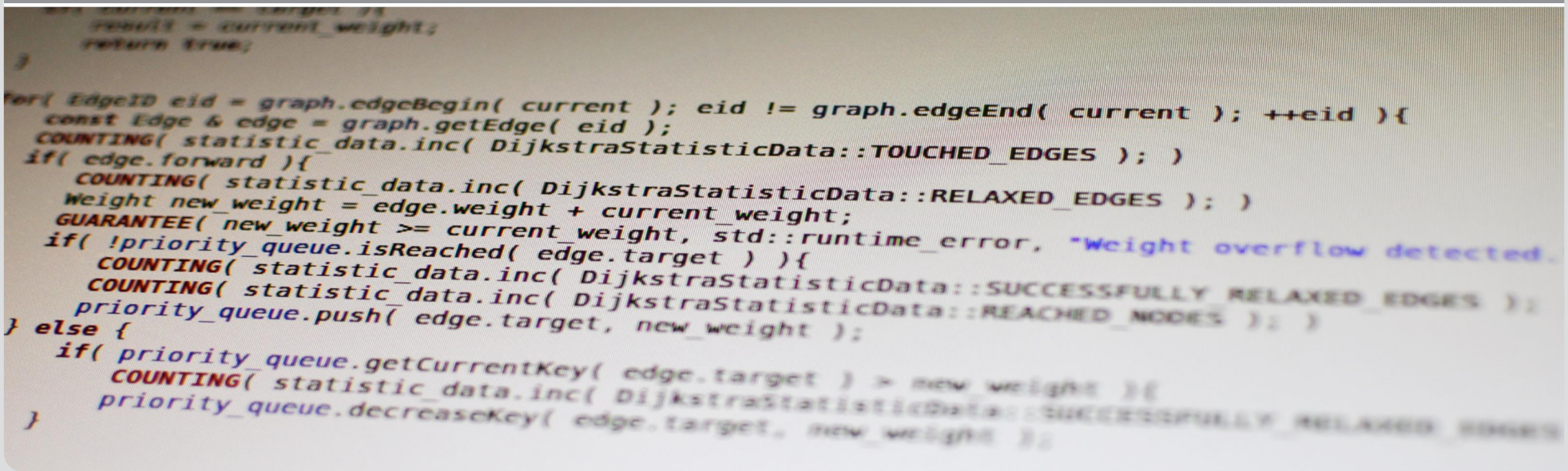
Karlsruher Institut für Technologie

# 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

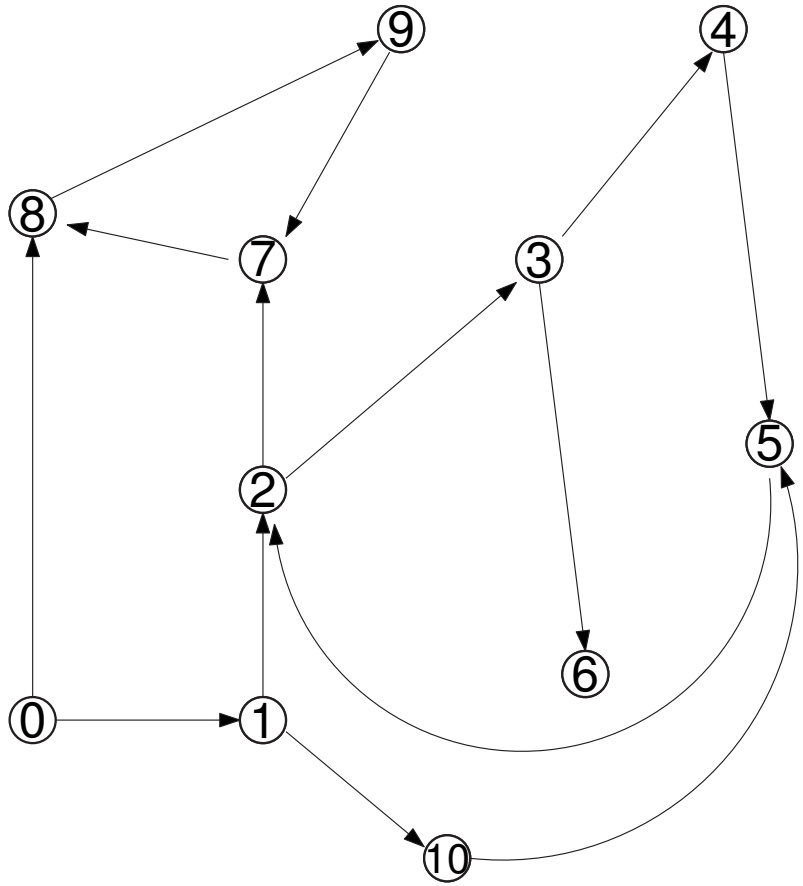


# Themen

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

# Starke Zusammenhangskomponenten

# SCC (Wiederholung)




| oNodes | oReps |
|---|---|
| | |

# SCC (Wiederholung)

**Invariante 1**

Kein Kanten von geschlossenen in offene Komponenten.

