

Übung 10 – Algorithmen II

Yaroslav Akhremtsev, Demian Hespe – yaroslav.akhremtsev@kit.edu, hespe@kit.edu

Mit Folien von Michael Axtmann

http://algo2.iti.kit.edu/AlgorithmenII_WS17.php

Institut für Theoretische Informatik - Algorithmik II

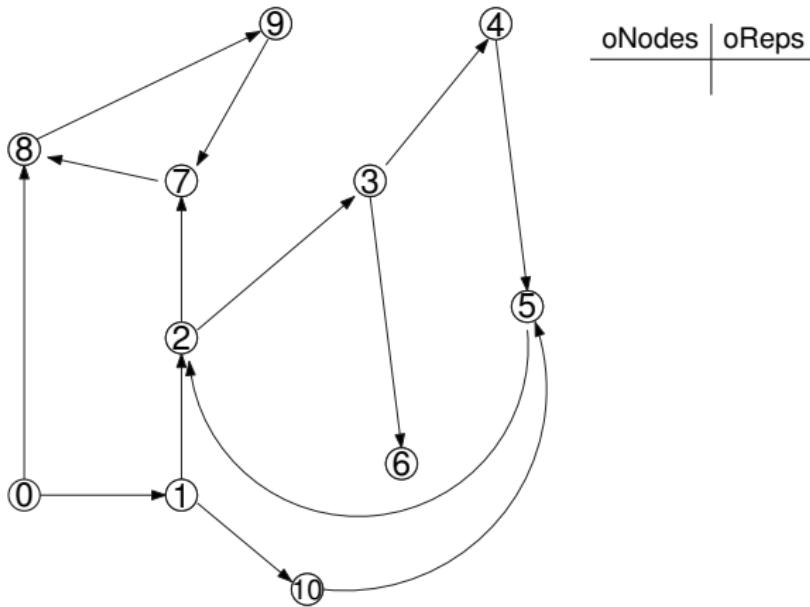
```
    result = current_weight;
    return true;
}

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::INCORRECTLY_RELAXED_EDGES ) );
                priority_queue.decreaseKey( edge.target, new_weight );
            }
        }
    }
}
```

- SCC - Ein Ausblick
 - SCC Algorithmus
 - Wiederholung
 - Invarianten
 - vertiefendes Beispiel – Floyd Warshall
 - nicht relevant für Klausur

Starke Zusammenhangskomponenten

SCC (Wiederholung)



Invariante 1

Kein Kanten von geschlossenen in offene Komponenten.

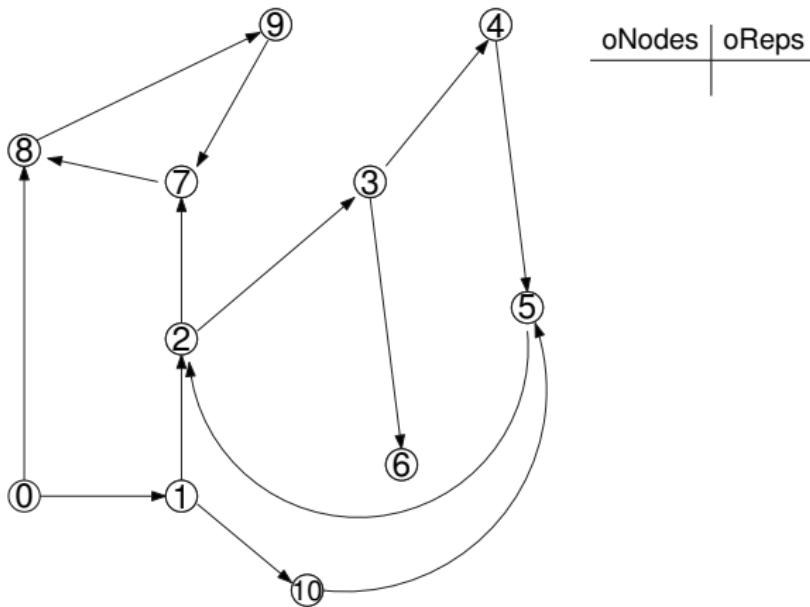
Invariante 2

Offene Komponenten liegen auf Pfad.

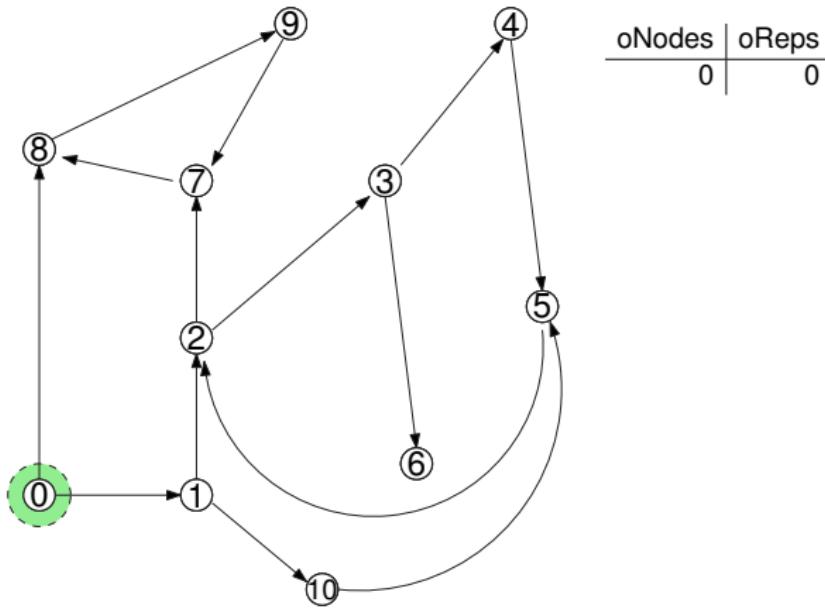
Invariante 3

Repräsentanten partitionieren offene Komponenten bzgl. dfsNum.

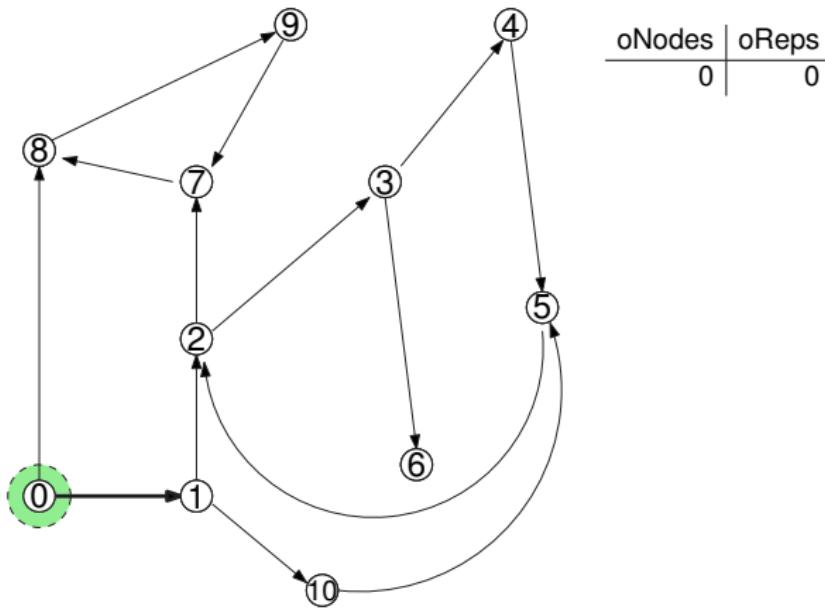
SCC (Wiederholung)



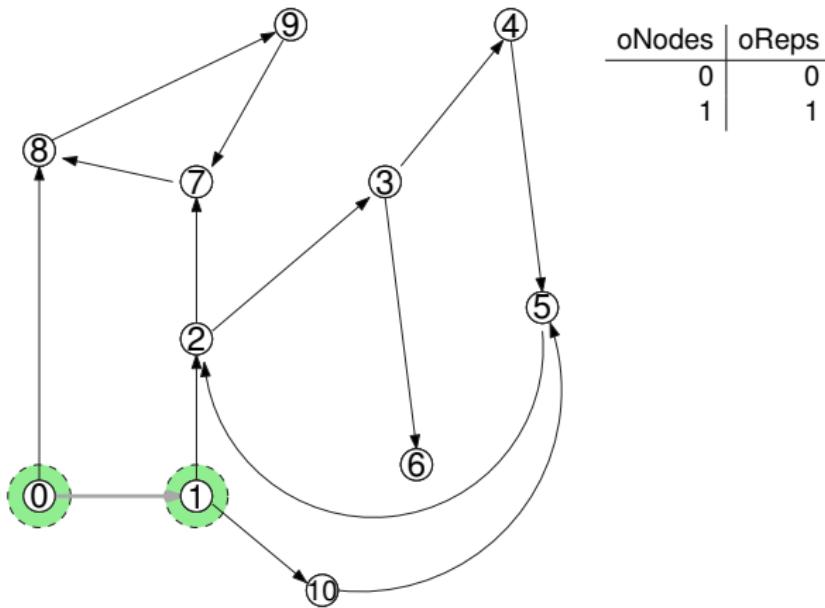
SCC (Wiederholung)



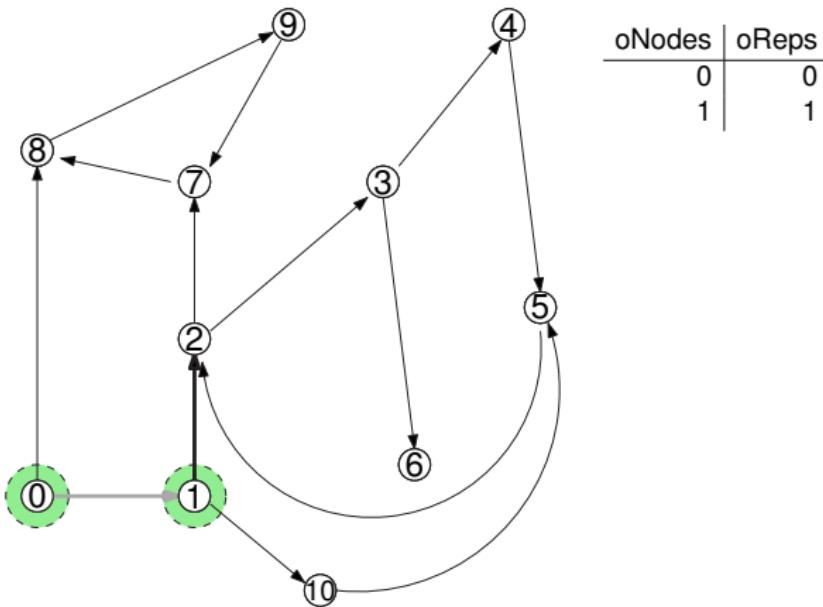
SCC (Wiederholung)



