

# Übung 3 – Algorithmen II

Moritz Laupichler, Nikolai Maas – {moritz.laupichler, nikolai.maas}@kit.edu  
[https://algo2.iti.kit.edu/AlgorithmenII\\_WS23.php](https://algo2.iti.kit.edu/AlgorithmenII_WS23.php)

Institut für Theoretische Informatik - Algorithm Engineering

```
    result = current_weight;
    return true;
}

for( EdgeID eid = graph.edgeBegin( current ); eid != graph.edgeEnd( current ); ++eid ){
    const Edge & edge = graph.getEdge( eid );
    COUNTING( statistic_data.inc( DijkstraStatisticData::TOUCHED_EDGES ); )
    if( edge.forward ){
        COUNTING( statistic_data.inc( DijkstraStatisticData::RELAXED_EDGES ); )
        weight new_weight = edge.weight + current_weight;
        GUARANTEE( new_weight >= current_weight, std::runtime_error, "Weight overflow detected." );
        if( !priority_queue.isReached( edge.target ) ){
            COUNTING( statistic_data.inc( DijkstraStatisticData::SUCCESSFULLY_RELAXED_EDGES ); )
            COUNTING( statistic_data.inc( DijkstraStatisticData::REACHED_NODES ); )
            priority_queue.push( edge.target, new_weight );
        } else {
            if( priority_queue.getCurrentKey( edge.target ) > new_weight ){
                COUNTING( statistic_data.inc( DijkstraStatisticData::SUCCESSFULLY_RELAXED_NODES ); )
                priority_queue.decreaseKey( edge.target, new_weight );
            }
        }
    }
}
```

- SCCs mit DFS berechnen
- Besprechung Übungsblatt 1

# Starke Zusammenhangskomponenten

## Invariante 1

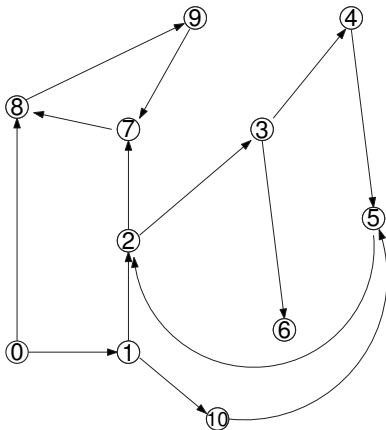
Keine Kanten von geschlossenen in offene Komponenten.

## Invariante 2

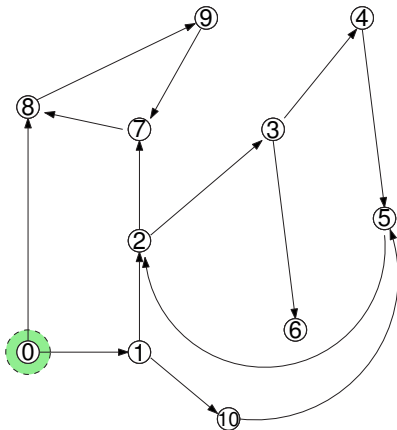
Offene Komponenten liegen auf Pfad.

## Invariante 3

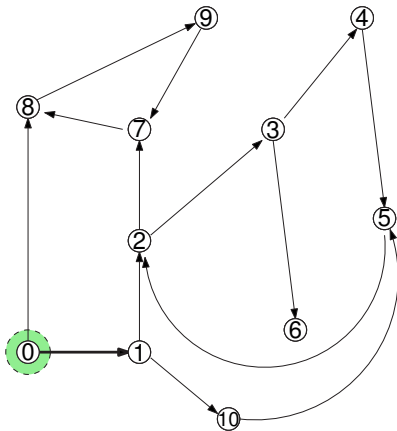
Repräsentanten partitionieren offene Komponenten bzgl. dfsNum.



