# Graph-Scan (Alg. 3.7)

Sucht Zusammenhangskomponenten vom Graphen ab, erzeugt dabei einen Spannbaum. „Eigentlich" 2 Algorithmen:

(1) Breitensuche (BFS) Breadth First Search


1.: Startknoten s

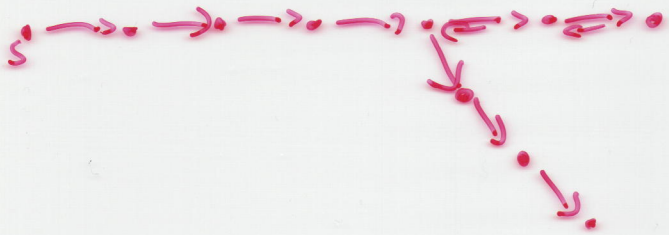
2.: Alle Nachbarn von s (Distanz 1)

3.: Aller mit Distanz 2

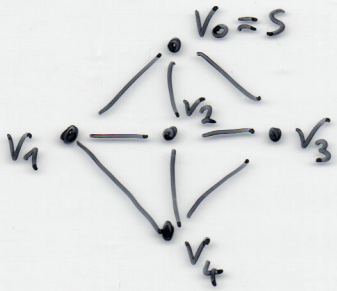
⋮

(2) Tiefensuche (DFS) Depth First Search

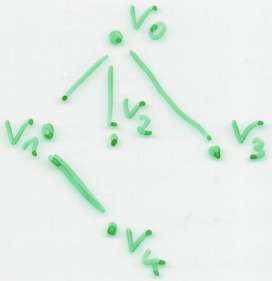
- Immer „möglichst weit" von s weg, zurück bei Sackgasse




## Beispiel



## BFS




R: $v_0$

$v_0\ v_1$

$v_0\ v_1\ v_2$

$v_0\ v_1\ v_2\ v_3$

$v_1\ v_2\ v_3$

$v_1\ v_2\ v_3\ v_4$

$v_2\ v_3\ v_4$

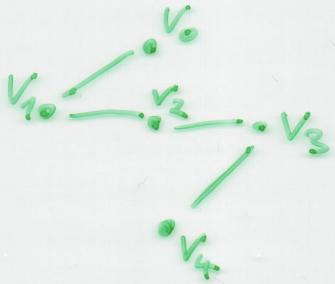
$v_3\ v_4$

$v_4$

$\varnothing$

BFS: Typischerweise stark verästelt (hohe Knotengrade), aber kurze Wege

## DFS




R: $v_0$

$v_1\ v_0$

$v_2\ v_1\ v_0$

$v_3\ v_2\ v_1\ v_0$

$v_4\ v_3\ v_2\ v_1\ v_0$

$v_3\ v_2\ v_1\ v_0$

$v_2\ v_1\ v_0$

$v_1\ v_0$

$v_0$

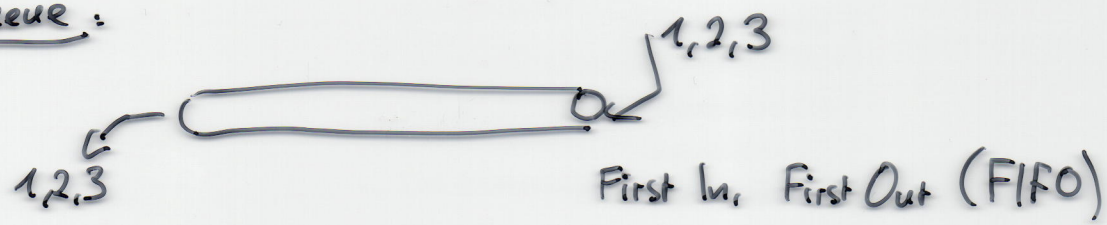
$\varnothing$

DFS: Oft geringe Knotengrade, lange Wege

# Abstrakte Datentypen (ADTs)

Idee: Spezifiziere, <u>was</u> eine Datenstruktur leisten muss, aber <u>nicht, wie</u>.