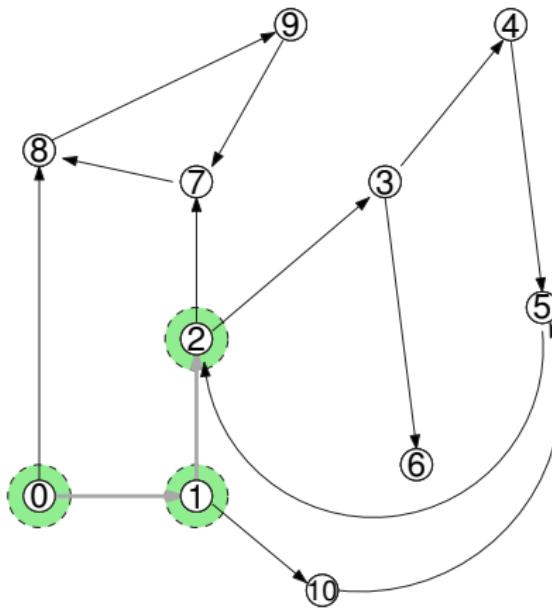
SCC (Wiederholung)



SCC (Wiederholung)

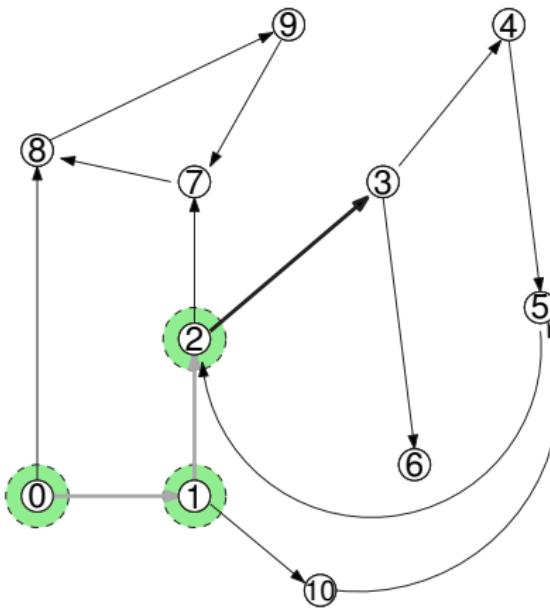


SCC (Wiederholung)



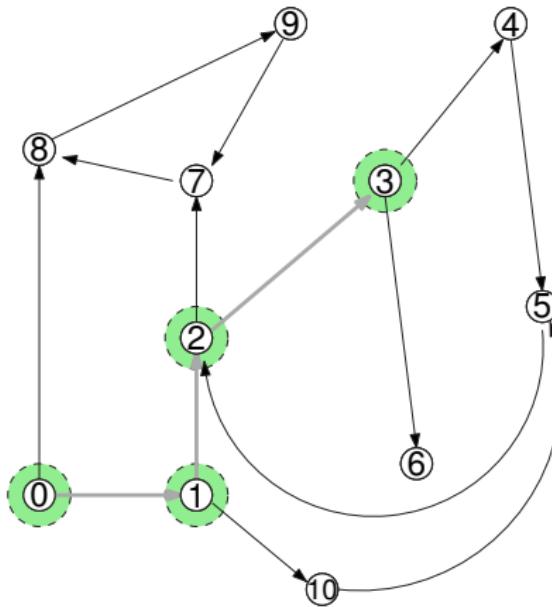
oNodes	oReps
0	0
1	1
2	2

SCC (Wiederholung)



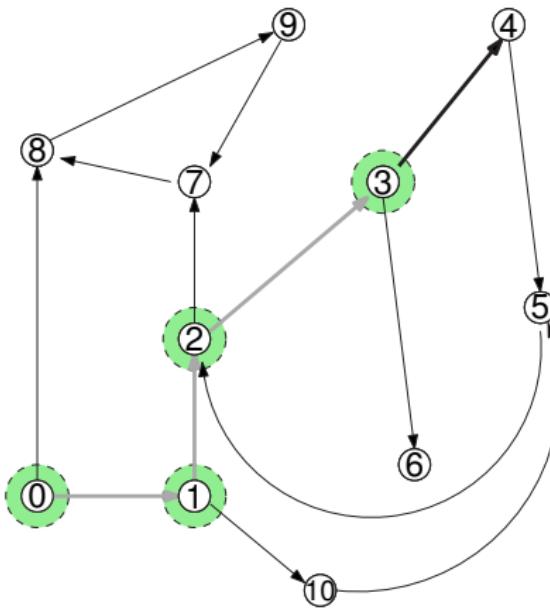
oNodes	oReps
0	0
1	1
2	2

SCC (Wiederholung)



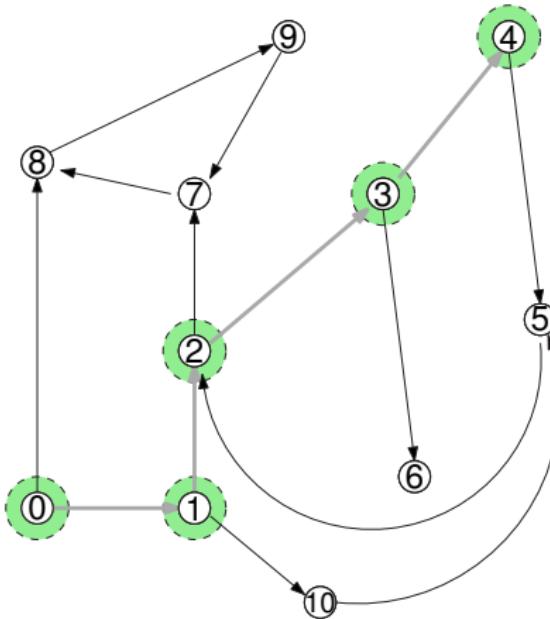
oNodes	oReps
0	0
1	1
2	2
3	3

SCC (Wiederholung)



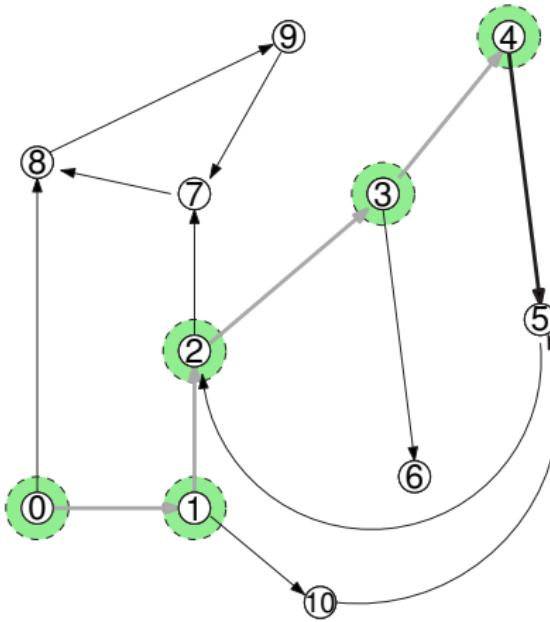
oNodes	oReps
0	0
1	1
2	2
3	3

SCC (Wiederholung)



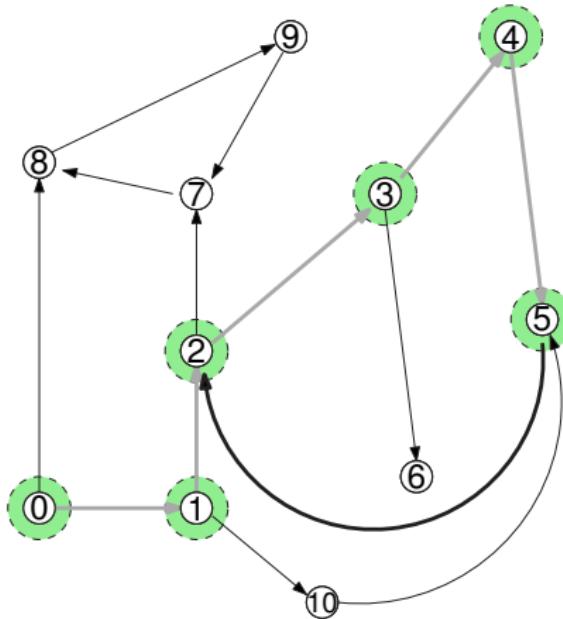
oNodes	oReps
0	0
1	1
2	2
3	3
4	4

SCC (Wiederholung)



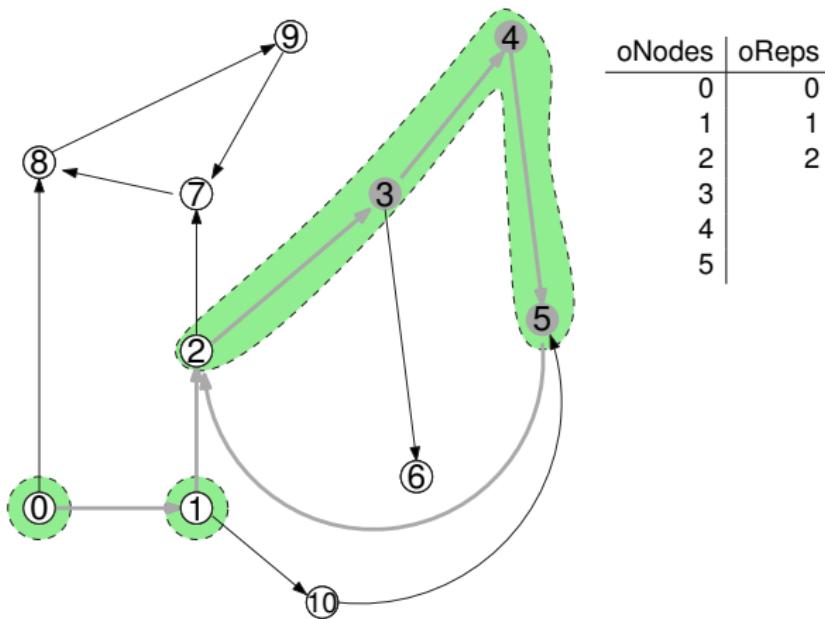
oNodes	oReps
0	0
1	1
2	2
3	3
4	4

SCC (Wiederholung)