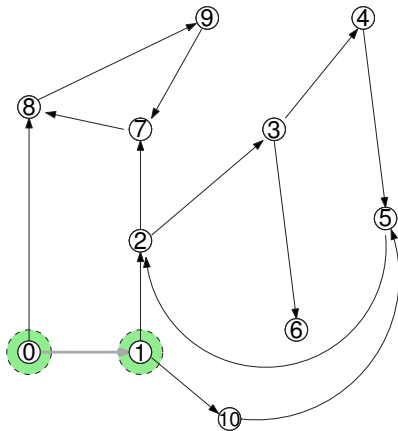
$oNodes$	$oReps$



oNodes	oReps
0	0

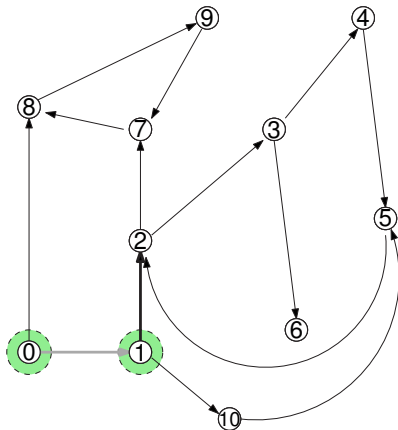


oNodes	oReps
0	0

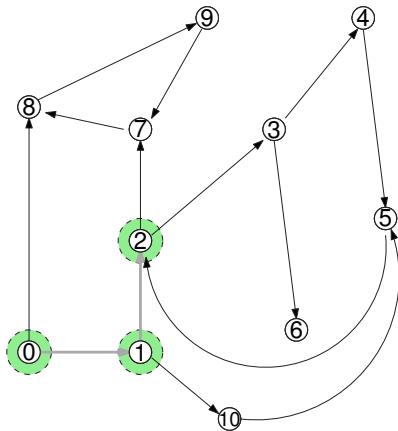


oNodes	oReps
0	0
1	1

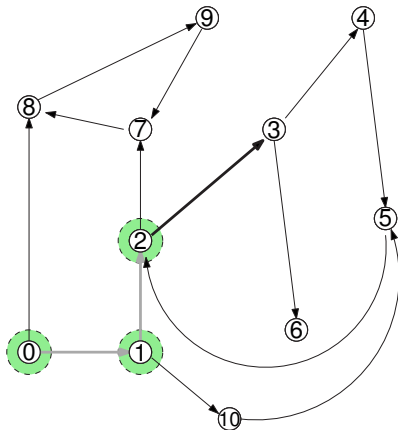




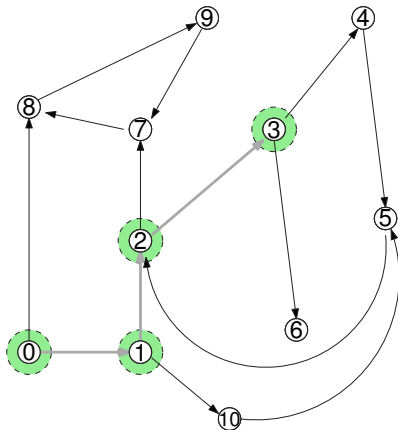
oNodes	oReps
0	0
1	1



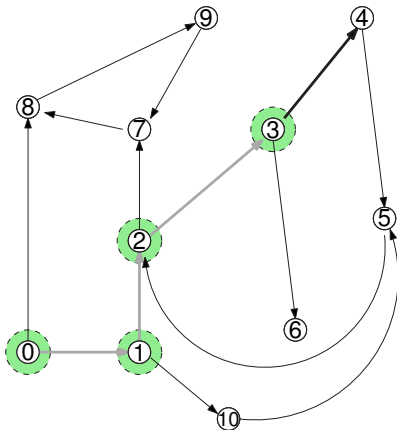
oNodes	oReps
0	0
1	1
2	2



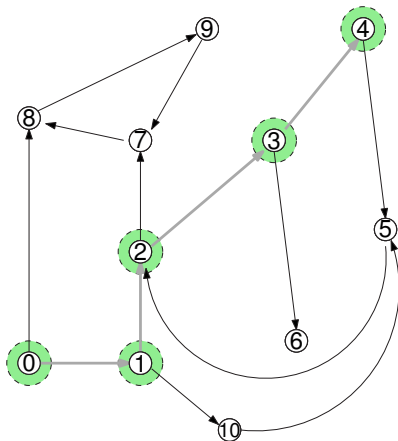
oNodes	oReps
0	0
1	1
2	2



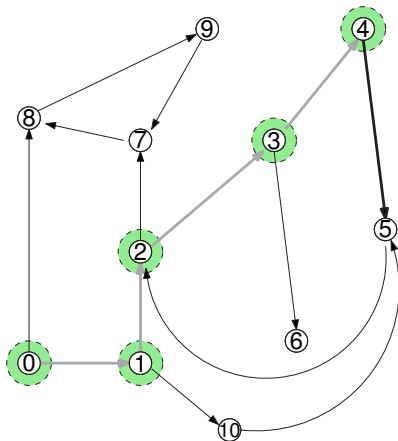
oNodes	oReps
0	0
1	1
2	2
3	3



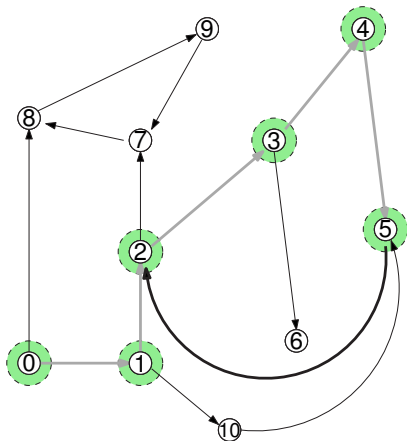
oNodes	oReps
0	0
1	1
2	2
3	3



oNodes	oReps
0	0
1	1
2	2
3	3
4	4