**Queue:**




**Interface:**

QUEUE e:

empty: -> QUEUE

isEmpty: QUEUE -> Bool

head: QUEUE -> e

dequeue: QUEUE -> QUEUE

enqueue: e x QUEUE -> QUEUE

**Axiome**

q: QUEUE

x: e

(1) isEmpty(empty()) = True

" (enqueue(x, q)) = False

(2) head(empty()) = ERROR

" (enqueue(x, q)) = IF isEmpty(q) THEN x ELSE head(q)

(3) dequeue(empty()) = ERROR

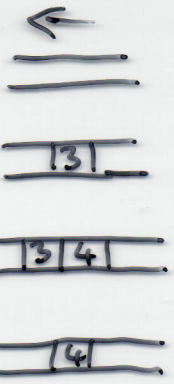
" (enqueue(x, q)) = IF isEmpty(q) THEN q
ELSE enqueue(x, dequeue(q))

1. $q \leftarrow empty()$

2. $q \leftarrow enqueue(3, q)$

3. $q \leftarrow enqueue(4, q)$

4. $q \leftarrow dequeue(q)$



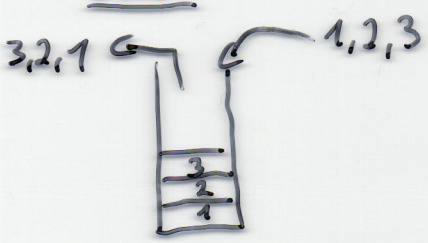

$dequeue(enqueue(4, enqueue(3, empty)))$

1. $empty$
2. $enqueue(3, empty)$
3. $enqueue(4, enqueue(3, empty))$
4. $dequeue(enqueue(4, enqueue(3, empty)))$

$\overset{(3)}{=} enqueue(4, dequeue(enqueue(3, empty())))$

$\overset{(3)}{=} enqueue(4, empty())$

Stack




Last In, First Out (LIFO)

Interface:

STACK e:

empty: -> STACK } Init.

isEmpty: STACK -> Bool } queries

top: STACK -> e

push: e x STACK -> STACK } modify

pop: STACK -> STACK

Axiome:

s : STACK

x : e

(1) isEmpty (empty()) = True

" (push(x,s)) = False

(2) pop (empty()) = ERROR

pop (push(x,s)) = s

(3) top (empty()) = ERROR

top (push(x,s)) = x

(4) push (top(s), pop(s)) = s , falls s nicht leer ist

1. $s \leftarrow empty()$
2. $s \leftarrow push('a', s)$ 

3. $s \leftarrow push('b', s)$ 

4. $s \leftarrow pop(s)$

$$pop(push('b', push('a', empty()))) \overset{(2)}{=} push('a', empty())$$

## Algorithmus 3.7 (Graph-Scan)

**Eingabe:** Graph $G = (V, E)$, Knoten $s \in V$

**Ausgabe:**

1. Knotenmenge $Y \subseteq V$, die von $s$ aus erreichbar ist (die Zusammenhangskomponente von $s$)
2. Kantenmenge $T \subseteq E$, die die Erreichbarkeit sicherstellt (Spannbaum der Zusammenhangskomponente)

```
1:  R ← {s}
2:  Y ← {s}
3:  T ← ∅

4:  while R ≠ ∅ do
5:      v ← wähle Knoten aus R

6:      if es gibt kein w ∈ V \ Y mit {v, w} ∈ E then
7:          R ← R \ {v}
8:      else
9:          w ← wähle w ∈ V \ Y mit e = {v, w} ∈ E

10:         R ← R ∪ {w}
11:         Y ← Y ∪ {w}
12:         T ← T ∪ {e}
13:     end if
14: end while
```