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

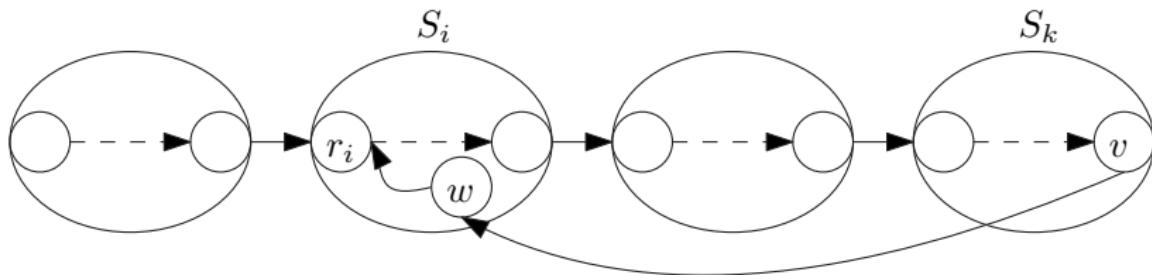
SCC (Wiederholung)



Invariante 1

Kein 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

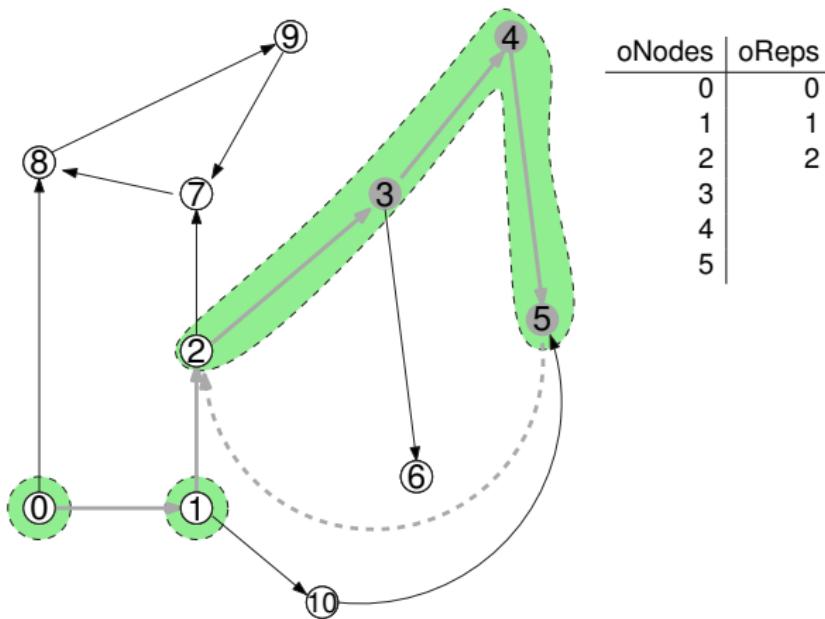
Offene Komponenten liegen auf Pfad.

- Repräsentant ist minimal auf Kreis

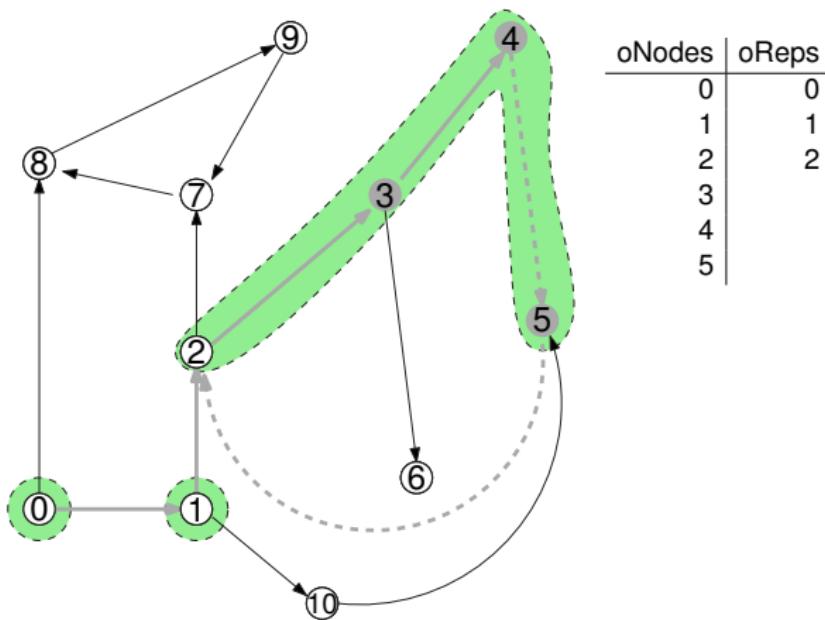
Invariante 3

Repräsentanten partitionieren offene Komponenten bzgl. dfsNum.

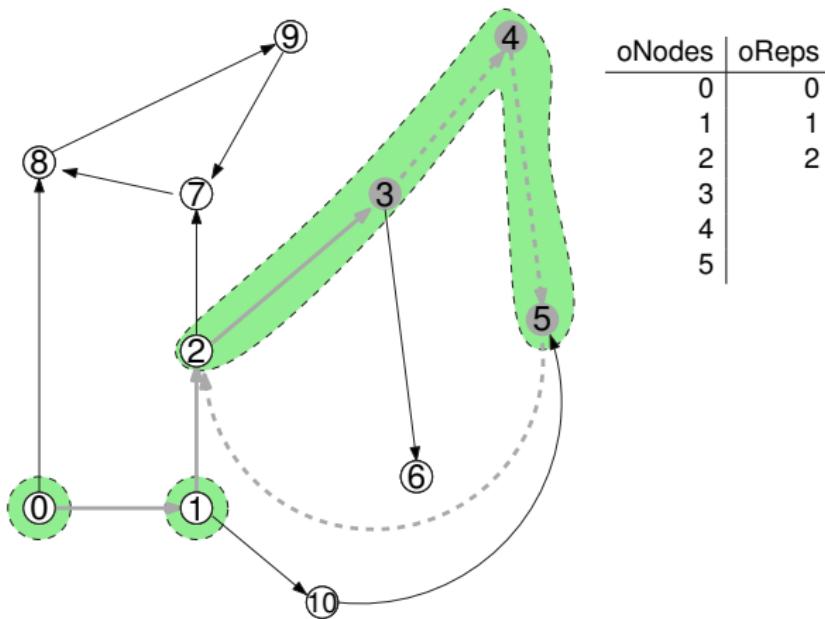
SCC (Wiederholung)



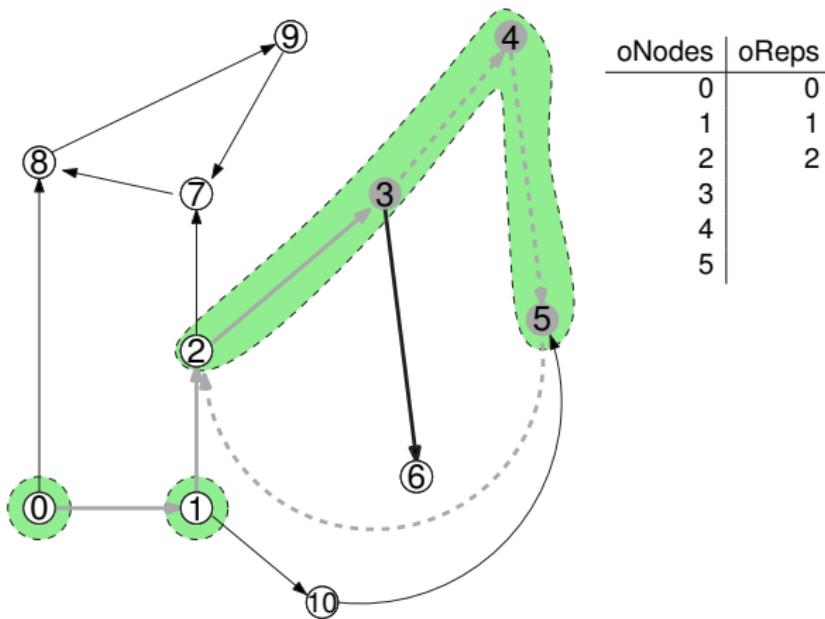
SCC (Wiederholung)



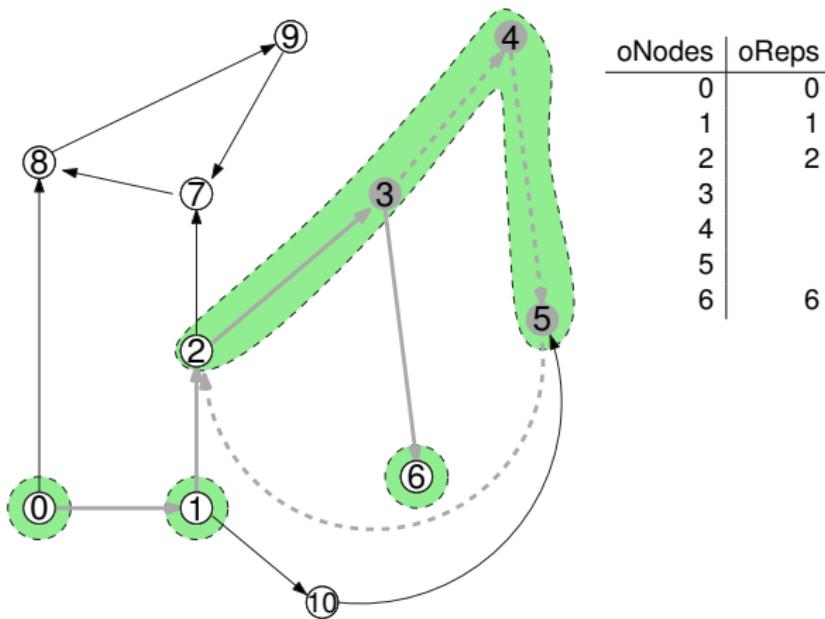
SCC (Wiederholung)



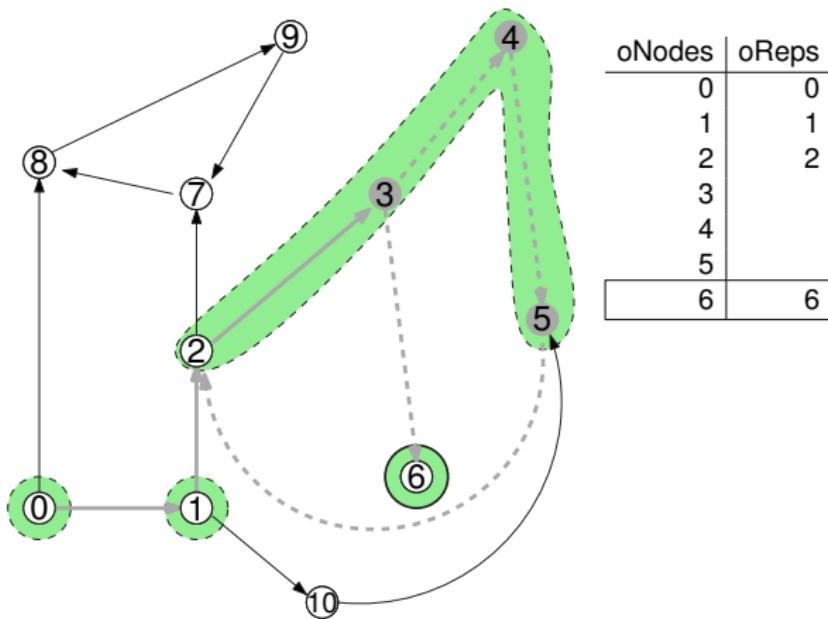
SCC (Wiederholung)



SCC (Wiederholung)



SCC (Wiederholung)



- Komponenten werden nach Bearbeitung aller ausgehenden Kanten geschlossen
- Alle offenen Komponenten liegen auf Stack

Invariante 1

Kein Kanten von geschlossenen in offene Komponenten.