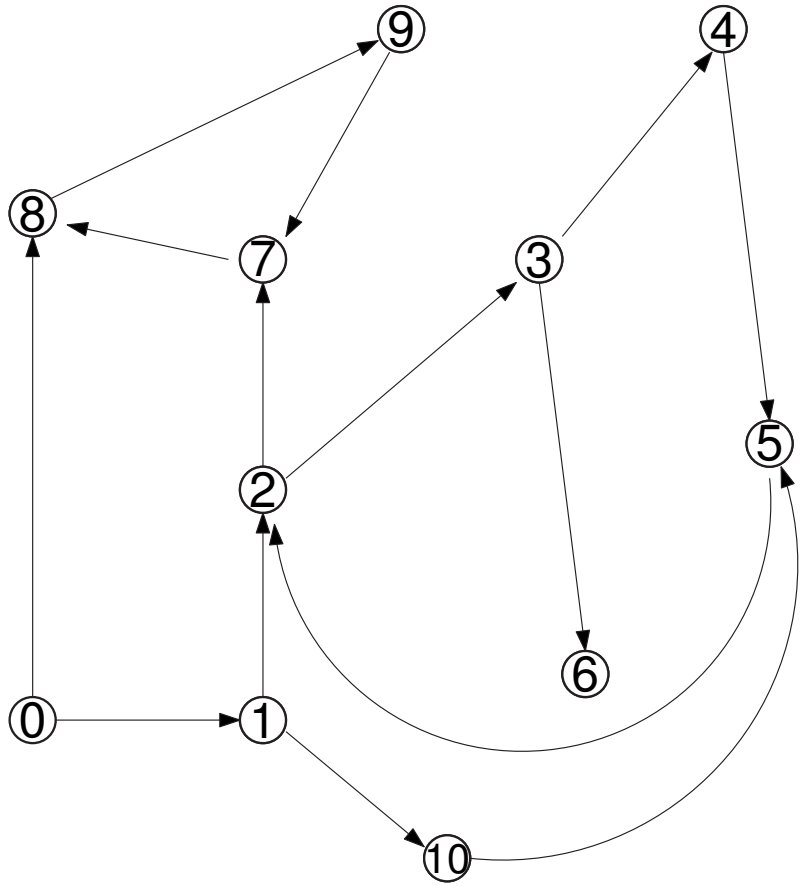
**Invariante 2**

Offene Komponenten liegen auf Pfad.

**Invariante 3**

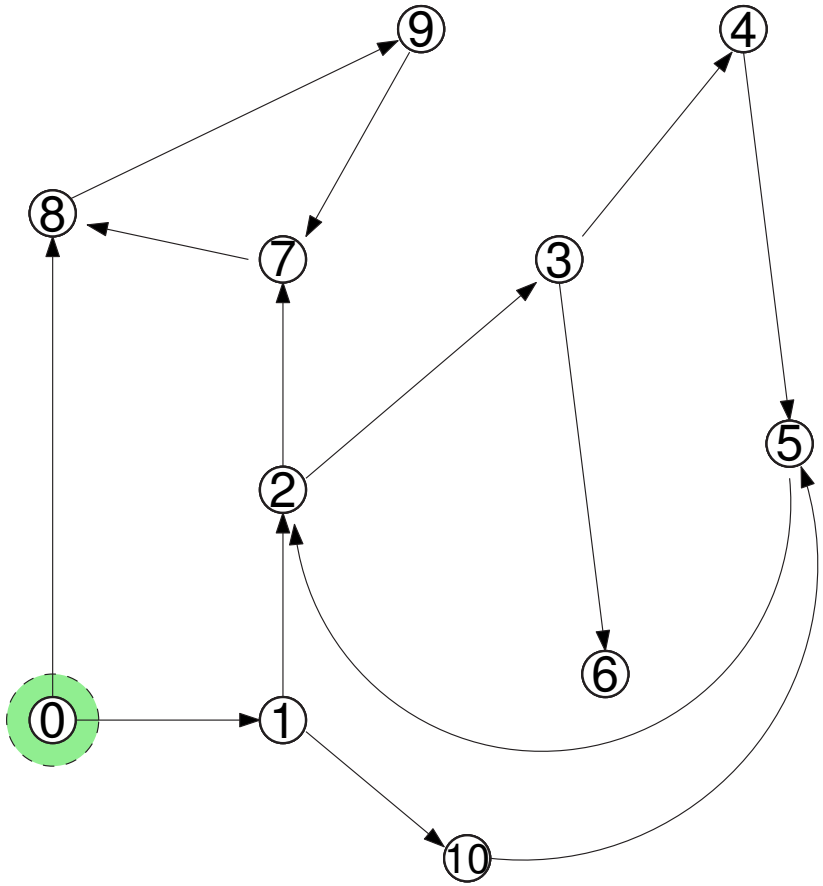
Repräsentanten partitionieren offene Komponenten bzgl. dfsNum.

