Informatik II

Übung 8

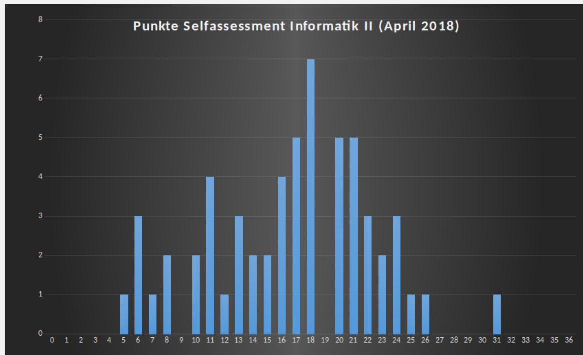
Andreas Bärtschi, Andreea Ciuprina, Felix Friedrich, Patrick Gruntz, Hermann Lehner, Max Rossmannek, Chris Wendler

FS 2018

Heutiges Programm

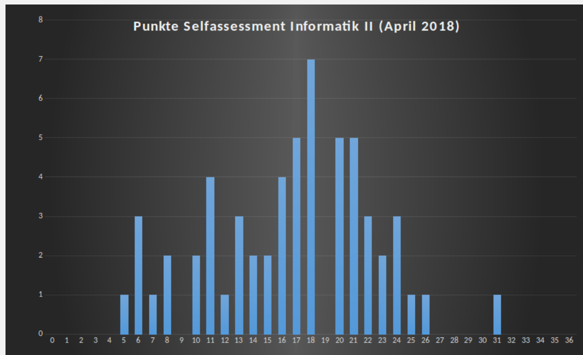
1. Self-Assessment
2. Wiederholung Vorlesung: Adjazenzlisten
3. Breitensuche BFS
4. In-Class-Exercise

Letzte Woche: Self-Assessment



Nr	Schwierigkeit	Trennschärfe
1	58% der Punkte	0.18
2	36% der Punkte	0.23
3	3% der Punkte	0.08

Letzte Woche: Self-Assessment



Nr	Schwierigkeit	Trennschärfe
1	58% der Punkte	0.18
2	36% der Punkte	0.23
3	3% der Punkte	0.08

Zur Einordung: 20 Punkte wären eine genügende Leistung – in der Basisprüfung würde in Aufgabe 3 aber ein Punkteschema mit mehr Teilpunkten verwendet.

Adjazenzliste

```
class Graph { // G = (V,E) as adjacency list
        private int V; // number of vertices
        private ArrayList<LinkedList<Integer>> adj; // adj. list
        // Constructor
        public Graph(int n) {
                V = n;
                adj = new ArrayList<LinkedList<Integer>>(V);
                for (int i=0; i<V; ++i)
                        adj.add(i,new LinkedList<Integer>());
        }
        // Edge adder method
        public void addEdge(int u,int v) {
                adj.get(u).add(v);
        }
}
```

Adjazenzliste

Eigenschaften:

`ArrayList`
: Elementzugriff in konstanter Zeit.

`LinkedList`
: Element hinzufügen in konstanter Zeit.
: Über die ganze Liste iterieren in linearer Zeit.

Adjazenzliste

Eigenschaften:

`ArrayList`
: Elementzugriff in konstanter Zeit.

`LinkedList`
: Element hinzufügen in konstanter Zeit.
: Über die ganze Liste iterieren in linearer Zeit.

- `addEdge(u,v) = adj.get(u).add(v)` runs in constant time $\mathcal{O}(1)$.

Adjazenzliste

Eigenschaften:

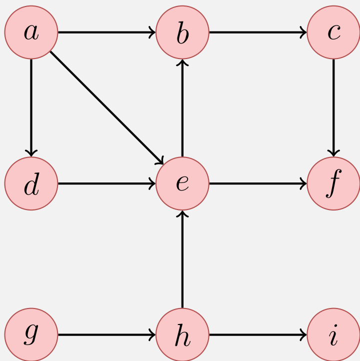
`ArrayList`
: Elementzugriff in konstanter Zeit.

`LinkedList`
: Element hinzufügen in konstanter Zeit.
: Über die ganze Liste iterieren in linearer Zeit.