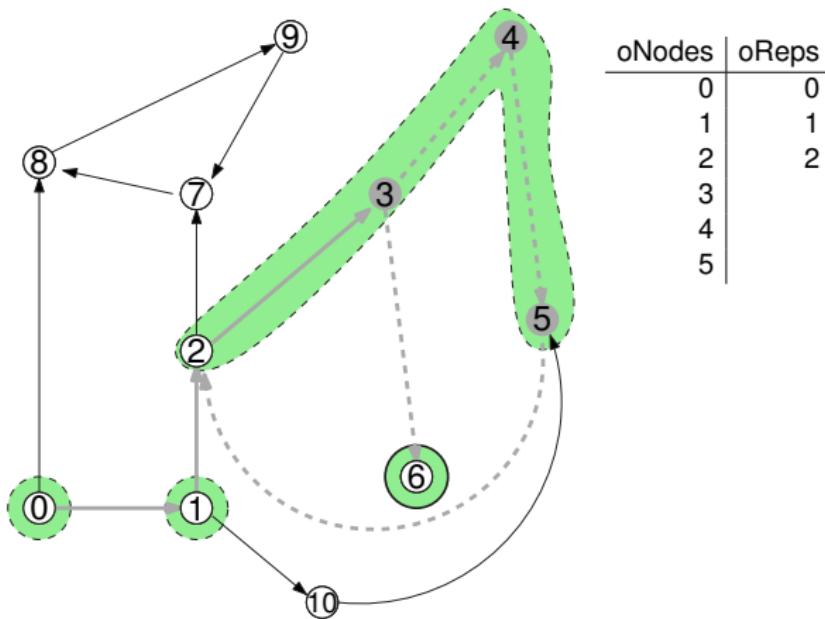
Invariante 2

Offene Komponenten liegen auf Pfad.

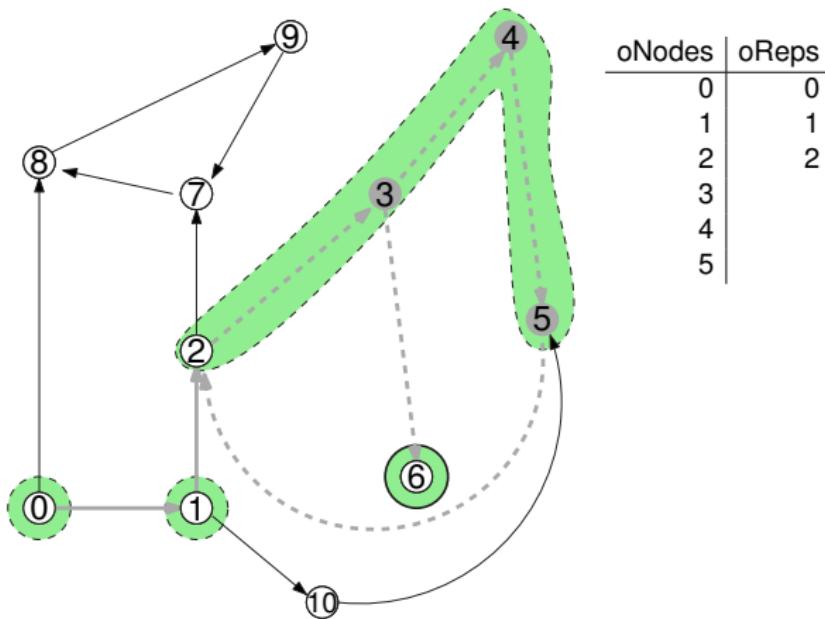
Invariante 3

Repräsentanten partitionieren offene Komponenten bzgl. dfsNum.

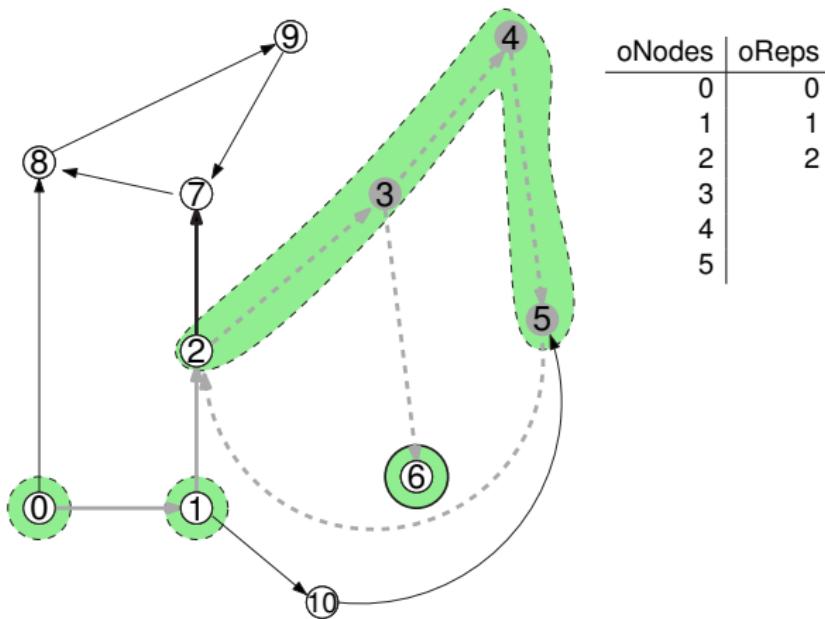
SCC (Wiederholung)



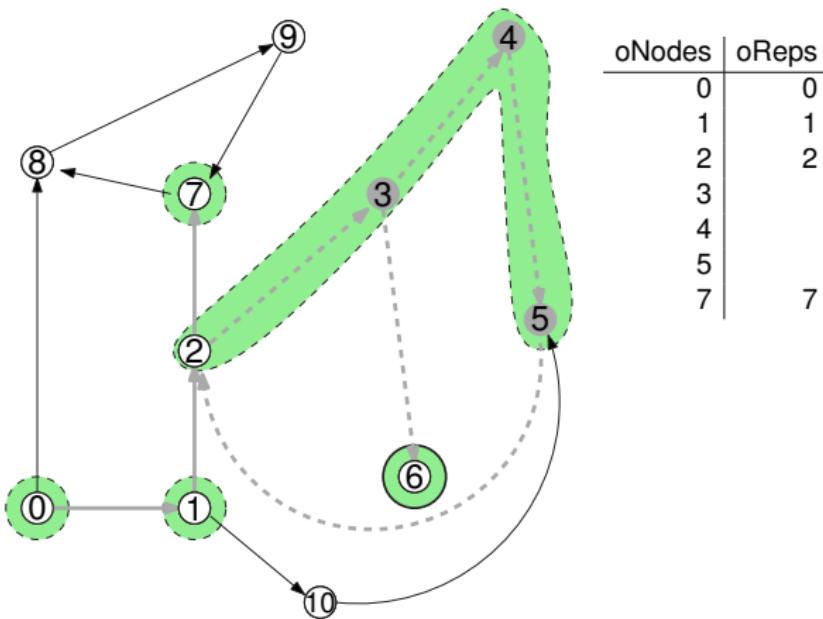
SCC (Wiederholung)



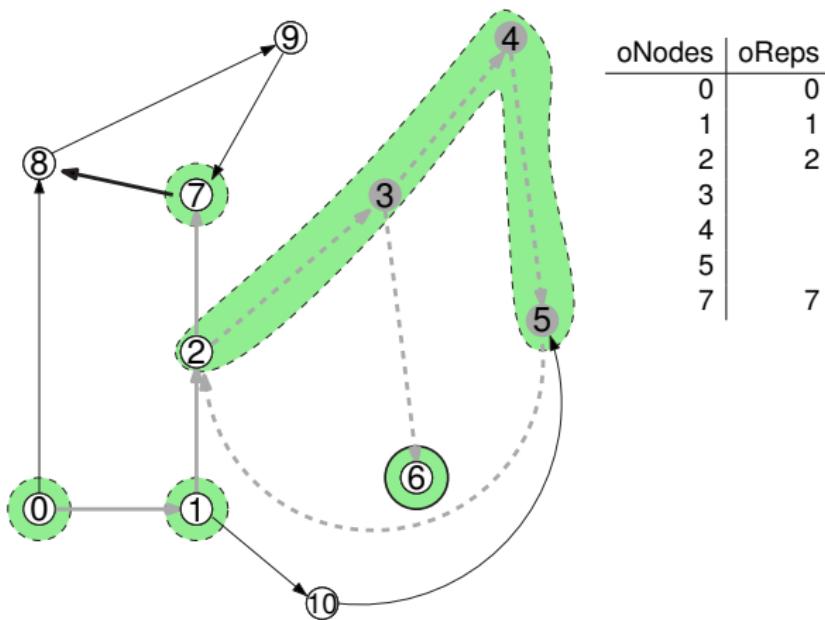
SCC (Wiederholung)



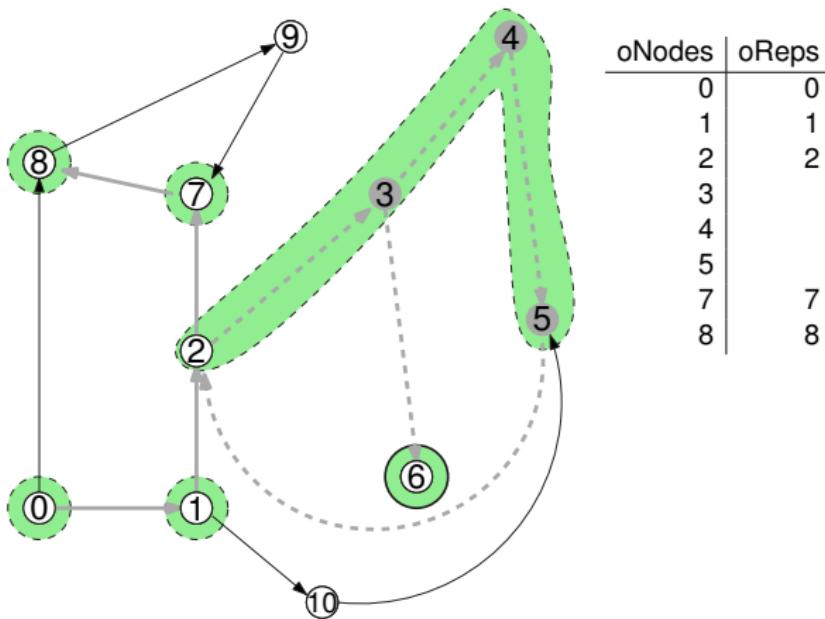
SCC (Wiederholung)