# SCC (Wiederholung)



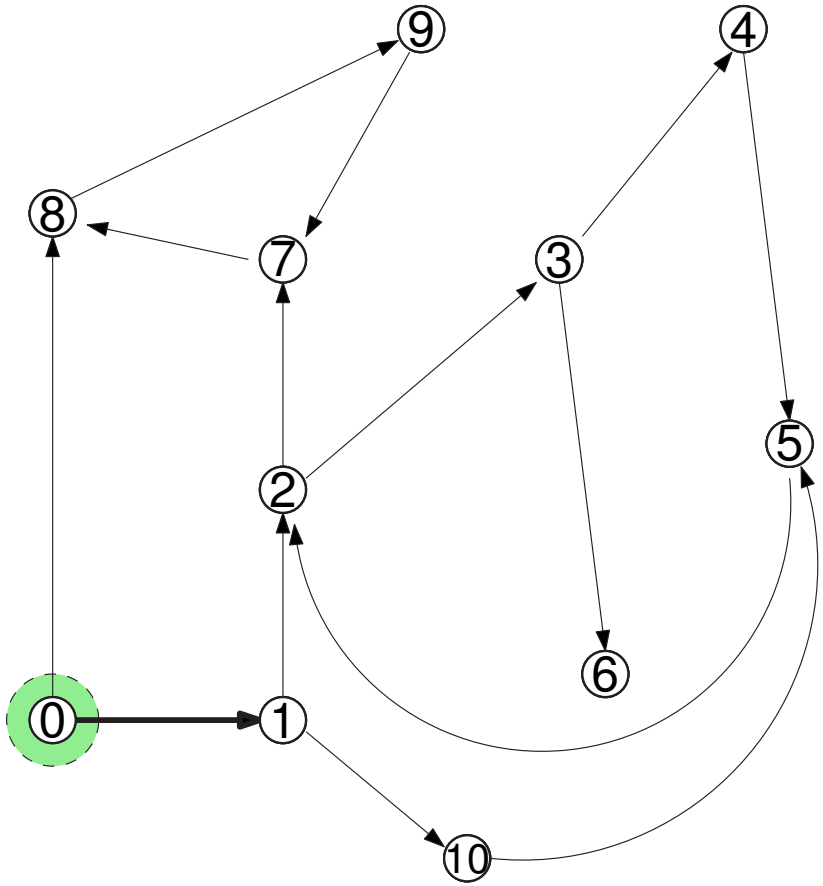

| oNodes | oReps |
|---|---|
| | |

# SCC (Wiederholung)



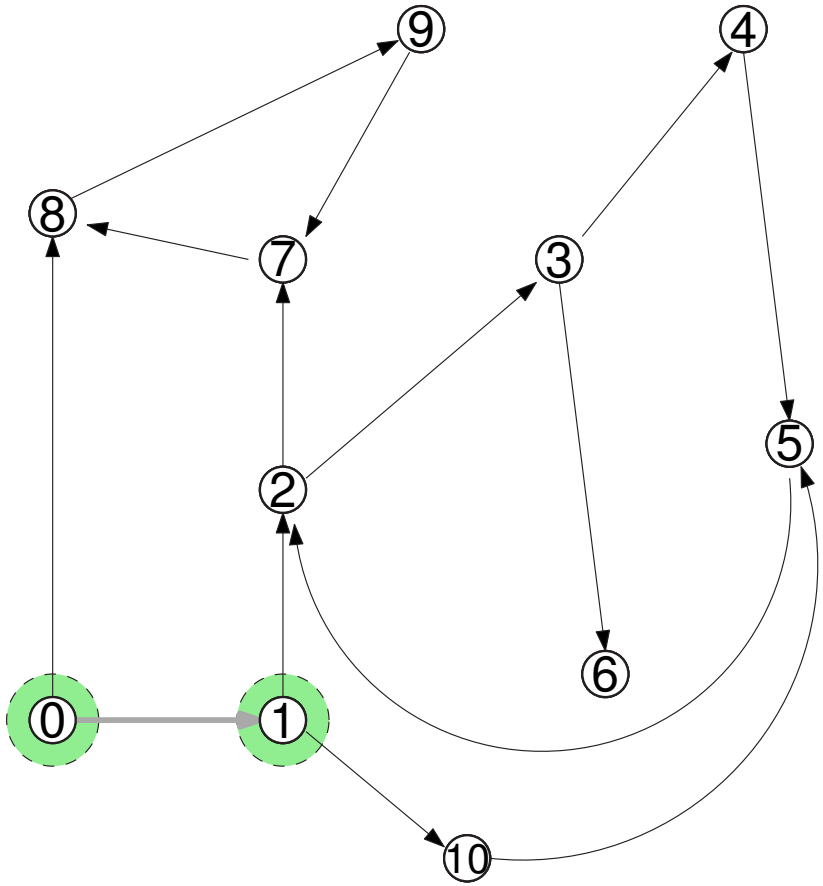

| oNodes | oReps |
|---|---|