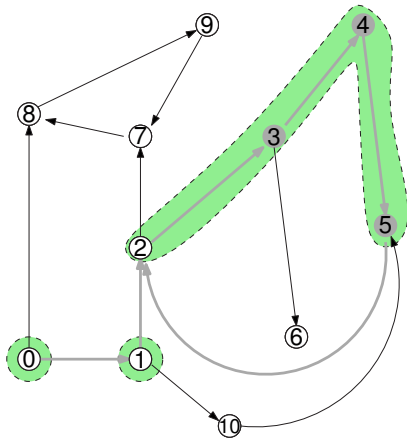


oNodes	oReps
0	0
1	1
2	2
3	3
4	4



oNodes	oReps
0	0
1	1
2	2
3	3
4	4
5	5





oNodes	oReps
0	0
1	1
2	2
3	
4	
5	

## Invariante 1

Keine Kanten von geschlossenen in offene Komponenten.

- Tiefensuche sucht Pfad durch Graphen
- Nur Rückwärtskanten ergeben Kreise
- Kreise vereinigen alle auf dem Kreis liegenden offenen Komponenten

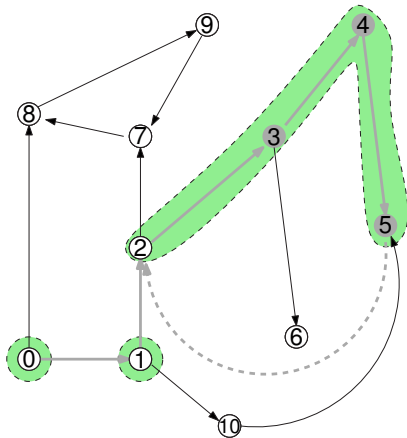
## Invariante 2

Offene Komponenten liegen auf Pfad.

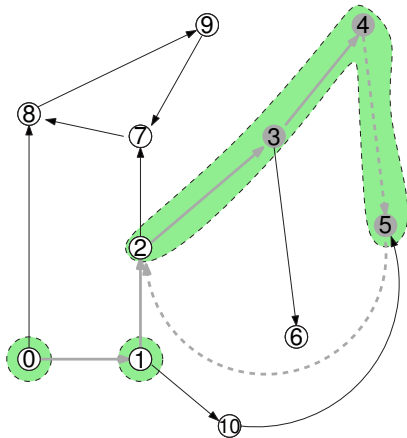
- Repräsentant ist minimal auf Kreis

## Invariante 3

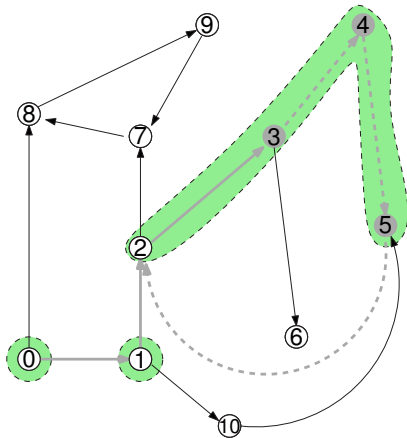
Repräsentanten partitionieren offene Komponenten bzgl. dfsNum.