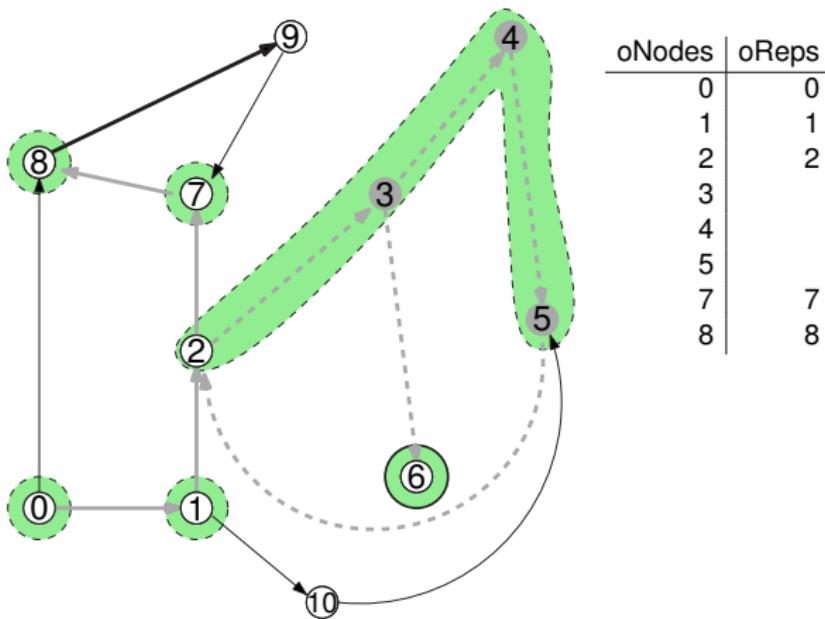
SCC (Wiederholung)



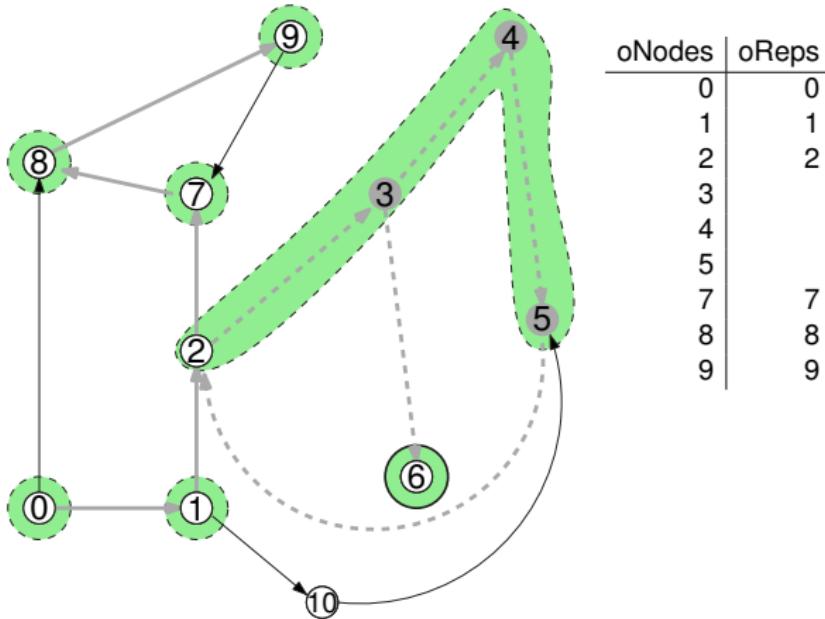
SCC (Wiederholung)



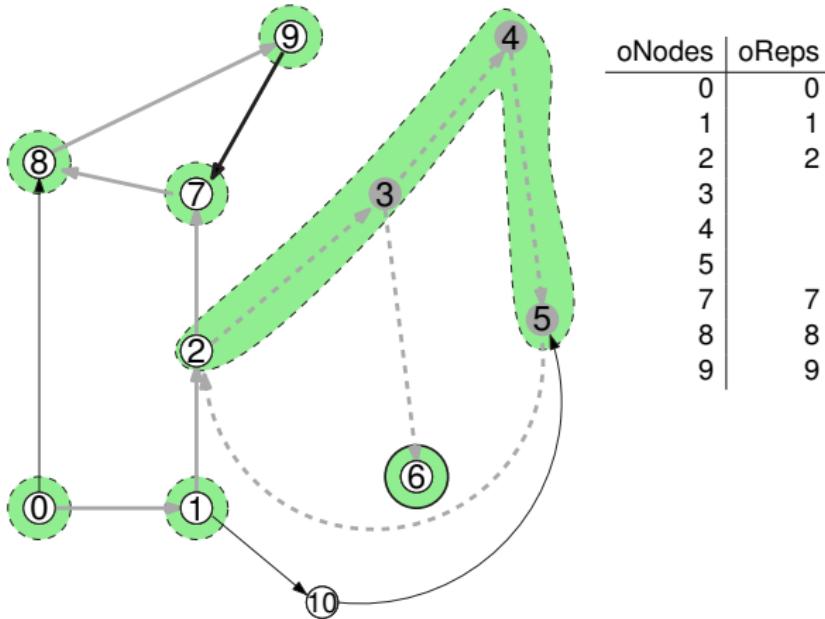
SCC (Wiederholung)



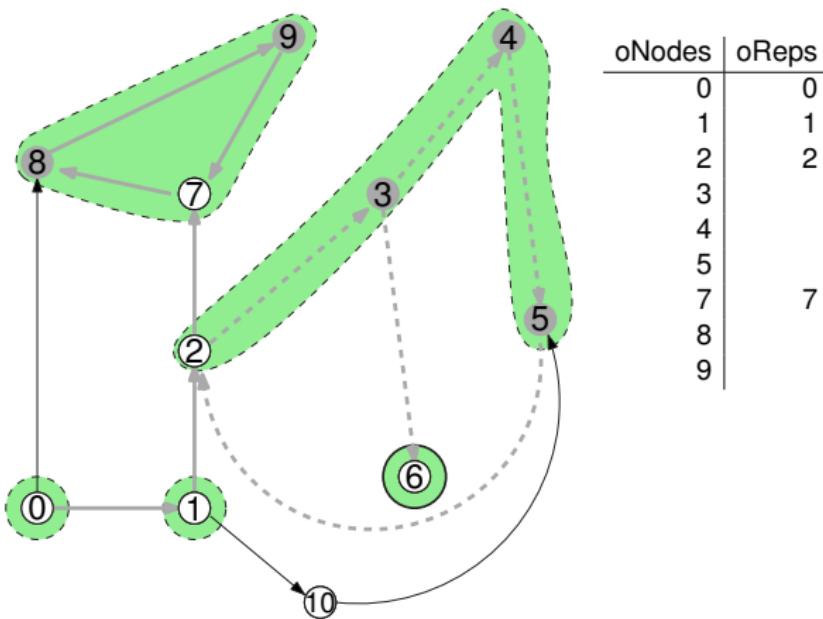
SCC (Wiederholung)



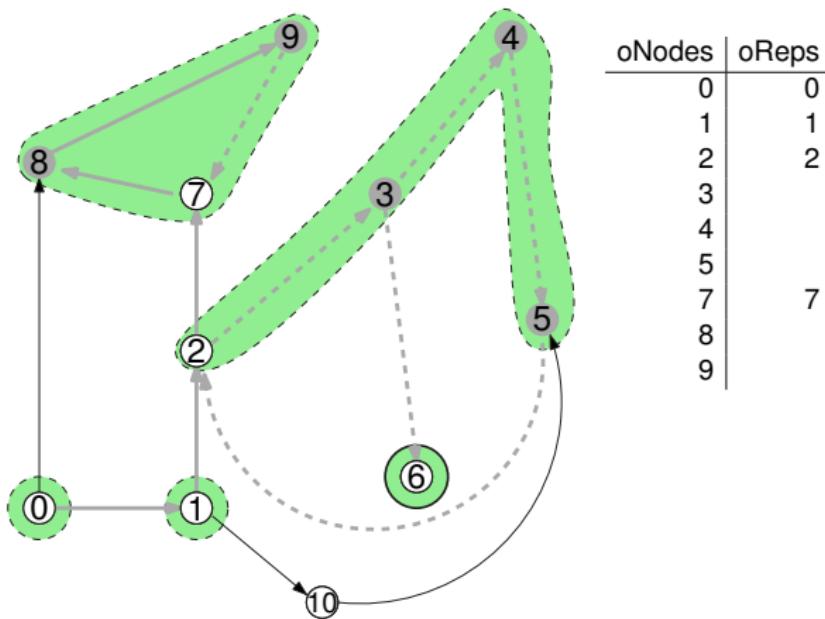
SCC (Wiederholung)



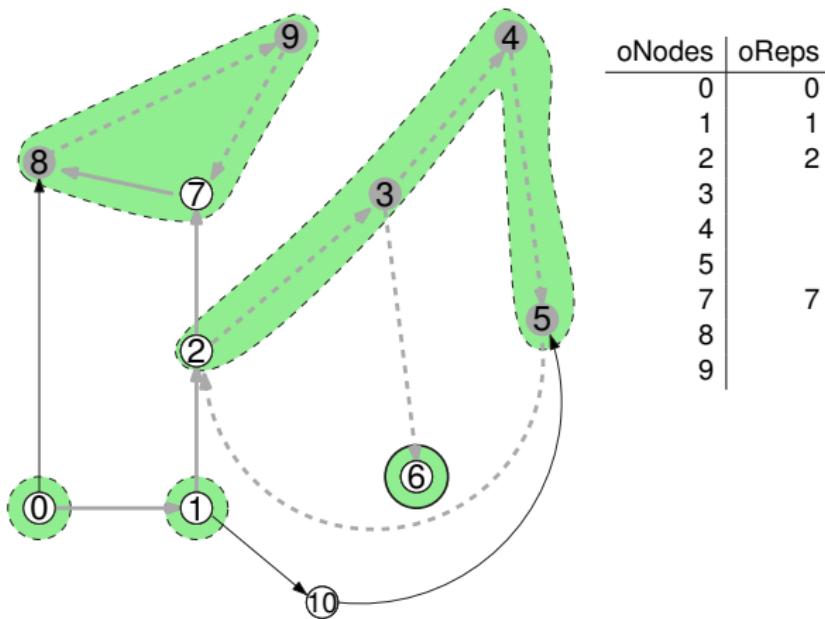
SCC (Wiederholung)