- `addEdge(u,v) = adj.get(u).add(v)` runs in constant time $\mathcal{O}(1)$.
- `for (int v : adj.get(u))` runs in time $\mathcal{O}(\deg^+(u))$.

Breitensuche BFS

BFS von a aus:

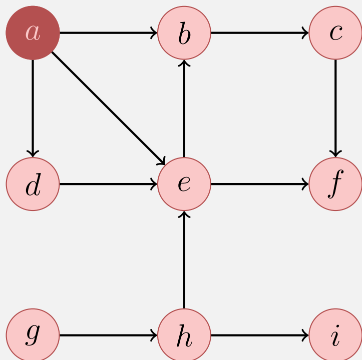


BFS-Baum: Distanzen und Vorgänger

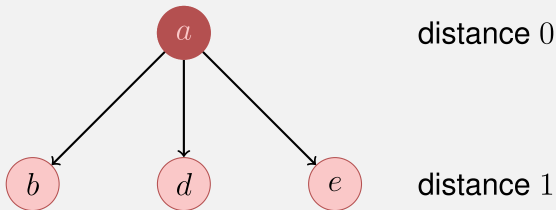
distance 0

Breitensuche BFS

BFS von a aus:

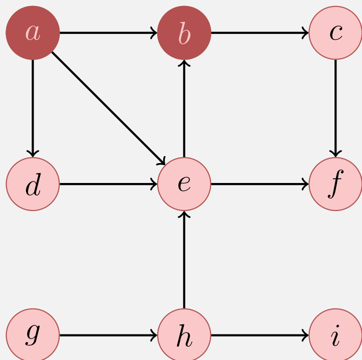


BFS-Baum: Distanzen und Vorgänger

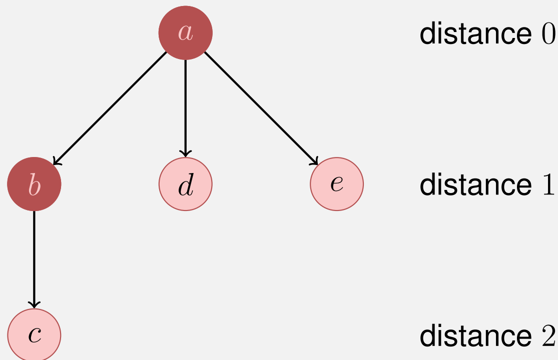


Breitensuche BFS

BFS von a aus:

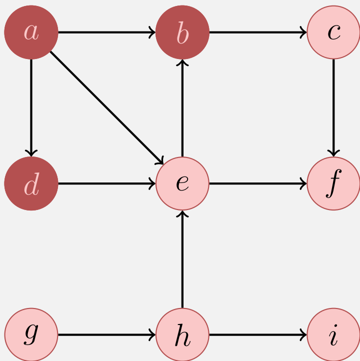


BFS-Baum: Distanzen und Vorgänger

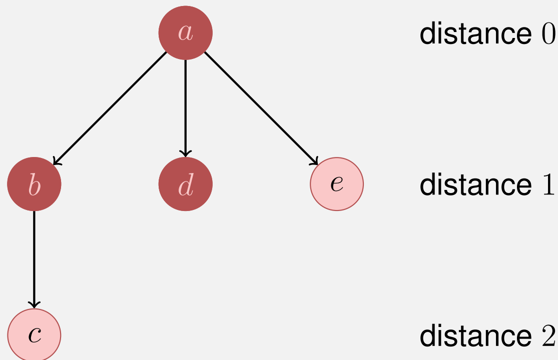


Breitensuche BFS

BFS von a aus:

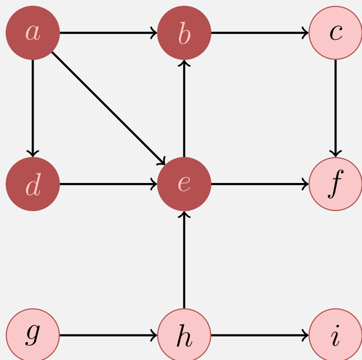


BFS-Baum: Distanzen und Vorgänger

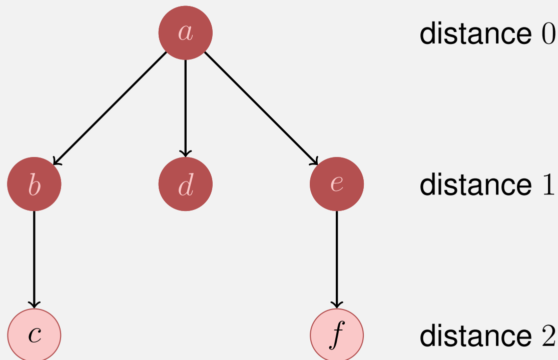


Breitensuche BFS

BFS von a aus:

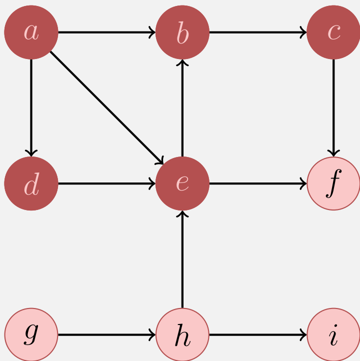


BFS-Baum: Distanzen und Vorgänger

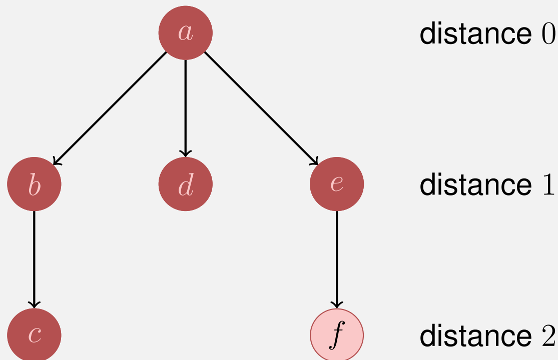


Breitensuche BFS

BFS von a aus:

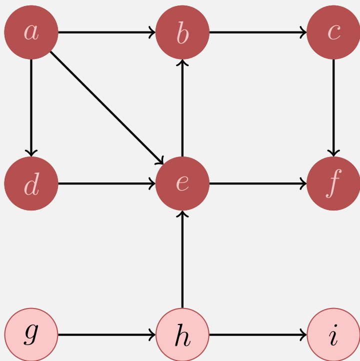


BFS-Baum: Distanzen und Vorgänger

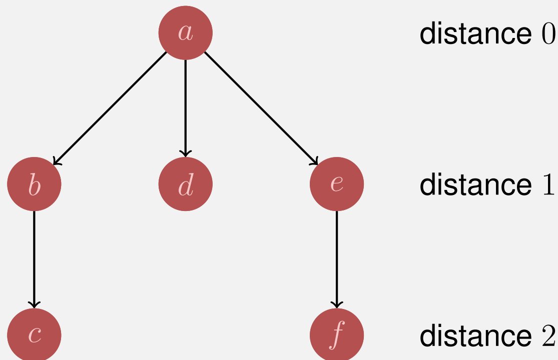


Breitensuche BFS

BFS von a aus:



BFS-Baum: Distanzen und Vorgänger



In-Class-Exercises: Wegplanung

Aufgabe: Gegeben sei

- ein gerichteter, ungewichteter Graph $G = (V, E)$, gespeichert als Adjazenzliste,
- und ein spezieller Knoten $t \in V$ (z.Bsp. ein Notausgang).

Entwerfen Sie einen Algorithmus,

- welcher für jeden Knoten $u \in V$ eine Ausgangskante in Richtung eines kürzesten Pfades nach t angibt.
- und eine Laufzeit von $\mathcal{O}(|V| + |E|)$ hat.

In-Class-Exercises: Wegplanung

Lösung:

1. Erstelle eine Kopie des Graphen mit umgerichteten Kanten:
   $G^T = (V, E^T)$, mit $E^T = \{(v, u) \mid (u, v) \in E\}$.
   Laufzeit: $\mathcal{O}(|V| + |E|)$.
2. Starte eine Breitensuche von G^T ausgehend von t,
   und speichere alle BFS-Baum-Kanten.
   Laufzeit: $\mathcal{O}(|V| + |E^T|) = \mathcal{O}(|V| + |E|)$.
3. Weise die gefundenen Kanten (in umgekehrter Richtung)
   den entdeckten Knoten zu.
   Laufzeit: $\mathcal{O}(|V|)$.

Fragen oder Anregungen?