| 0 | 0 |

# SCC (Wiederholung)



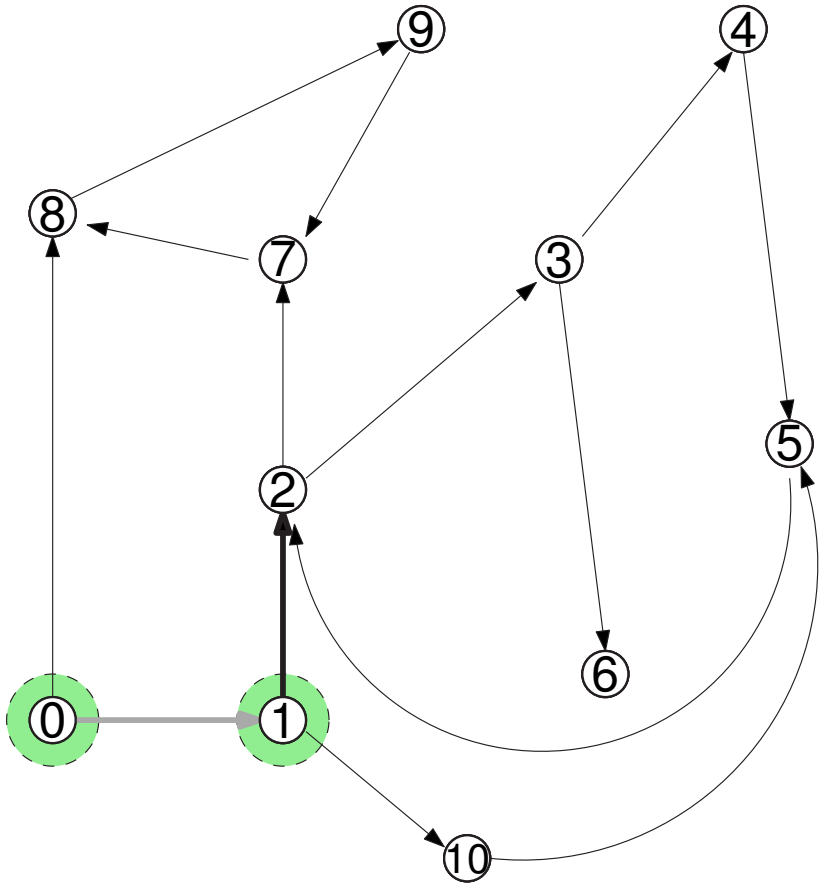

| oNodes | oReps |
|---|---|
| 0 | 0 |

# SCC (Wiederholung)



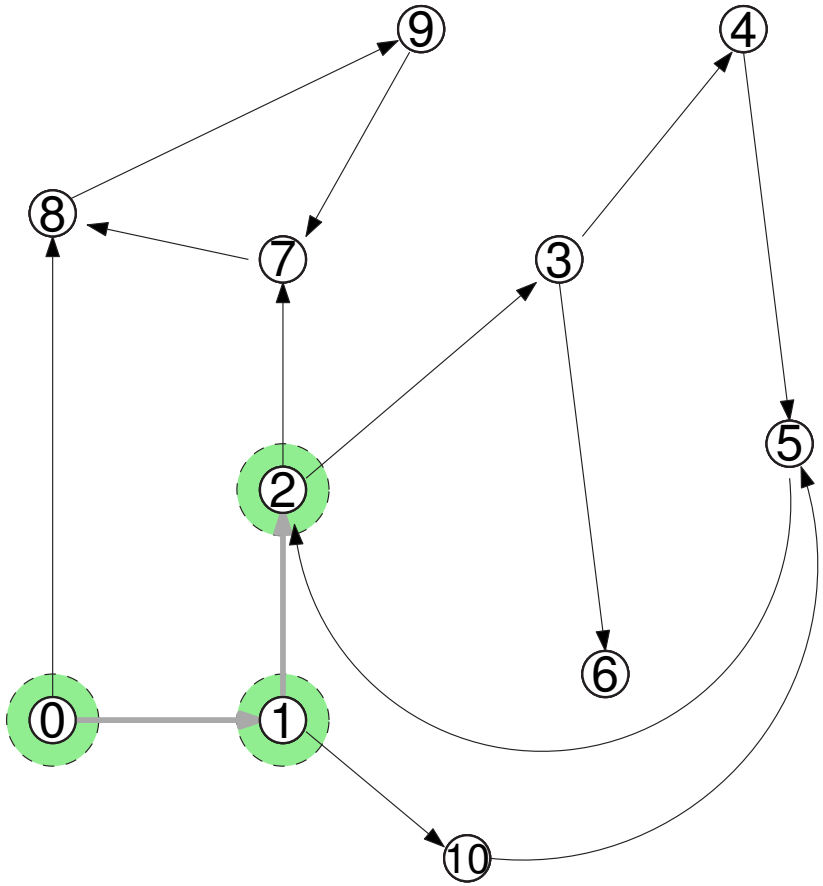

| oNodes | oReps |