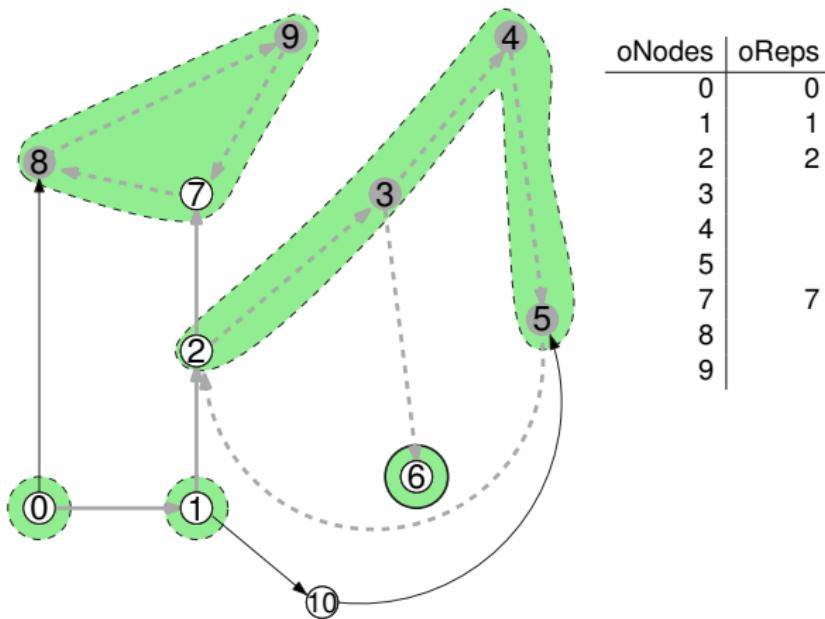
SCC (Wiederholung)



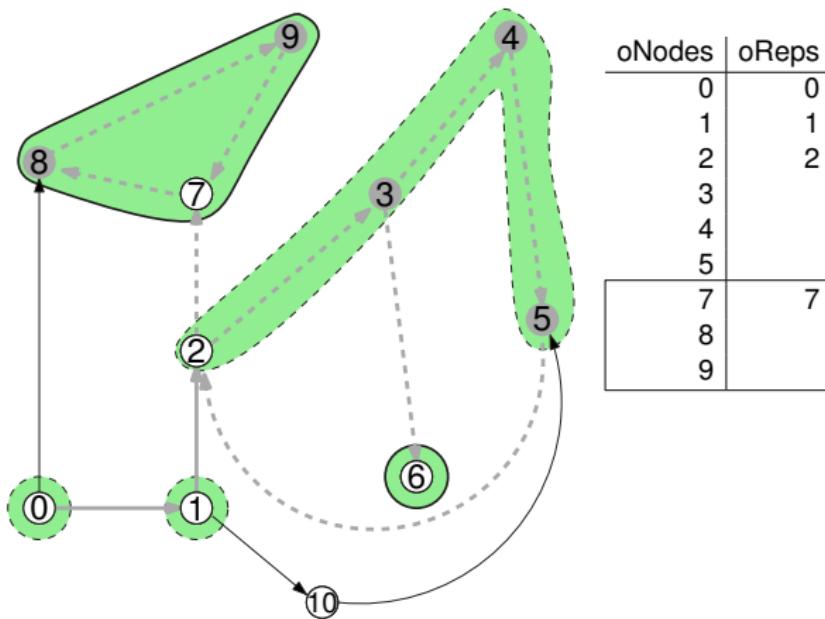
SCC (Wiederholung)



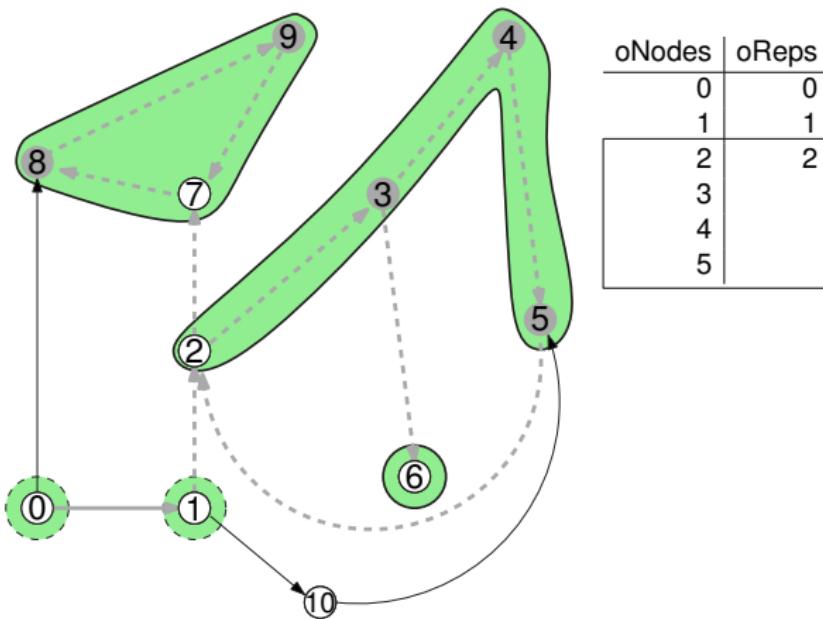
SCC (Wiederholung)



SCC (Wiederholung)



SCC (Wiederholung)



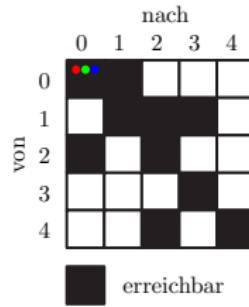
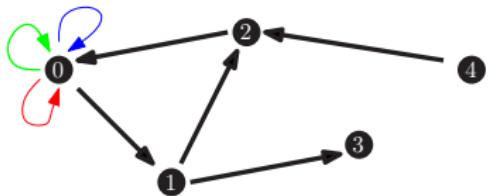
Floyd Warshall: SCC als Speedup Technik

Floyd Warshall

- kubischer Algorithmus
- berechnet transitive Hülle
 - auch anwendbar für *all-to-all* kürzeste Wege

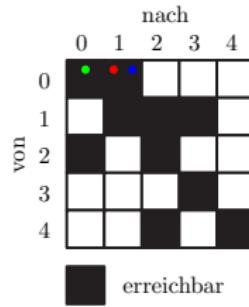
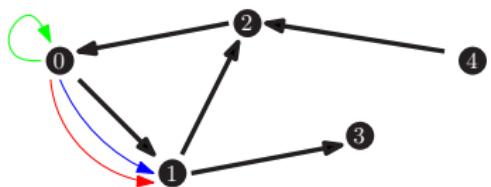
```
for int k = 0; k < n; ++k do
    for int i = 0; i < n; ++i do
        for int j = 0; j < n; ++j do
            array[i][j] = array[i][j] ||
                (array[i][k] && array[k][j]);
        end
    end
end
```

Floyd Warshall



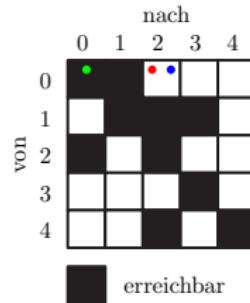
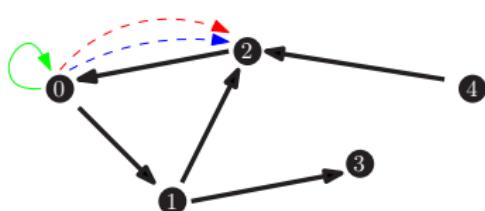
```
for int k = 0; k < n; ++k do
    for int i = 0; i < n; ++i do
        for int j = 0; j < n; ++j do
            array[i][j] = array[i][j] ||
                (array[i][k] && array[k][j]);
```

Floyd Warshall



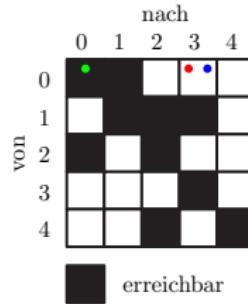
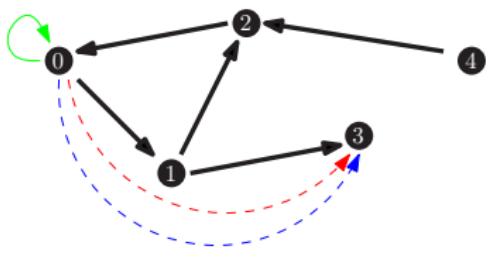
```
for int k = 0; k < n; ++k do
    for int i = 0; i < n; ++i do
        for int j = 0; j < n; ++j do
            array[i][j] = array[i][j] ||
                (array[i][k] && array[k][j]);
```

Floyd Warshall



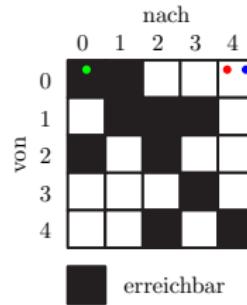
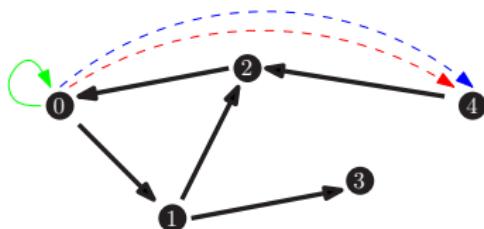
```
for int k = 0; k < n; ++k do
    for int i = 0; i < n; ++i do
        for int j = 0; j < n; ++j do
            array[i][j] = array[i][j] ||
                (array[i][k] && array[k][j]);
```

Floyd Warshall



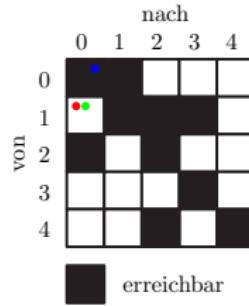
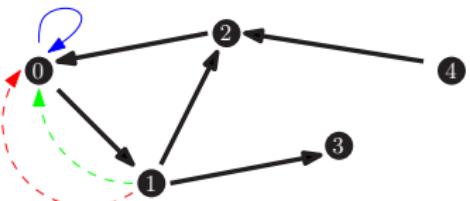
```
for int k = 0; k < n; ++k do
    for int i = 0; i < n; ++i do
        for int j = 0; j < n; ++j do
            array[i][j] = array[i][j] ||
                (array[i][k] && array[k][j]);
        end
    end
end
```