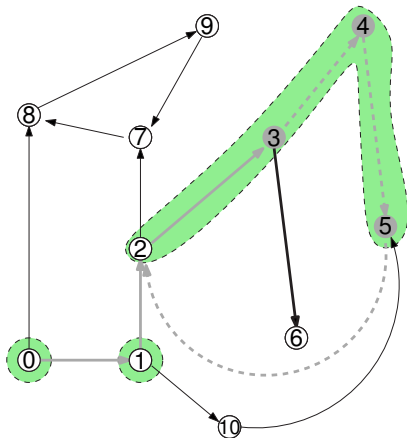
oNodes	oReps
0	0
1	1
2	2
3	
4	
5	



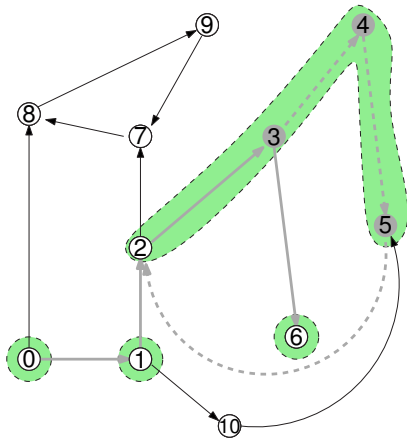
oNodes	oReps
0	0
1	1
2	2
3	
4	
5	



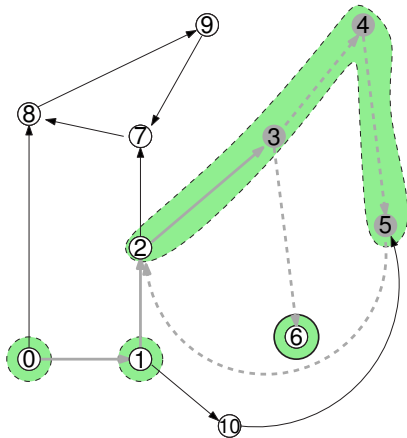
oNodes	oReps
0	0
1	1
2	2
3	
4	
5	



oNodes	oReps
0	0
1	1
2	2
3	
4	
5	



oNodes	oReps
0	0
1	1
2	2
3	
4	
5	
6	6



oNodes	oReps
0	0
1	1
2	2
3	
4	
5	
6	6



- Komponenten werden nach Bearbeitung aller ausgehenden Kanten geschlossen
- Alle offenen Komponenten liegen auf Stack
- Kante von geschlossener in offene Komponenten hätte bei Bearbeitung Kreis induziert

## Invariante 1

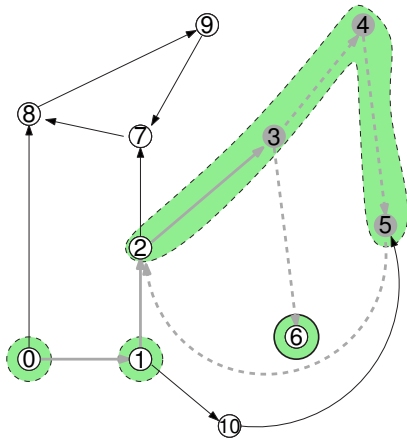
Keine Kanten von geschlossenen in offene Komponenten.

## Invariante 2

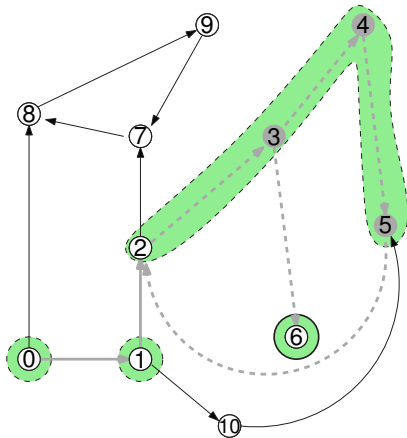
Offene Komponenten liegen auf Pfad.

## Invariante 3

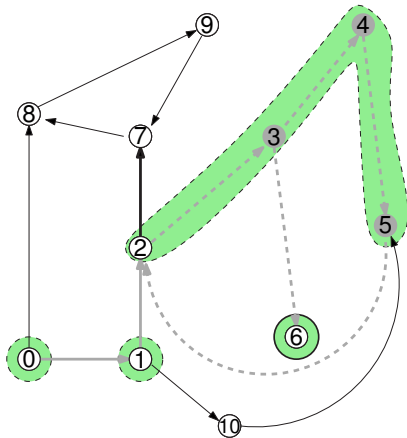
Repräsentanten partitionieren offene Komponenten bzgl. dfsNum.