|---|---|
| 0 | 0 |
| 1 | 1 |

# SCC (Wiederholung)



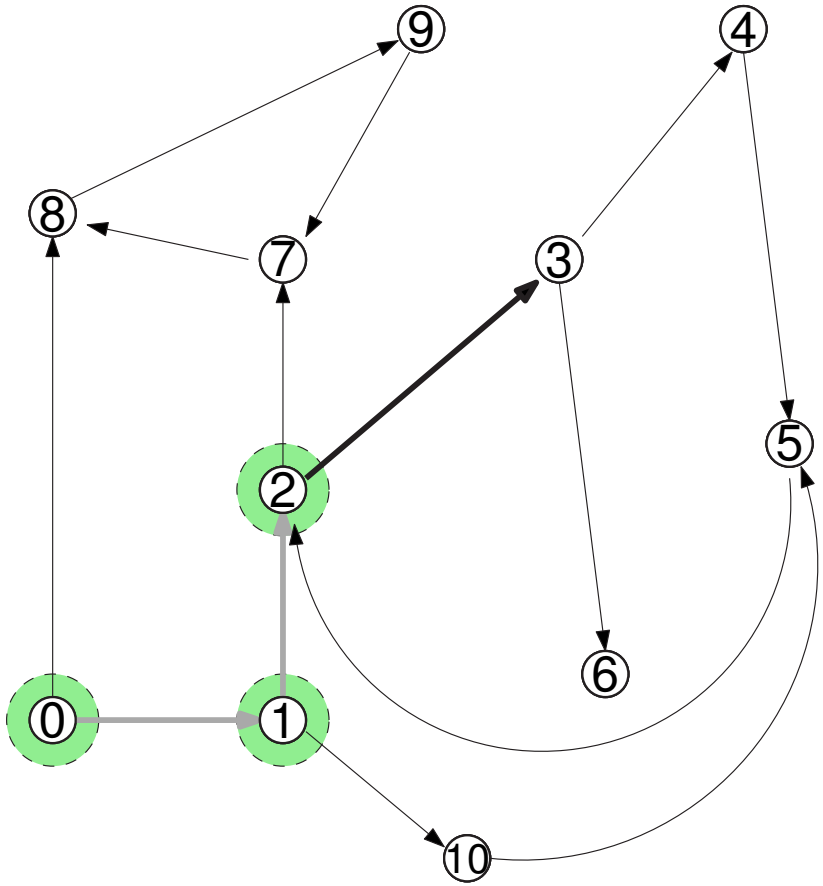

| oNodes | oReps |
|---|---|
| 0 | 0 |
| 1 | 1 |

# SCC (Wiederholung)



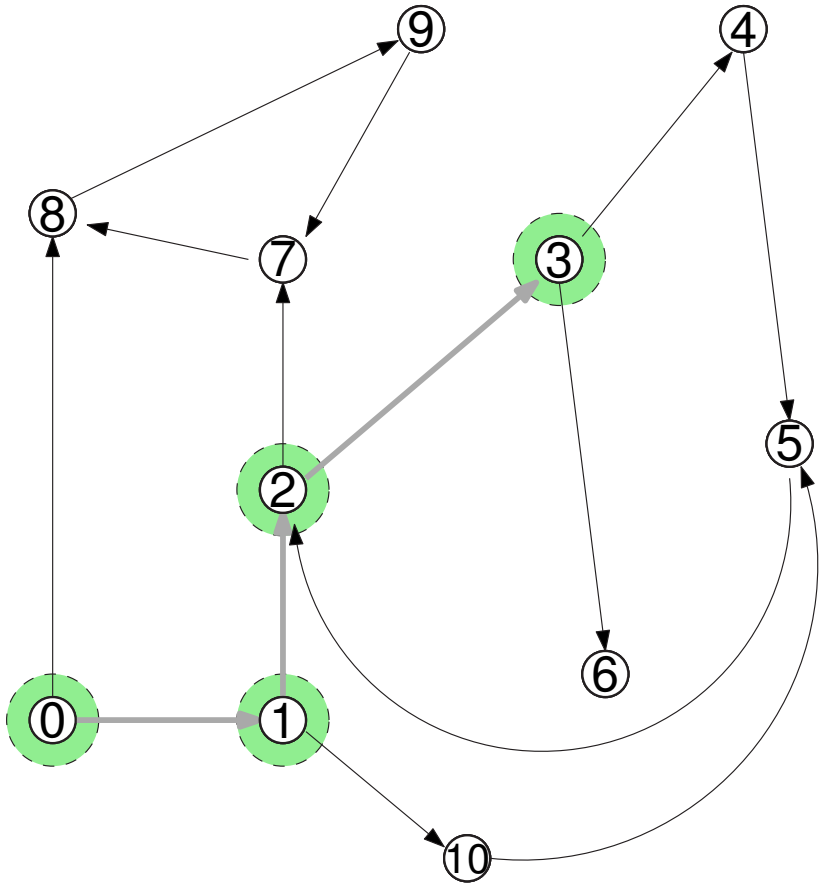

