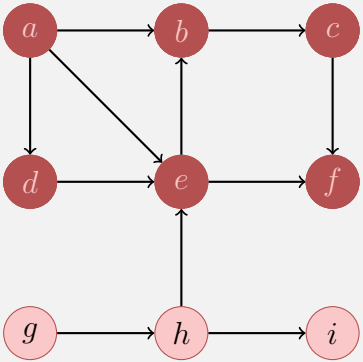


Adjazenzliste

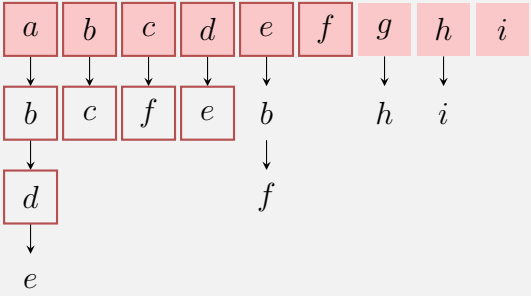


Tiefensuche

Verfolge zuerst Pfad in die Tiefe, bis nichts mehr besucht werden kann.

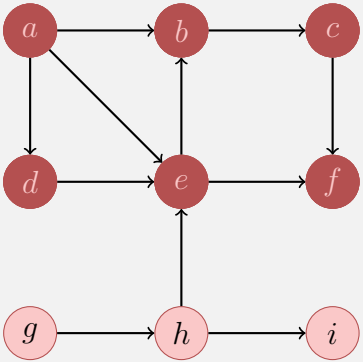


Adjazenzliste

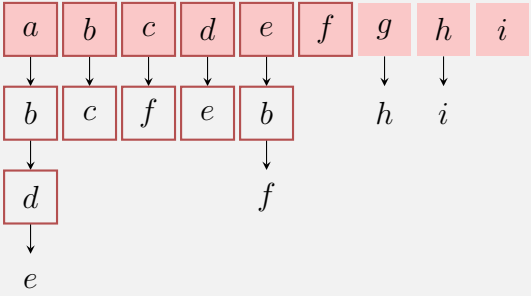


Tiefensuche

Verfolge zuerst Pfad in die Tiefe, bis nichts mehr besucht werden kann.

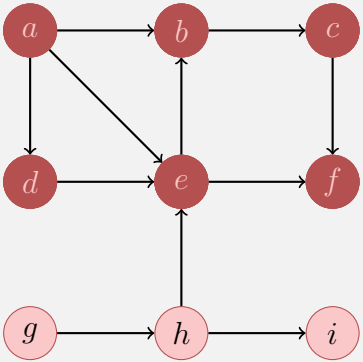


Adjazenzliste

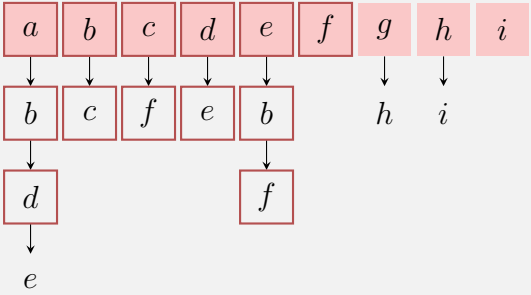


Tiefensuche

Verfolge zuerst Pfad in die Tiefe, bis nichts mehr besucht werden kann.

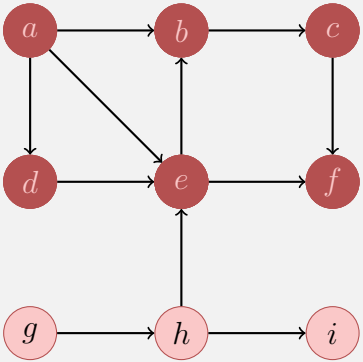


Adjazenzliste

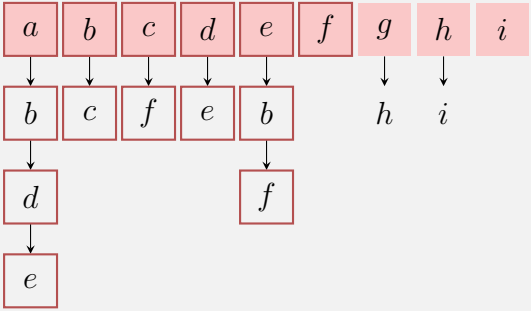


Tiefensuche

Verfolge zuerst Pfad in die Tiefe, bis nichts mehr besucht werden kann.

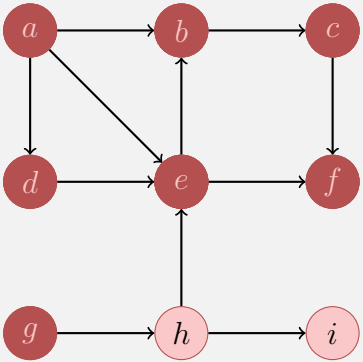


Adjazenzliste

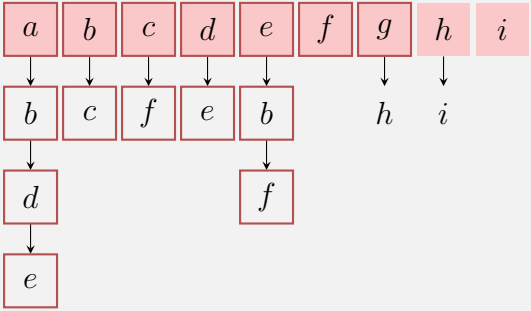


Tiefensuche

Verfolge zuerst Pfad in die Tiefe, bis nichts mehr besucht werden kann.

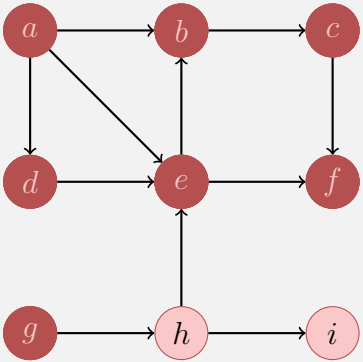


Adjazenzliste

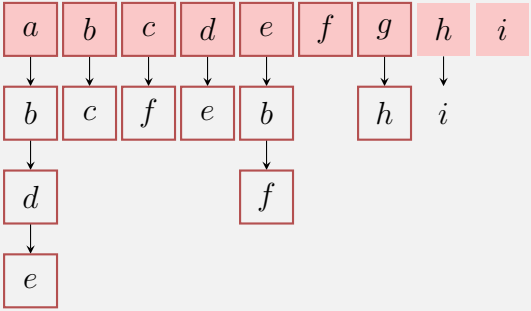


Tiefensuche

Verfolge zuerst Pfad in die Tiefe, bis nichts mehr besucht werden kann.