Floyd Warshall



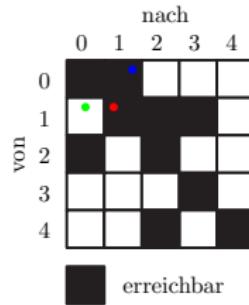
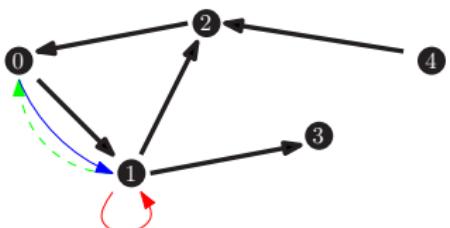
```
for int k = 0; k < n; ++k do
    for int i = 0; i < n; ++i do
        for int j = 0; j < n; ++j do
            array[i][j] = array[i][j] ||
                (array[i][k] && array[k][j]);
        end
    end
end
```

Floyd Warshall



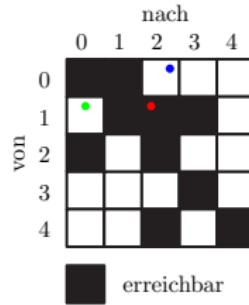
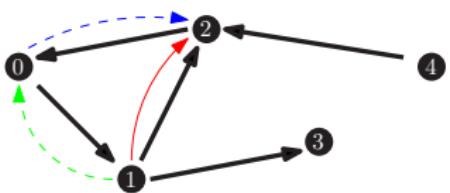
```
for int k = 0; k < n; ++k do
    for int i = 0; i < n; ++i do
        for int j = 0; j < n; ++j do
            array[i][j] = array[i][j] ||
                (array[i][k] && array[k][j]);
        end
    end
end
```

Floyd Warshall



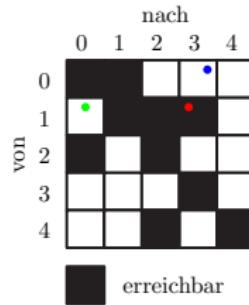
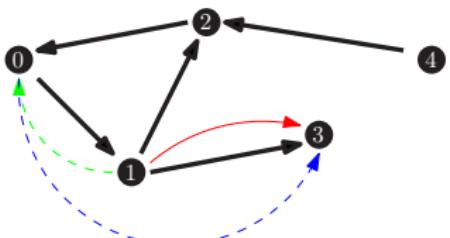
```
for int k = 0; k < n; ++k do
    for int i = 0; i < n; ++i do
        for int j = 0; j < n; ++j do
            array[i][j] = array[i][j] ||
                (array[i][k] && array[k][j]);
        end
    end
end
```

Floyd Warshall



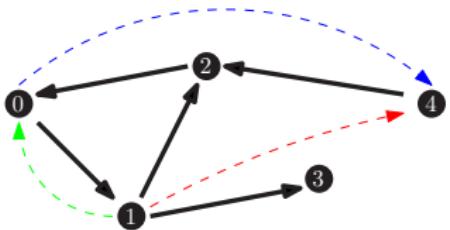
```
for int k = 0; k < n; ++k do
    for int i = 0; i < n; ++i do
        for int j = 0; j < n; ++j do
            array[i][j] = array[i][j] ||
                (array[i][k] && array[k][j]);
```

Floyd Warshall



```
for int k = 0; k < n; ++k do
    for int i = 0; i < n; ++i do
        for int j = 0; j < n; ++j do
            array[i][j] = array[i][j] ||
                (array[i][k] && array[k][j]);
        end
    end
end
```

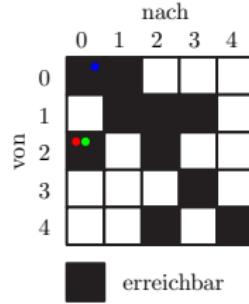
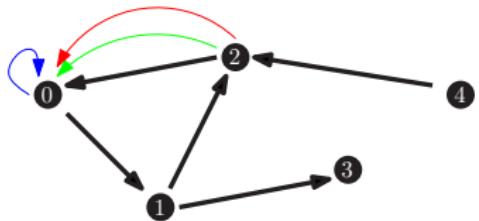
Floyd Warshall



		nach				
		0	1	2	3	4
von	0	■				●
	1	●				●
2		■				
3			■			
4				■		

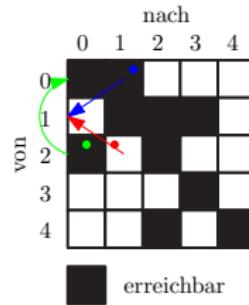
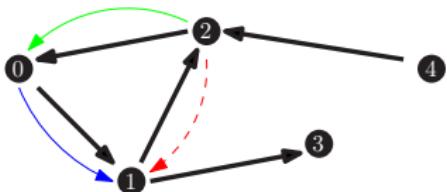
```
for int k = 0; k < n; ++k do
    for int i = 0; i < n; ++i do
        for int j = 0; j < n; ++j do
            array[i][j] = array[i][j] ||
                (array[i][k] && array[k][j]);
        end
    end
end
```

Floyd Warshall



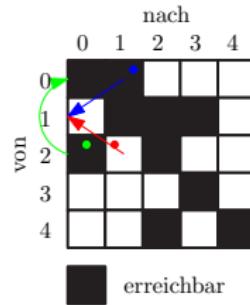
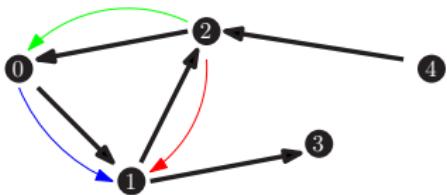
```
for int k = 0; k < n; ++k do
    for int i = 0; i < n; ++i do
        for int j = 0; j < n; ++j do
            array[i][j] = array[i][j] ||
                (array[i][k] && array[k][j]);
```

Floyd Warshall



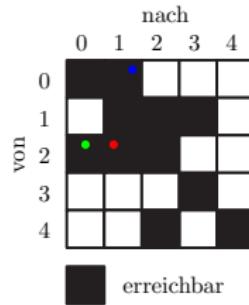
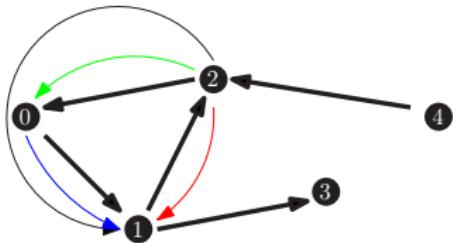
```
for int k = 0; k < n; ++k do
    for int i = 0; i < n; ++i do
        for int j = 0; j < n; ++j do
            array[i][j] = array[i][j] ||
                (array[i][k] && array[k][j]);
        end
    end
end
```

Floyd Warshall



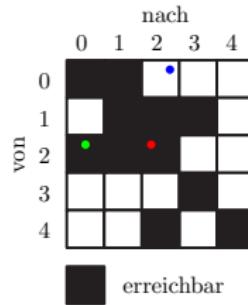
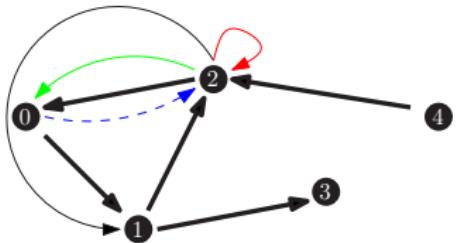
```
for int k = 0; k < n; ++k do
    for int i = 0; i < n; ++i do
        for int j = 0; j < n; ++j do
            array[i][j] = array[i][j] ||
                (array[i][k] && array[k][j]);
        end
    end
end
```

Floyd Warshall



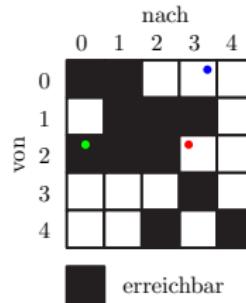
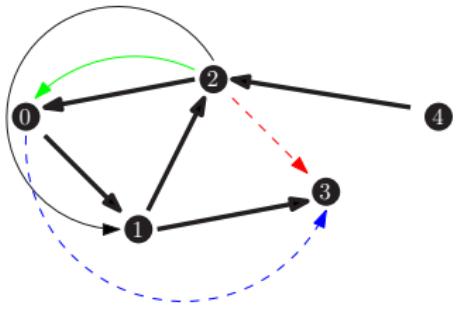
```
for int k = 0; k < n; ++k do
    for int i = 0; i < n; ++i do
        for int j = 0; j < n; ++j do
            array[i][j] = array[i][j] ||
                (array[i][k] && array[k][j]);
```

Floyd Warshall



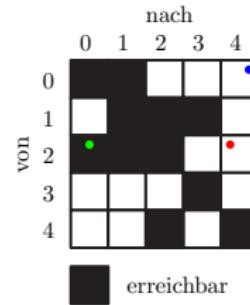
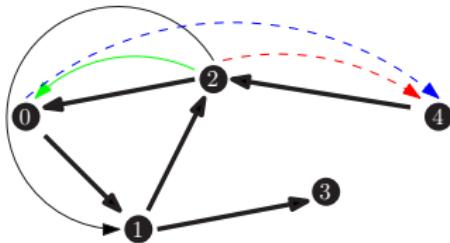
```
for int k = 0; k < n; ++k do
    for int i = 0; i < n; ++i do
        for int j = 0; j < n; ++j do
            array[i][j] = array[i][j] ||
                (array[i][k] && array[k][j]);
        end
    end
end
```

Floyd Warshall



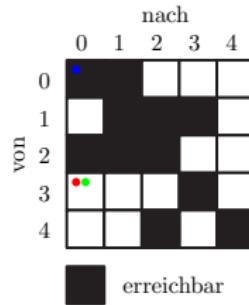
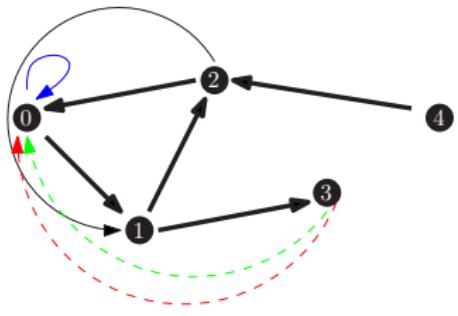
```
for int k = 0; k < n; ++k do
    for int i = 0; i < n; ++i do
        for int j = 0; j < n; ++j do
            array[i][j] = array[i][j] ||
                (array[i][k] && array[k][j]);
        end
    end
end
```