| oNodes | oReps |
|---|---|
| 0 | 0 |
| 1 | 1 |
| 2 | 2 |

# SCC (Wiederholung)



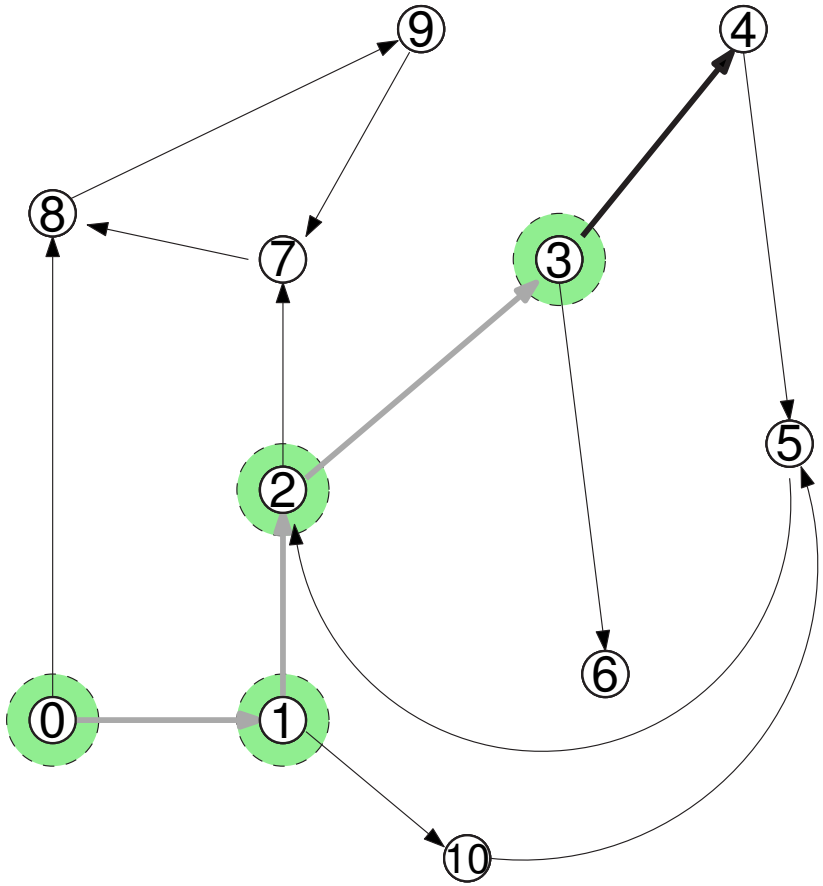

| oNodes | oReps |
|---|---|
| 0 | 0 |
| 1 | 1 |