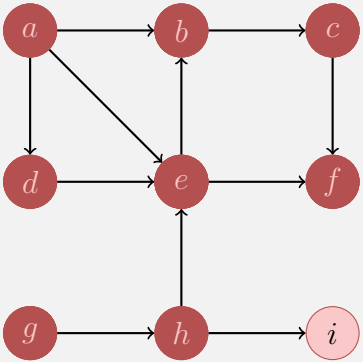


Adjazenzliste

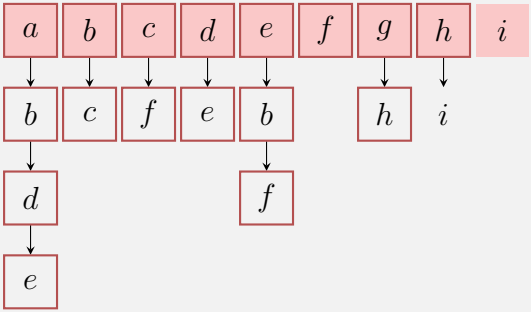


Tiefensuche

Verfolge zuerst Pfad in die Tiefe, bis nichts mehr besucht werden kann.

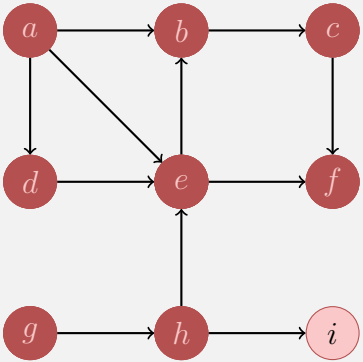


Adjazenzliste

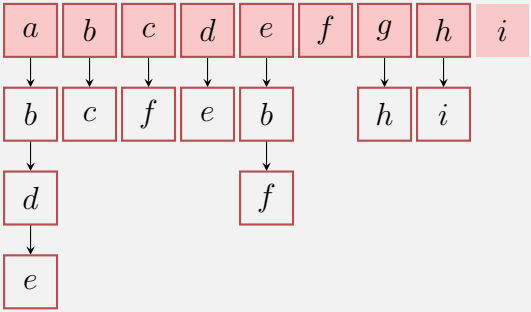


Tiefensuche

Verfolge zuerst Pfad in die Tiefe, bis nichts mehr besucht werden kann.

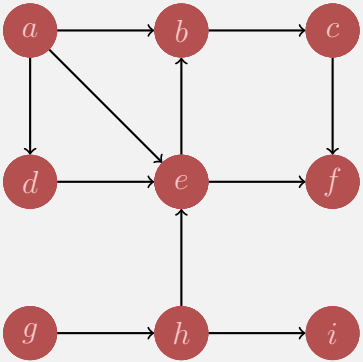


Adjazenzliste



Tiefensuche

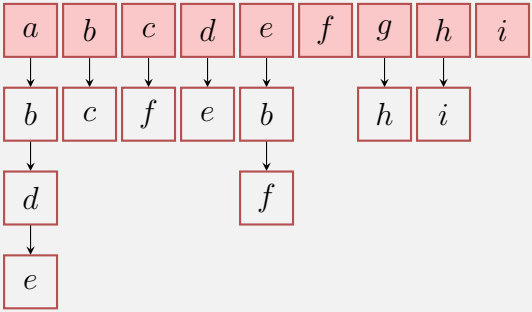
Verfolge zuerst Pfad in die Tiefe, bis nichts mehr besucht werden kann.



Reihenfolge

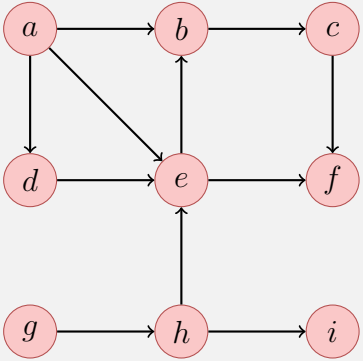
$a, b, c, f, d, e, g, h, i$

Adjazenzliste

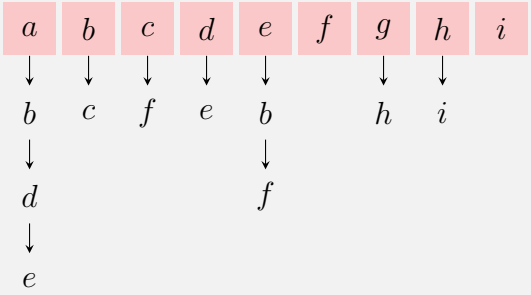


Breitensuche

Verfolge zuerst Pfad in die Breite, gehe dann in die Tiefe.

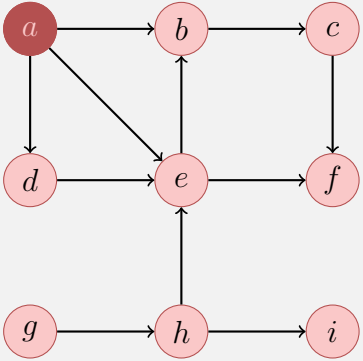


Adjazenzliste

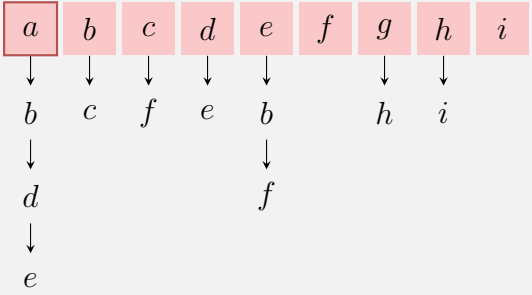


Breitensuche

Verfolge zuerst Pfad in die Breite, gehe dann in die Tiefe.

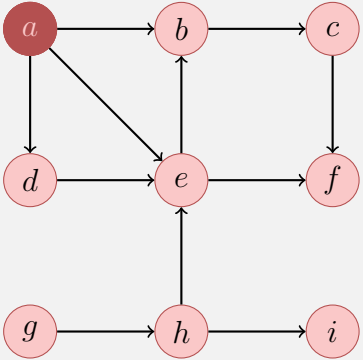


Adjazenzliste

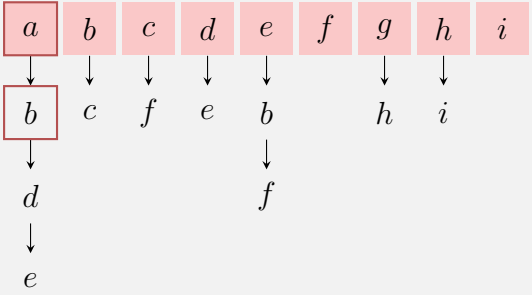


Breitensuche

Verfolge zuerst Pfad in die Breite, gehe dann in die Tiefe.