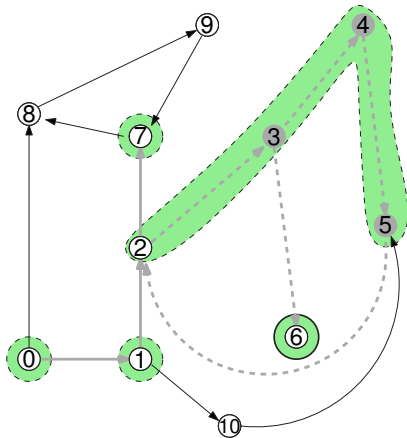
oNodes	oReps
0	0
1	1
2	2
3	
4	
5	



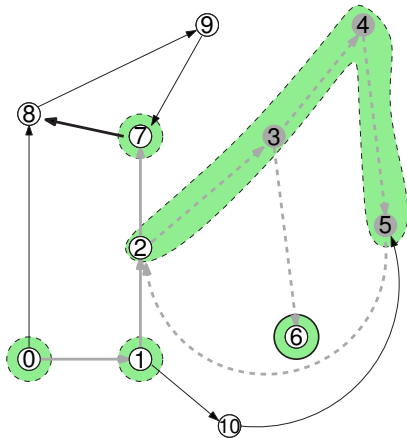
oNodes	oReps
0	0
1	1
2	2
3	
4	
5	



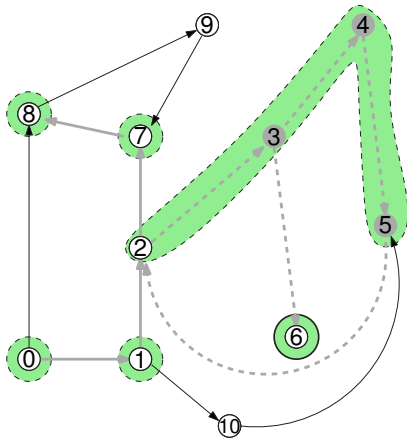
oNodes	oReps
0	0
1	1
2	2
3	
4	
5	



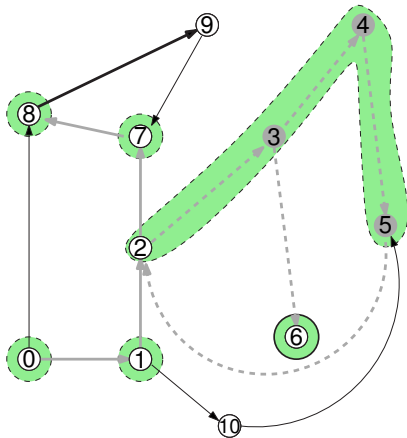
oNodes	oReps
0	0
1	1
2	2
3	
4	
5	
7	7



oNodes	oReps
0	0
1	1
2	2
3	
4	
5	
7	7

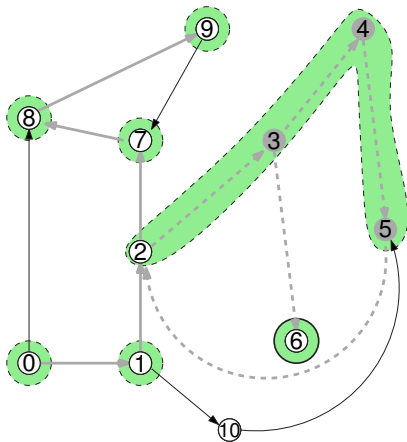


oNodes	oReps
0	0
1	1
2	2
3	
4	
5	
7	7
8	8

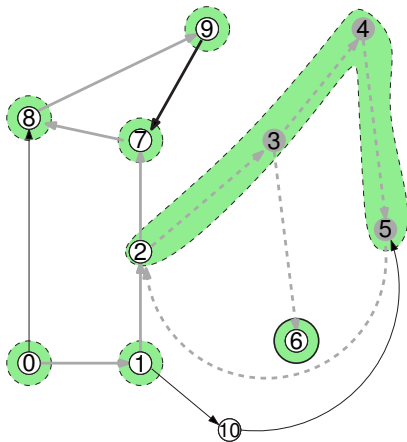


oNodes	oReps
0	0
1	1
2	2
3	
4	
5	
7	7
8	8

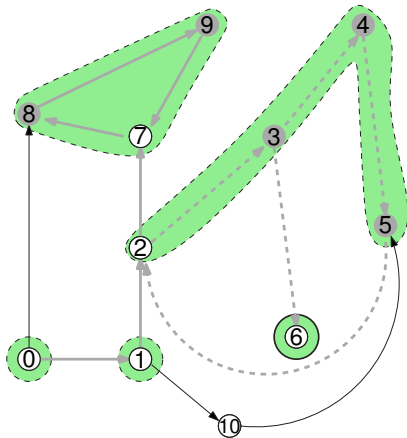




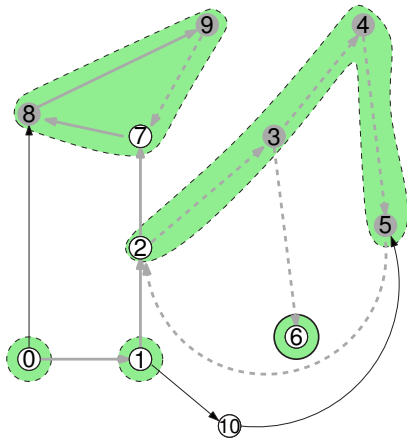
oNodes	oReps
0	0