| 2 | 2 |

# SCC (Wiederholung)



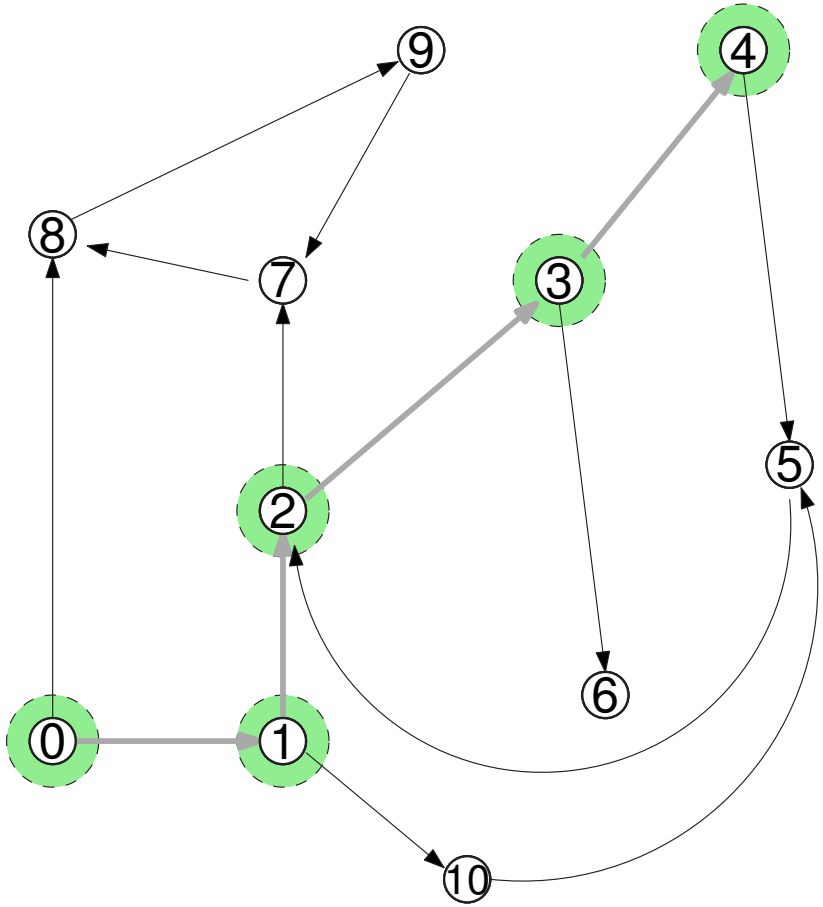

| oNodes | oReps |
|---|---|
| 0 | 0 |
| 1 | 1 |
| 2 | 2 |
| 3 | 3 |

# SCC (Wiederholung)



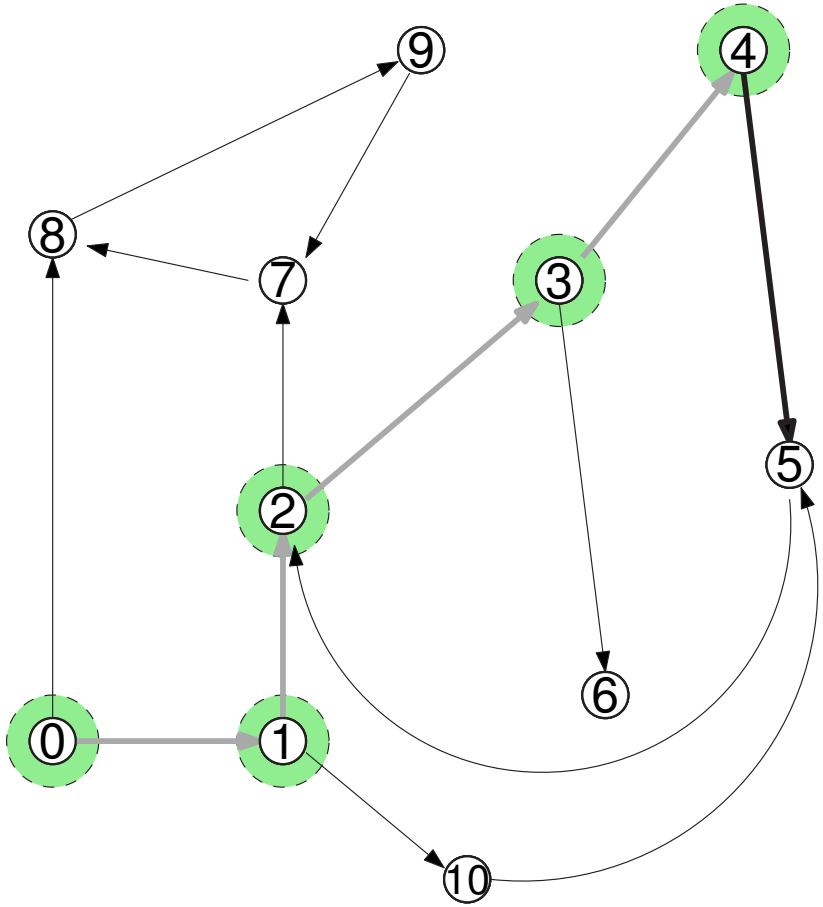