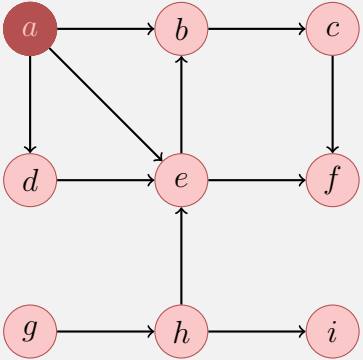


Adjazenzliste

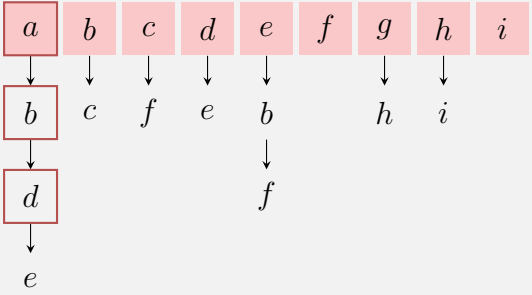


Breitensuche

Verfolge zuerst Pfad in die Breite, gehe dann in die Tiefe.

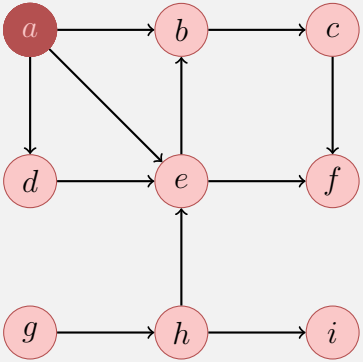


Adjazenzliste

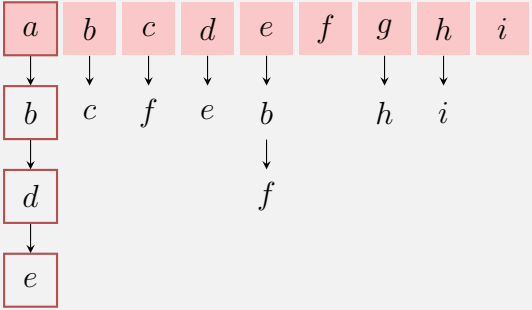


Breitensuche

Verfolge zuerst Pfad in die Breite, gehe dann in die Tiefe.

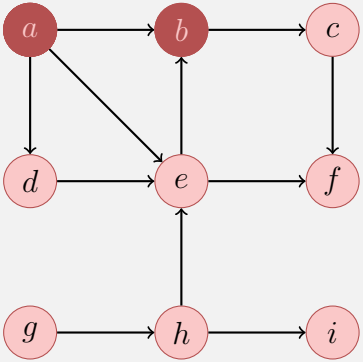


Adjazenzliste

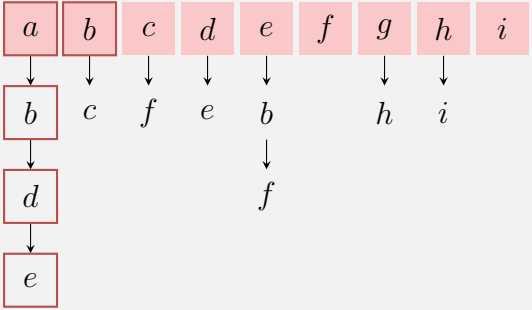


Breitensuche

Verfolge zuerst Pfad in die Breite, gehe dann in die Tiefe.

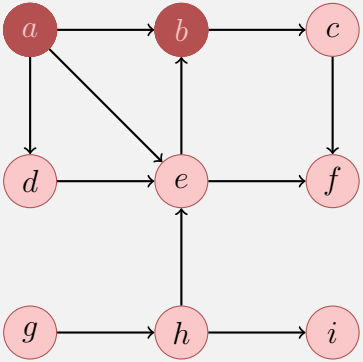


Adjazenzliste

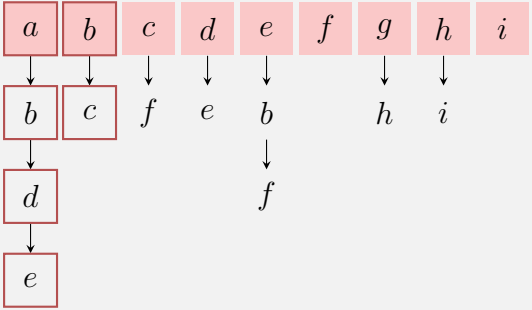


Breitensuche

Verfolge zuerst Pfad in die Breite, gehe dann in die Tiefe.

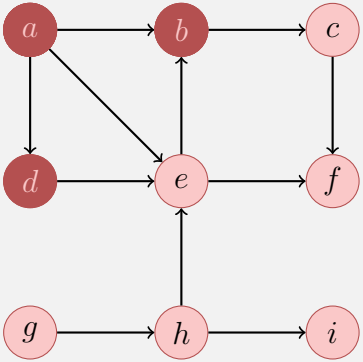


Adjazenzliste

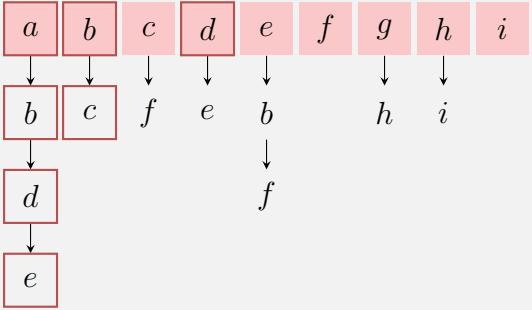


Breitensuche

Verfolge zuerst Pfad in die Breite, gehe dann in die Tiefe.