Floyd Warshall



```
for int k = 0; k < n; ++k do
    for int i = 0; i < n; ++i do
        for int j = 0; j < n; ++j do
            array[i][j] = array[i][j] ||
                (array[i][k] && array[k][j]);
        end
    end
end
```

Floyd Warshall

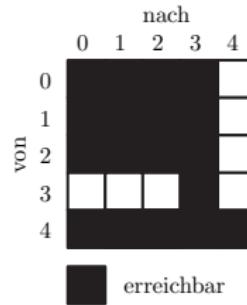
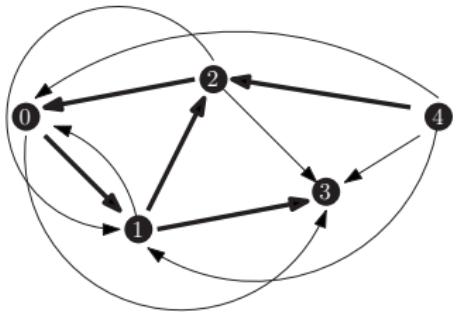


```
for int k = 0; k < n; ++k do
    for int i = 0; i < n; ++i do
        for int j = 0; j < n; ++j do
            array[i][j] = array[i][j] ||
                (array[i][k] && array[k][j]);
        end
    end
end
```

Floyd Warshall

107 Iterationen später ...

Floyd Warshall

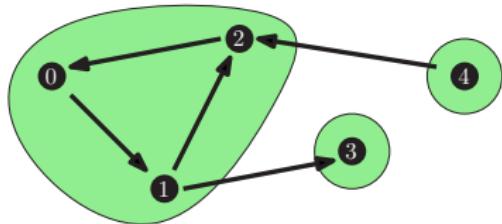


```
for int k = 0; k < n; ++k do
    for int i = 0; i < n; ++i do
        for int j = 0; j < n; ++j do
            array[i][j] = array[i][j] ||
                (array[i][k] && array[k][j]);
        end
    end
end
```

Floyd Warshall und SCC

- Transitiver Hülle einer SCC ist ein vollständiger Graph
- Betrachte Schrumpfgraphen: Floyd Warshall in $\#SCC^3$
- Deutlich Schneller falls $\#SCC < n$

Floyd Warshall und SCC

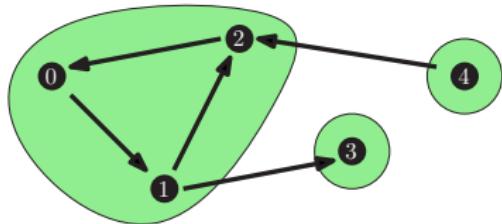


		nach	
		3	4
von	{0,1,2}	3	4
		3	4
3		0	0
4		0	0

■ erreichbar

```
for int k = 0; k < n; ++k do
    for int i = 0; i < n; ++i do
        for int j = 0; j < n; ++j do
            array[i][j] = array[i][j] ||
                (array[i][k] && array[k][j]);
        end
    end
end
```

Floyd Warshall und SCC

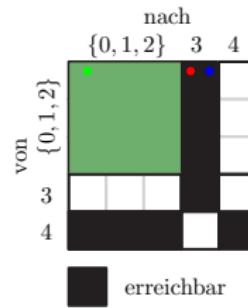
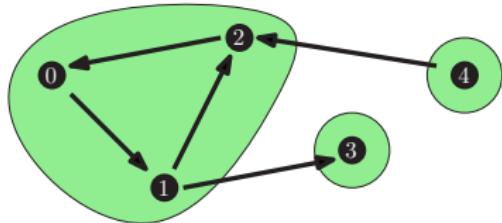


		nach	
		3	4
von	{0,1,2}	3	4
		■	

■ erreichbar

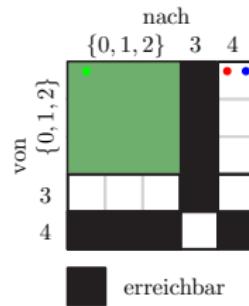
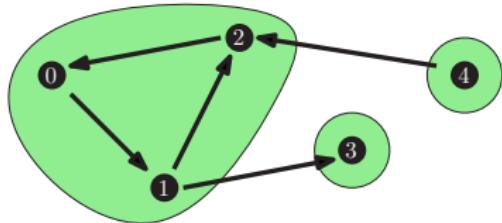
```
for int k = 0; k < n; ++k do
    for int i = 0; i < n; ++i do
        for int j = 0; j < n; ++j do
            array[i][j] = array[i][j] ||
                (array[i][k] && array[k][j]);
        end
    end
end
```

Floyd Warshall und SCC



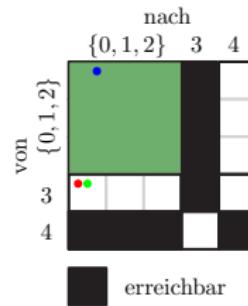
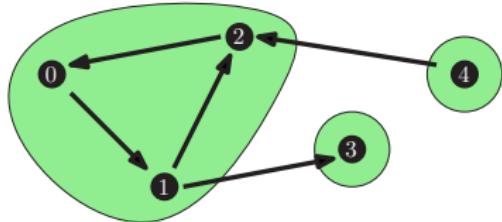
```
for int k = 0; k < n; ++k do
    for int i = 0; i < n; ++i do
        for int j = 0; j < n; ++j do
            array[i][j] = array[i][j] ||
                (array[i][k] && array[k][j]);
        end
    end
end
```

Floyd Warshall und SCC



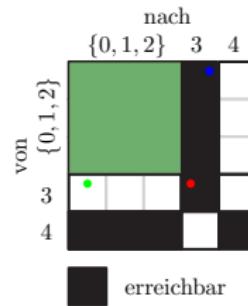
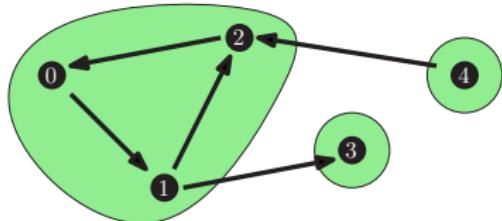
```
for int k = 0; k < n; ++k do
    for int i = 0; i < n; ++i do
        for int j = 0; j < n; ++j do
            array[i][j] = array[i][j] ||
                (array[i][k] && array[k][j]);
        end
    end
end
```

Floyd Warshall und SCC



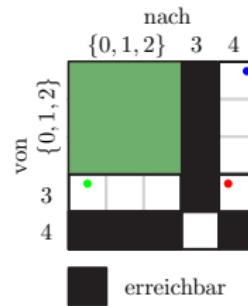
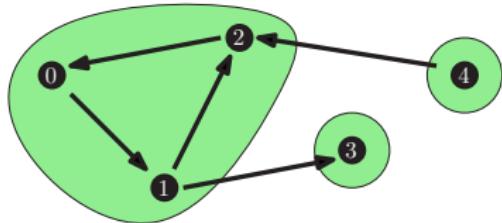
```
for int k = 0; k < n; ++k do
    for int i = 0; i < n; ++i do
        for int j = 0; j < n; ++j do
            array[i][j] = array[i][j] ||
                (array[i][k] && array[k][j]);
        end
    end
end
```

Floyd Warshall und SCC



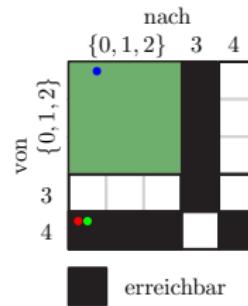
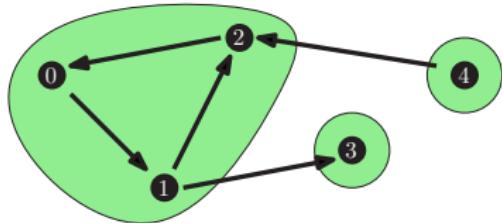
```
for int k = 0; k < n; ++k do
    for int i = 0; i < n; ++i do
        for int j = 0; j < n; ++j do
            array[i][j] = array[i][j] ||
                (array[i][k] && array[k][j]);
        end
    end
end
```

Floyd Warshall und SCC



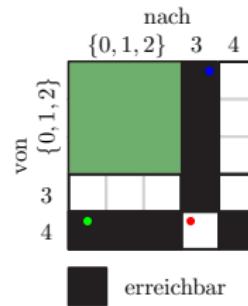
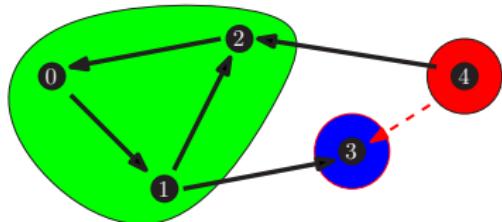
```
for int k = 0; k < n; ++k do
    for int i = 0; i < n; ++i do
        for int j = 0; j < n; ++j do
            array[i][j] = array[i][j] ||
                (array[i][k] && array[k][j]);
        end
    end
end
```

Floyd Warshall und SCC



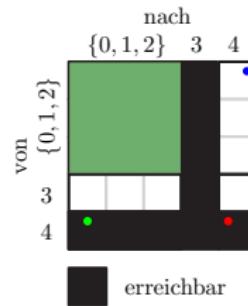
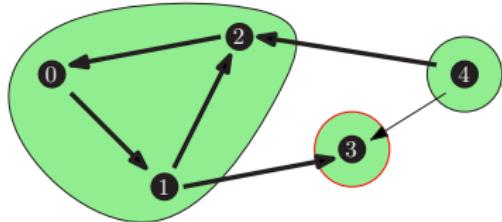
```
for int k = 0; k < n; ++k do
    for int i = 0; i < n; ++i do
        for int j = 0; j < n; ++j do
            array[i][j] = array[i][j] ||
                (array[i][k] && array[k][j]);
        end
    end
end
```

Floyd Warshall und SCC



```
for int k = 0; k < n; ++k do
    for int i = 0; i < n; ++i do
        for int j = 0; j < n; ++j do
            array[i][j] = array[i][j] ||
                (array[i][k] && array[k][j]);
        end
    end
end
```

Floyd Warshall und SCC



```
for int k = 0; k < n; ++k do
    for int i = 0; i < n; ++i do
        for int j = 0; j < n; ++j do
            array[i][j] = array[i][j] ||
                (array[i][k] && array[k][j]);
        end
    end
end
```

Floyd Warshall und SCC

- SCC-Algorithmus nicht nur eleganter Algorithmus . . .
- . . . sondern auch Tool für bessere Algorithmen
- Linearzeit Algorithmus kann teurere Algorithmen deutlich beschleunigen

Ende!



Feierabend!