1	1
2	2
3	
4	
5	
7	7
8	8
9	9



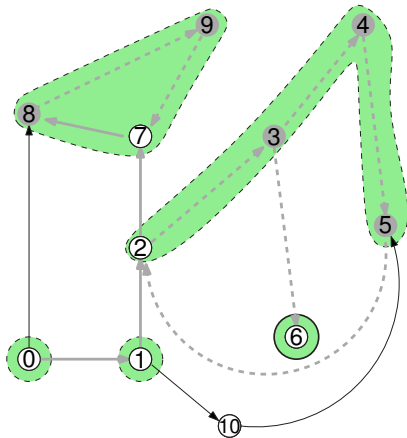
oNodes	oReps
0	0
1	1
2	2
3	
4	
5	
7	7
8	8
9	9



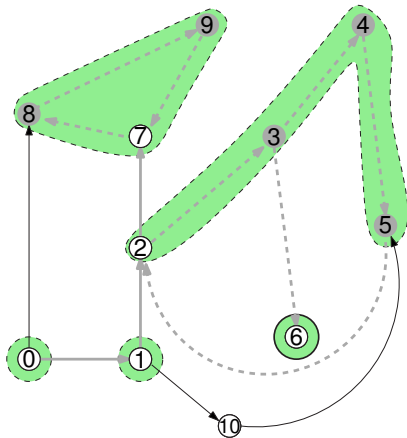
oNodes	oReps
0	0
1	1
2	2
3	
4	
5	
7	7
8	
9	



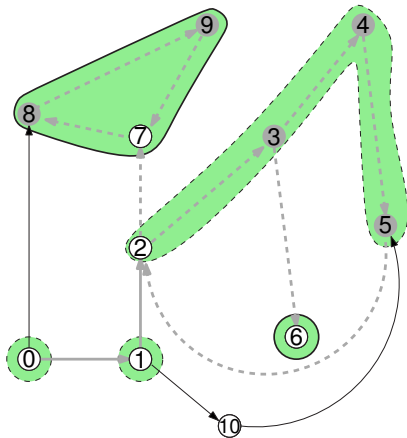
oNodes	oReps
0	0
1	1
2	2
3	
4	
5	
7	7
8	
9	



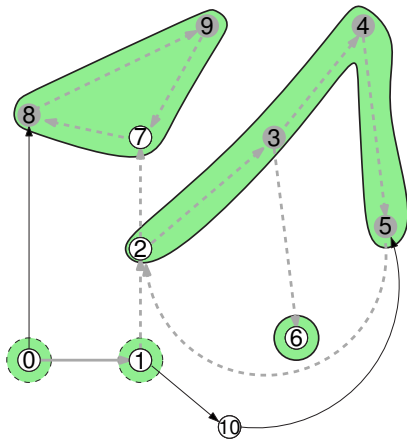
oNodes	oReps
0	0
1	1
2	2
3	
4	
5	
7	7
8	
9	



oNodes	oReps
0	0
1	1
2	2
3	
4	
5	
7	7
8	
9	



oNodes	oReps
0	0
1	1
2	2
3	
4	
5	
7	7
8	
9	



oNodes	oReps
0	0
1	1
2	2
3	
4	
5	



- Aufgabe 3: Best-Case Verhalten von Fibonacci-Heaps
- Aufgabe 7: Laufzeit von Dijkstra

# Ende!



# Feierabend!