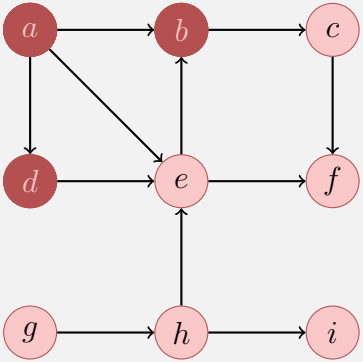


Adjazenzliste

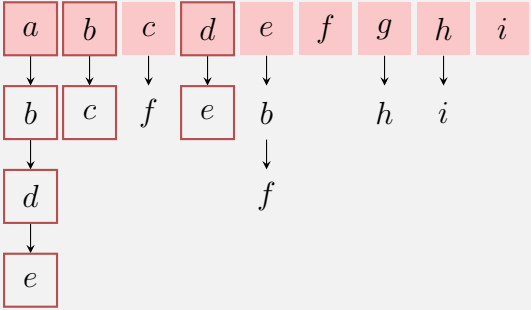


Breitensuche

Verfolge zuerst Pfad in die Breite, gehe dann in die Tiefe.

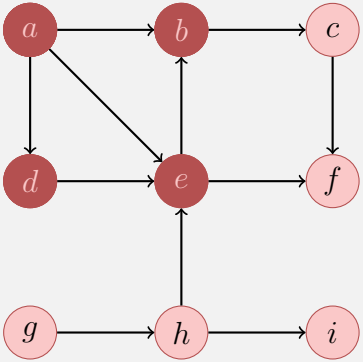


Adjazenzliste

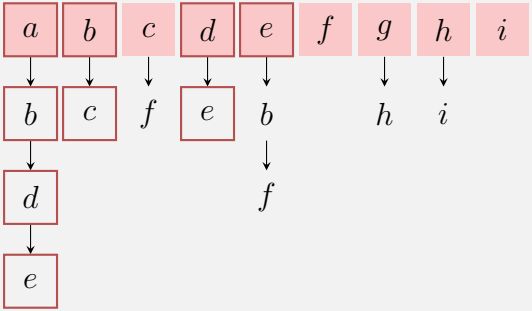


Breitensuche

Verfolge zuerst Pfad in die Breite, gehe dann in die Tiefe.

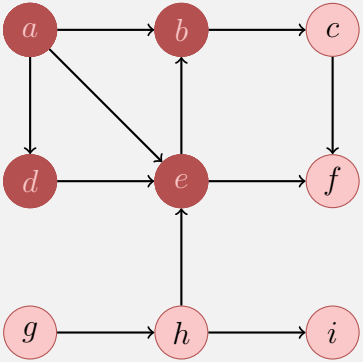


Adjazenzliste

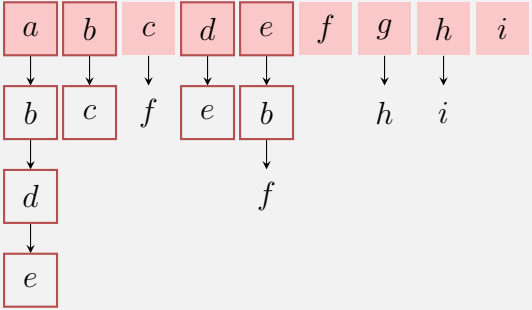


Breitensuche

Verfolge zuerst Pfad in die Breite, gehe dann in die Tiefe.

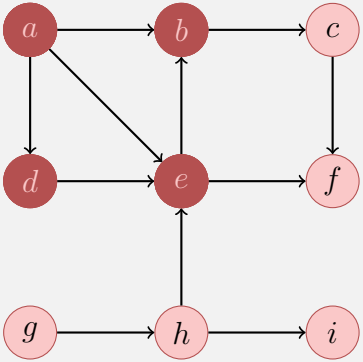


Adjazenzliste

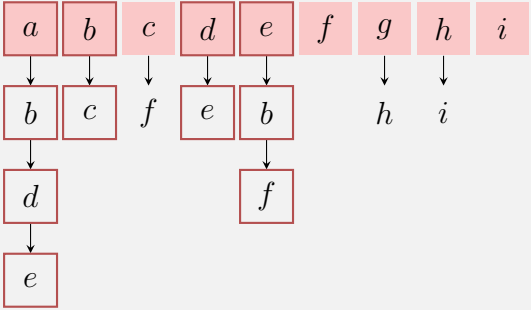


Breitensuche

Verfolge zuerst Pfad in die Breite, gehe dann in die Tiefe.

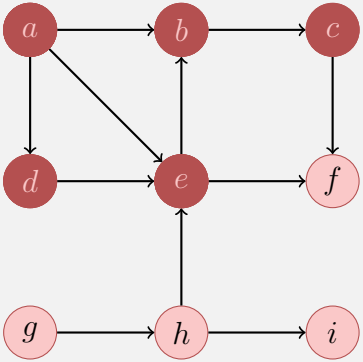


Adjazenzliste

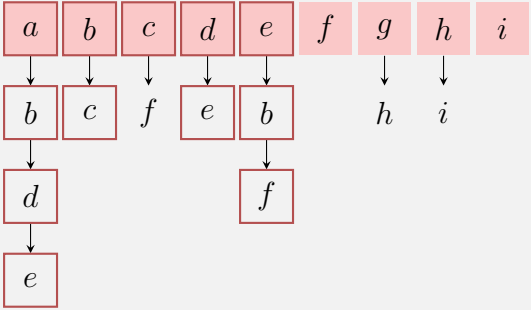


Breitensuche

Verfolge zuerst Pfad in die Breite, gehe dann in die Tiefe.

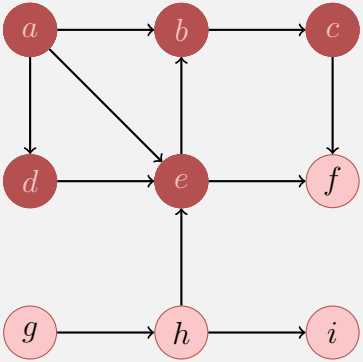


Adjazenzliste

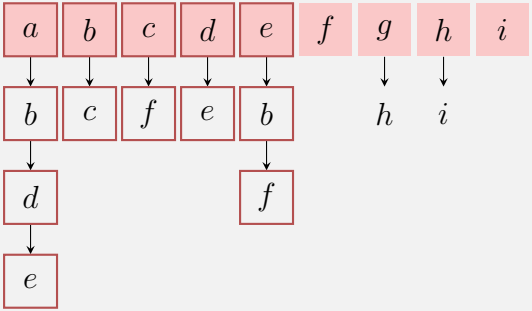


Breitensuche

Verfolge zuerst Pfad in die Breite, gehe dann in die Tiefe.

