# Informatik II

## Übung 8

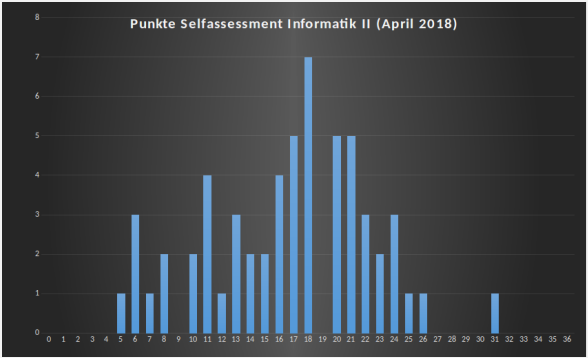
Andreas Bärtschi, Andreea Ciuprina, Felix Friedrich, Patrick Gruntz,
Hermann Lehner, Max Rossmannek, Chris Wendler

FS 2018

# Program Today

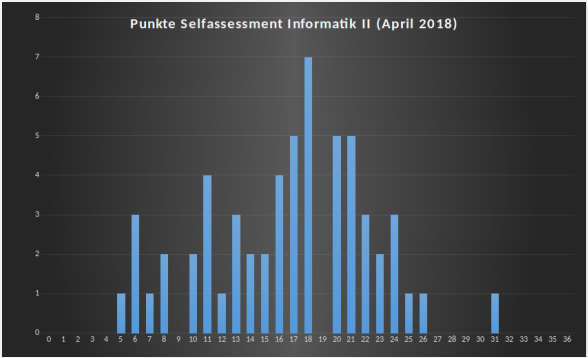
1. Self-Assessment
2. Repetition Lectures: Adjacency Lists
3. Breadth-First-Search BFS
4. In-Class-Exercise

# Last week: Self-Assessment




| Nr | Difficulty | Nr-Total-Corr. |
|---|---|---|
| 1 | 58% of points | 0.18 |
| 2 | 36% of points | 0.23 |
| 3 | 3% of points | 0.08 |

# Last week: Self-Assessment




| Nr | Difficulty | Nr-Total-Corr. |
|---|---|---|
| 1 | 58% of points | 0.18 |
| 2 | 36% of points | 0.23 |
| 3 | 3% of points | 0.08 |

For comparison: 20 points would correspond to a passing grade – in the first year exam however, Exercise 3 would be graded with a more fine-grained marking scheme.

# Adjacency List

```
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);
        }
}
```

# Adjacency List

**Properties:**

`ArrayList`
: Get element in constant time.

`LinkedList`
: Add element in constant time.
: Iterate over whole list in linear time.

# Adjacency List

**Properties:**

`ArrayList`
: Get element in constant time.

`LinkedList`
: Add element in constant time.
Iterate over whole list in linear time.

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

# Adjacency List

**Properties:**

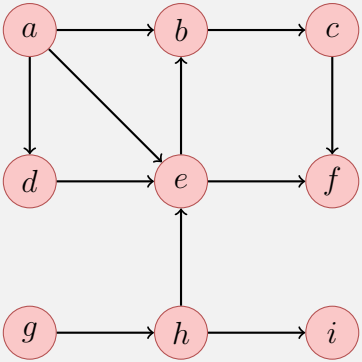
`ArrayList`
: Get element in constant time.

`LinkedList`
: Add element in constant time.
Iterate over whole list in linear time.

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

# Breadth-First-Search BFS

BFS starting from $a$:

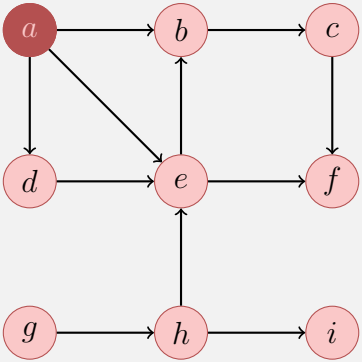



BFS-Tree: Distances and Parents

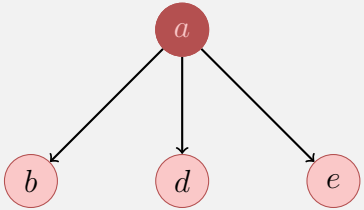
$a$ distance $0$

# Breadth-First-Search BFS

BFS starting from $a$:




BFS-Tree: Distances and Parents

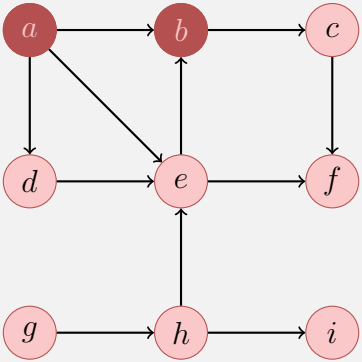



distance $0$

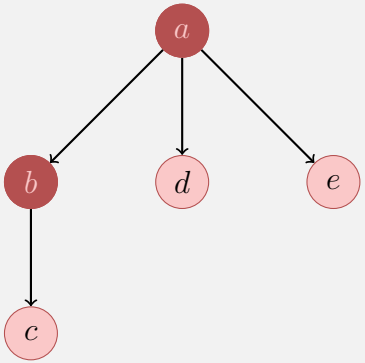
distance $1$

# Breadth-First-Search BFS

BFS starting from $a$:




BFS-Tree: Distances and Parents



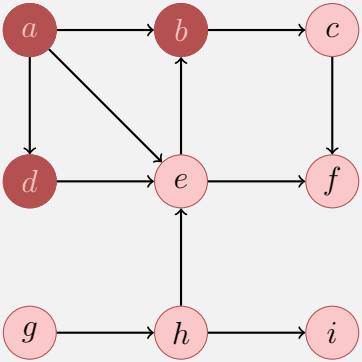

distance $0$

distance $1$

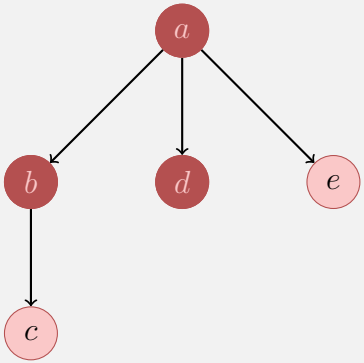
distance $2$

# Breadth-First-Search BFS

BFS starting from $a$:




BFS-Tree: Distances and Parents



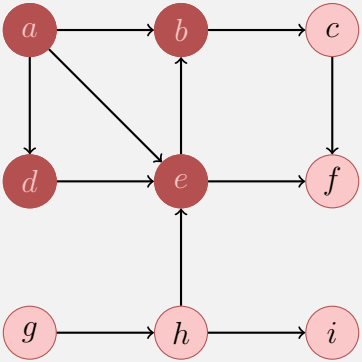

distance $0$

distance $1$

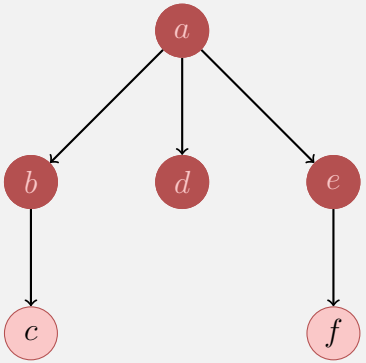
distance $2$

# Breadth-First-Search BFS

BFS starting from $a$:




BFS-Tree: Distances and Parents



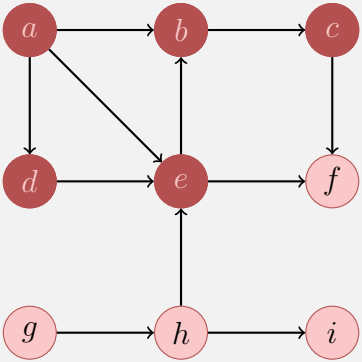

distance $0$

distance $1$

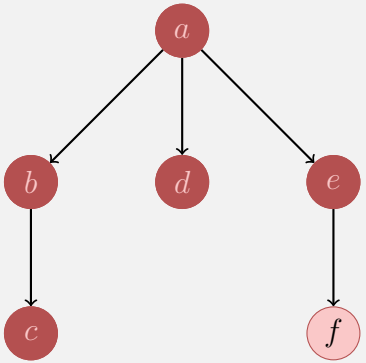
distance $2$

# Breadth-First-Search BFS

BFS starting from $a$:




BFS-Tree: Distances and Parents



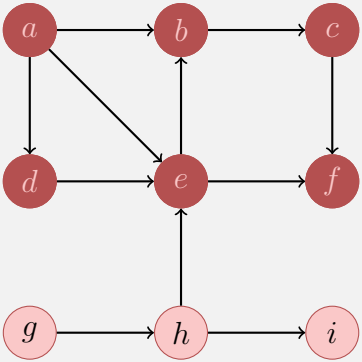

distance $0$

distance $1$

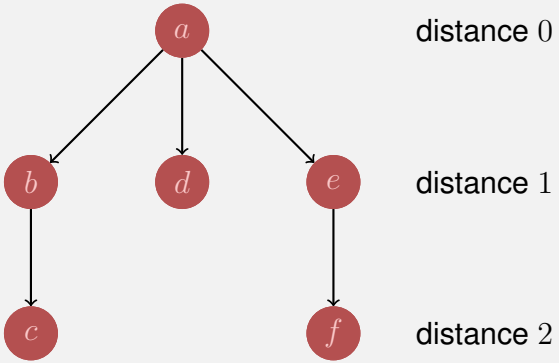
distance $2$

# Breadth-First-Search BFS

BFS starting from $a$:




BFS-Tree: Distances and Parents



# In-Class-Exercises: Route planning

**Exercise:** You are given

- a directed, unweighted Graph $G = (V, E)$, represented by an adjacency list,
- and a designated node $t \in V$ (e.g., an emergency exit).

Design an algorithm,

- which computes for each node $u \in V$ an outgoing edge in direction of a shortest path to $t$.
- and has a running time of $\mathcal{O}(|V| + |E|)$.

# In-Class-Exercises: Route planning

**Solution:**

1. Make a copy of the graph with edges having reverse direction: $G^T = (V, E^T)$, where $E^T = \{(v, u) \mid (u, v) \in E\}$.
   Running time: $\mathcal{O}(|V| + |E|)$.
2. Start a breadth-first search of $G^T$, starting from $t$,
   and store all edges of the BFS-Tree.
   Running time: $\mathcal{O}(|V| + |E^T|) = \mathcal{O}(|V| + |E|)$.
3. Assign the stored edges (in reverse direction)
   to the discovered nodes. Running time: $\mathcal{O}(|V|)$.

# Questions / Suggestions?