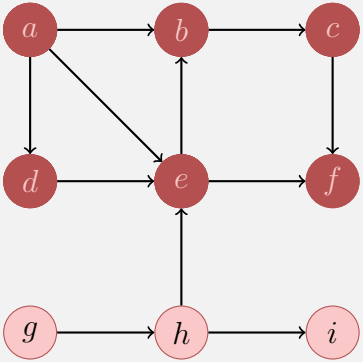


Adjazenzliste

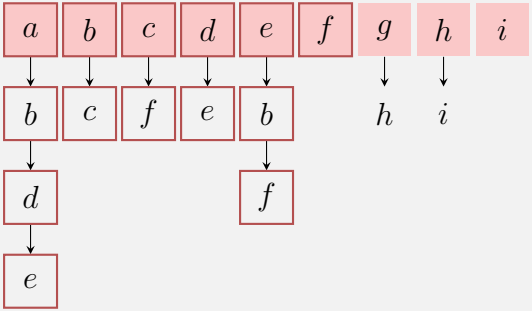


Breitensuche

Verfolge zuerst Pfad in die Breite, gehe dann in die Tiefe.

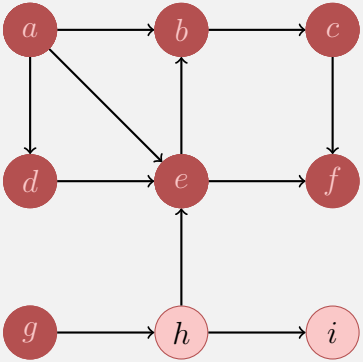


Adjazenzliste

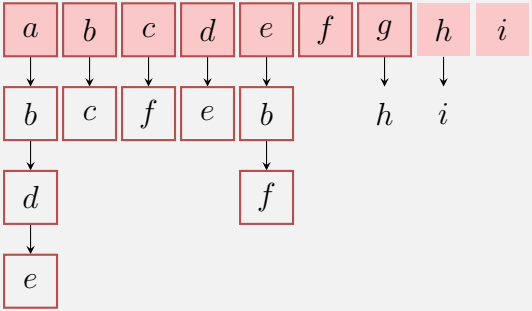


Breitensuche

Verfolge zuerst Pfad in die Breite, gehe dann in die Tiefe.

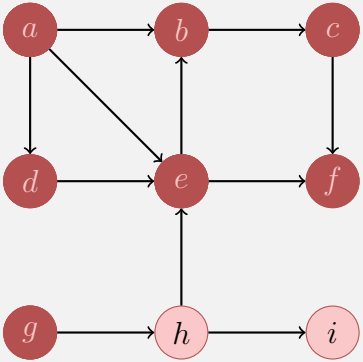


Adjazenzliste

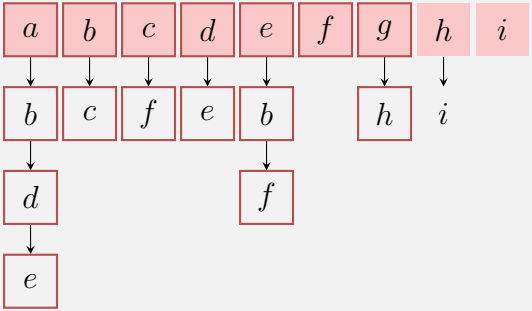


Breitensuche

Verfolge zuerst Pfad in die Breite, gehe dann in die Tiefe.

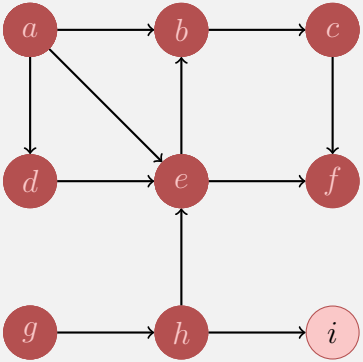


Adjazenzliste

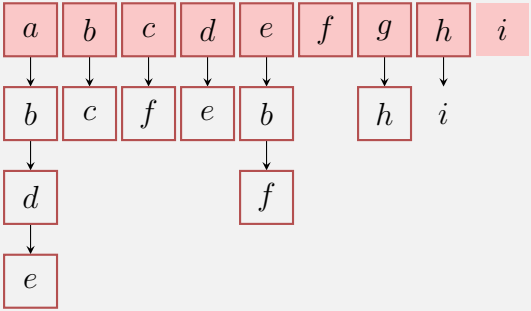


Breitensuche

Verfolge zuerst Pfad in die Breite, gehe dann in die Tiefe.

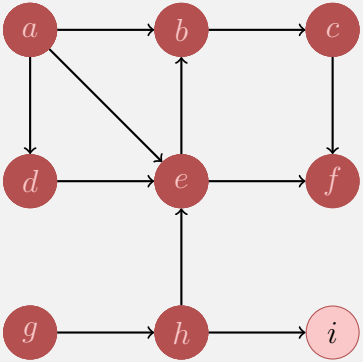


Adjazenzliste

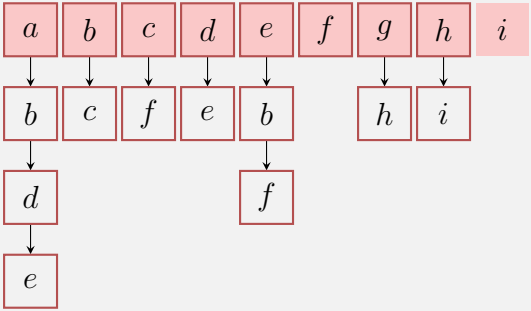


Breitensuche

Verfolge zuerst Pfad in die Breite, gehe dann in die Tiefe.

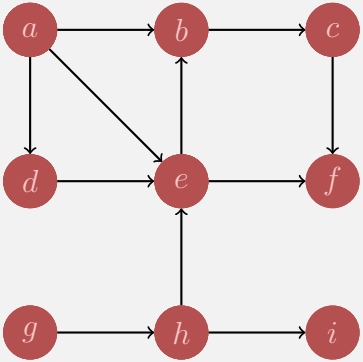


Adjazenzliste



Breitensuche

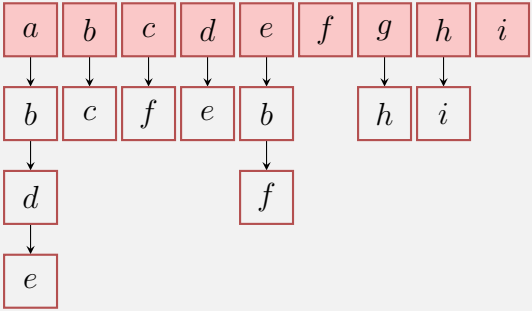
Verfolge zuerst Pfad in die Breite, gehe dann in die Tiefe.



Reihenfolge

$a, b, d, e, c, f, g, h, i$

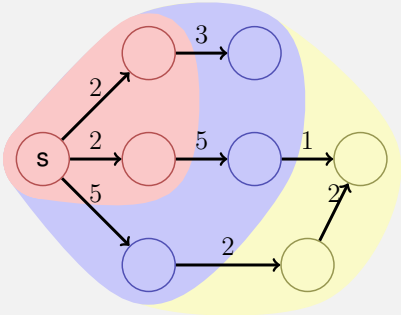
Adjazenzliste



Dijkstra's Kürzeste Wege Grundidee

Menge V aller Knoten wird unterteilt in

- die Menge M von Knoten, für die schon ein kürzester Weg von s bekannt ist
- die Menge $R = \bigcup_{v \in M} N^+(v) \setminus M$ von Knoten, für die kein kürzester Weg bekannt ist, die jedoch von M direkt erreichbar sind.
- die Menge $U = V \setminus (M \cup R)$ von Knoten, die noch nicht berücksichtigt wurden.

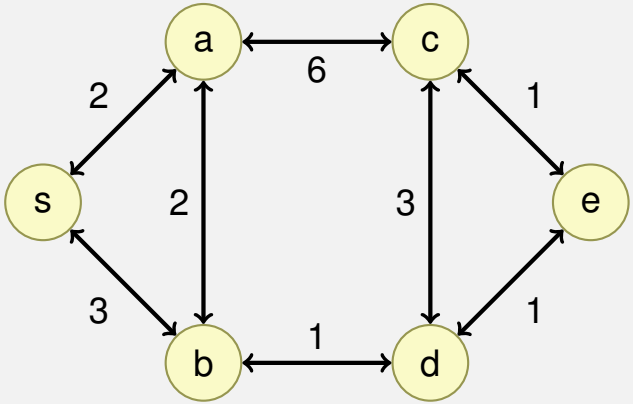


Algorithmus

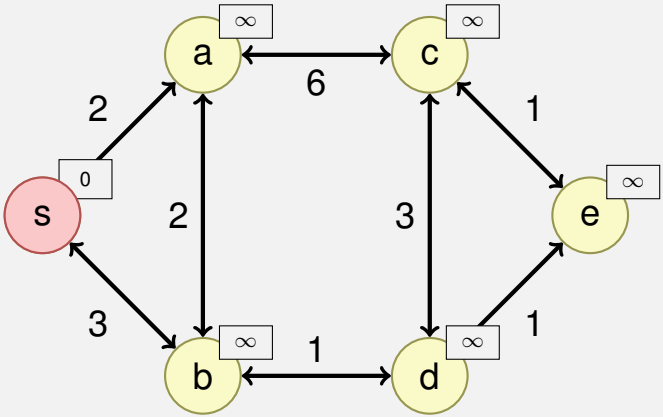
Initial: $\mathrm{PL}(n) \leftarrow \infty$ für alle Knoten.

- Setze $\mathrm{PL}(s) \leftarrow 0$
- Starte mit $M = \{s\}$. Setze $k \leftarrow s$.
- Solange ein neuer Knoten k hinzukommt und dieser nicht der Zielknoten ist
  1. Für jeden Nachbarknoten n von k:
     - Berechne Pfadlänge x nach n über k
     - Wenn $\mathrm{PL}(n) = \infty$, so nimm n zu R hinzu
     - Ist $x < \mathrm{PL}(n) < \infty$, so setze $\mathrm{PL}(n) \leftarrow x$ und passe R an.
  2. Wähle als neuen Knoten k den mit kleinster Pfadlänge in R.

Beispiel



Beispiel

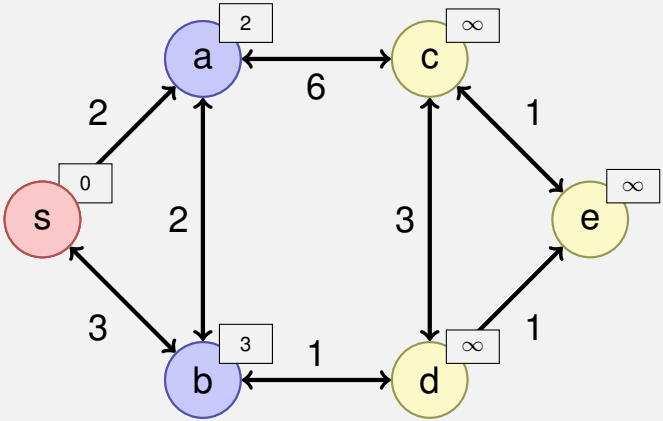


$$M = \{s\}$$

$$R = \{\}$$

$$U = \{a, b, c, d, e\}$$

Beispiel

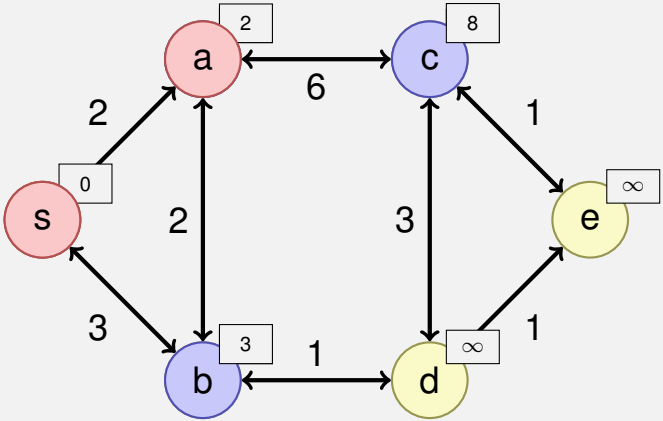


$$M = \{s\}$$

$$R = \{a, b\}$$

$$U = \{c, d, e\}$$

Beispiel

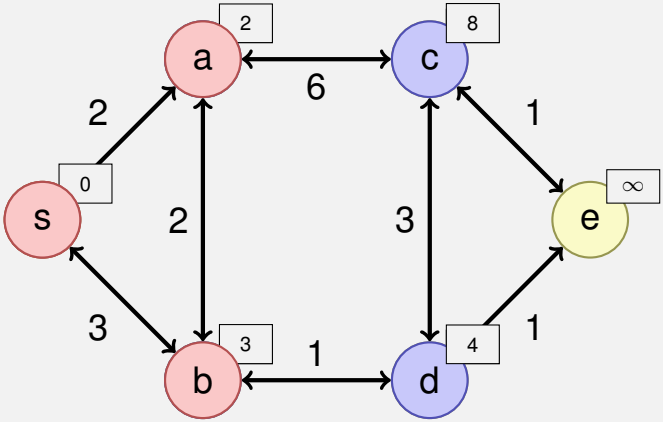


$$M = \{s, a\}$$

$$R = \{b, c\}$$

$$U = \{d, e\}$$

Beispiel

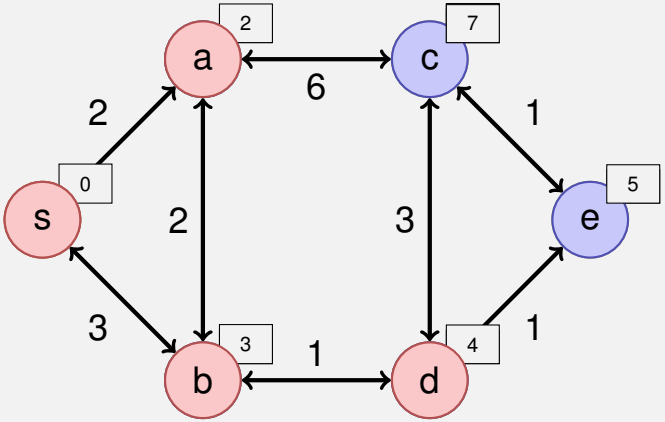


$$M = \{s, a, b\}$$

$$R = \{c, d\}$$

$$U = \{e\}$$

Beispiel

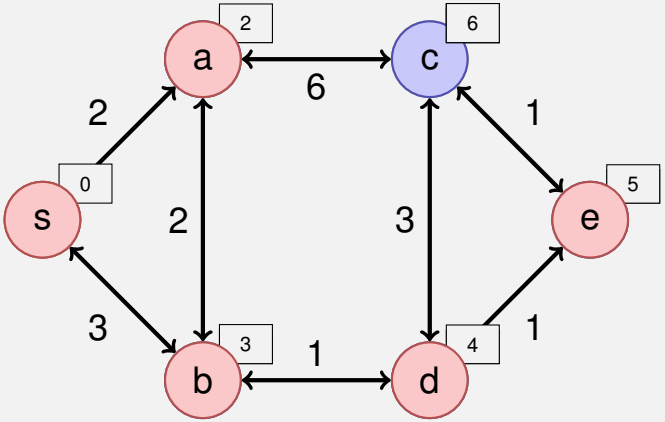


$$M = \{s, a, b, d\}$$

$$R = \{c, e\}$$

$$U = \{\}$$

Beispiel

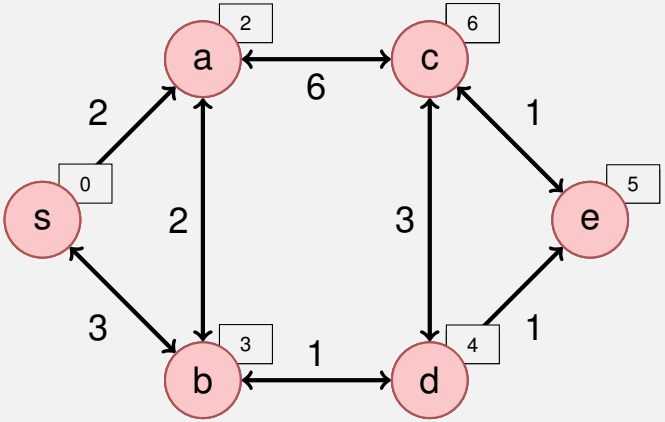


$$M = \{s, a, b, d, e\}$$

$$R = \{c\}$$

$$U = \{\}$$

Beispiel



$$M = \{s, a, b, d, e, c\}$$

$$R = \{\}$$

$$U = \{\}$$

Zur Implementation

Benötigte Operationen

- ExtractMin (über R)
- Insert (Hinzunehmen zu R)
- DecreaseKey (Update in R)

Datenstruktur? .

Zur Implementation

Benötigte Operationen

- ExtractMin (über R)
- Insert (Hinzunehmen zu R)
- DecreaseKey (Update in R)

Datenstruktur: MinHeap.

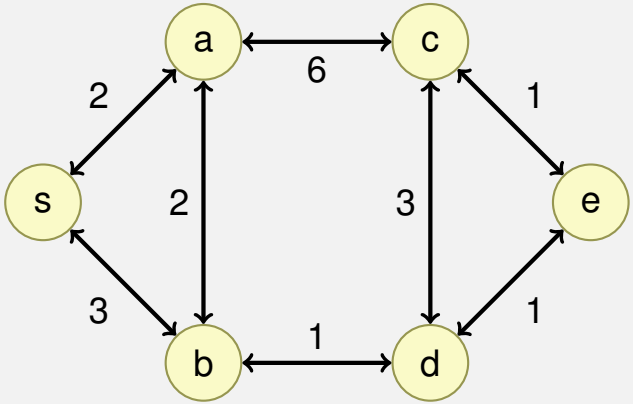
Laufzeit

- $|V|\times$ ExtractMin: $\mathcal{O}(|V| \log |V|)$
- $|E|\times$ Insert oder DecreaseKey: $\mathcal{O}(|E| \log |V|)$
- $1\times$ Init: $\mathcal{O}(|V|)$
- Insgesamt: $\mathcal{O}(|E| \log |V|)$.

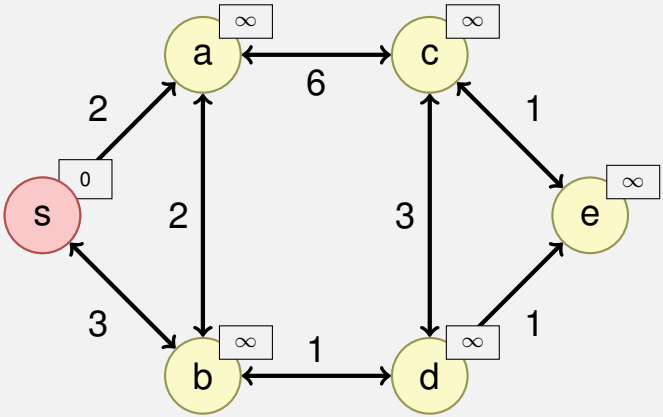
Kürzesten Weg Rekonstruieren

- Beim Updateschritt im obigen Algorithmus jeweils besten Vorgänger merken, an Knoten oder in separater Datenstruktur.
- Besten Pfad rekonstruieren durch Rückwärtslaufen der besten Kanten

Beispiel



Beispiel

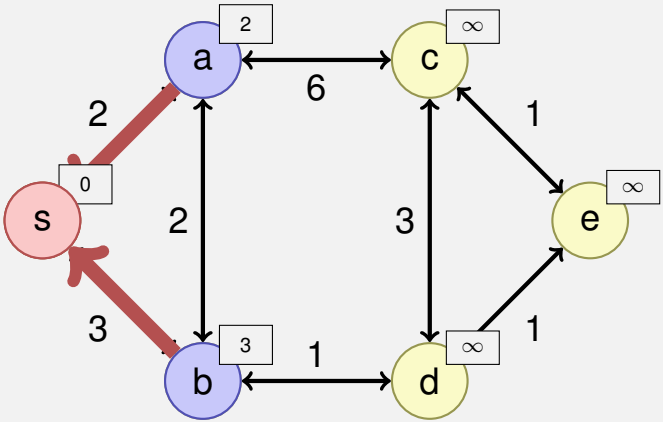


$$M = \{s\}$$

$$R = \{\}$$

$$U = \{a, b, c, d, e\}$$

Beispiel

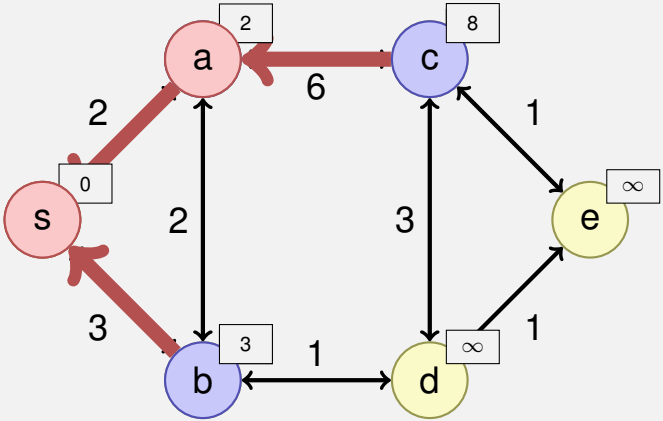


$$M = \{s\}$$

$$R = \{a, b\}$$

$$U = \{c, d, e\}$$

Beispiel

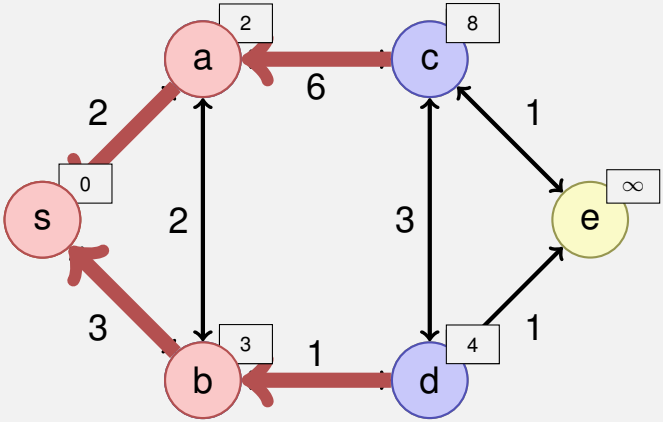


$$M = \{s, a\}$$

$$R = \{b, c\}$$

$$U = \{d, e\}$$

Beispiel

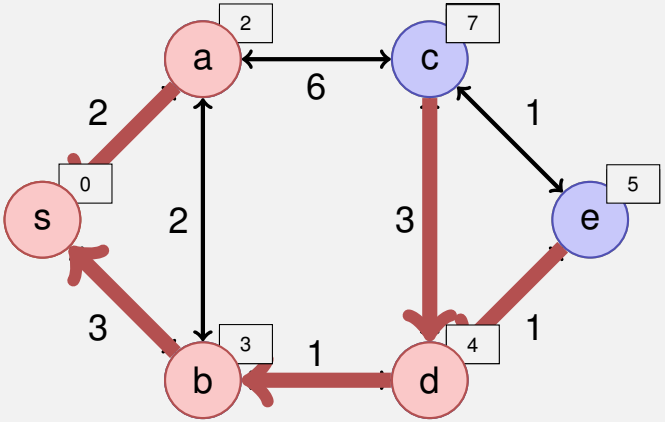


$$M = \{s, a, b\}$$

$$R = \{c, d\}$$

$$U = \{e\}$$

Beispiel

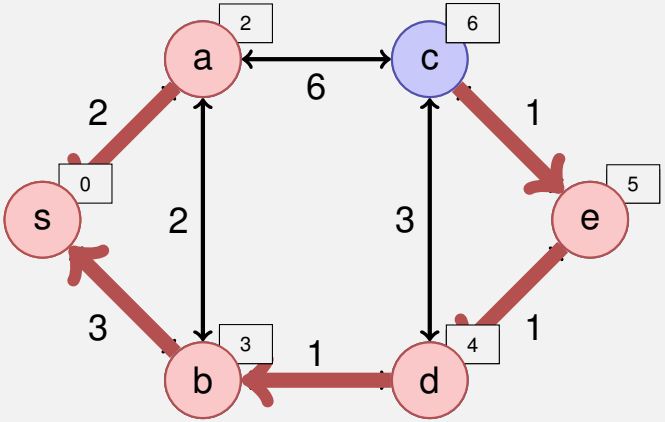


$$M = \{s, a, b, d\}$$

$$R = \{c, e\}$$

$$U = \{\}$$

Beispiel

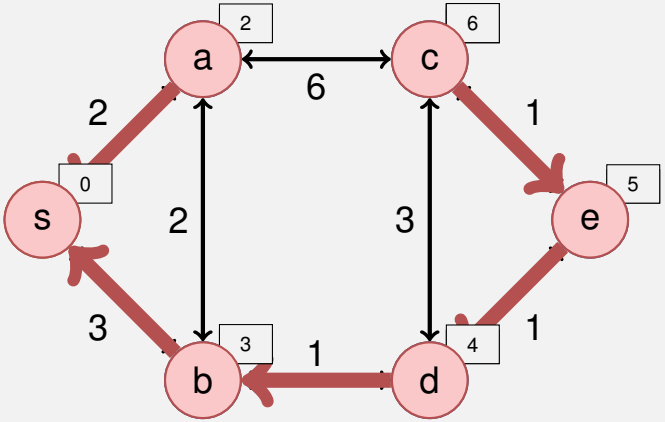


$$M = \{s, a, b, d, e\}$$

$$R = \{c\}$$

$$U = \{\}$$

Beispiel



$$M = \{s, a, b, d, e, c\}$$

$$R = \{\}$$

$$U = \{\}$$

Fragen oder Anregungen?