| oNodes | oReps |
|---|---|
| 0 | 0 |
| 1 | 1 |
| 2 | 2 |
| 3 | 3 |

# SCC (Wiederholung)



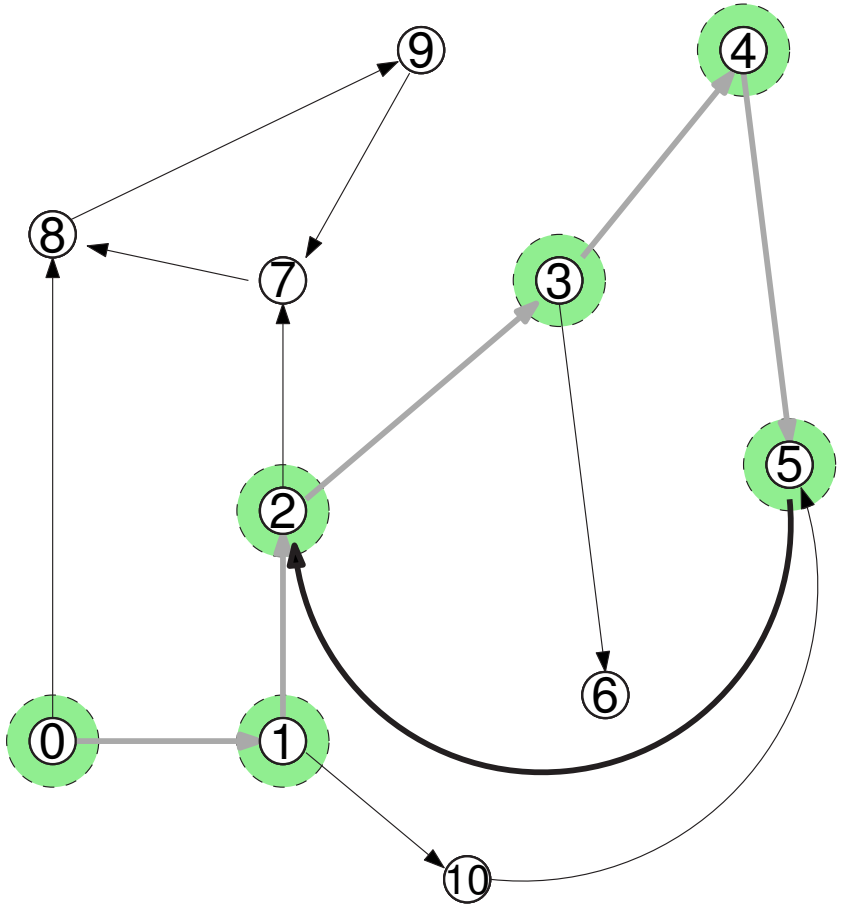

| oNodes | oReps |
|---|---|
| 0 | 0 |
| 1 | 1 |
| 2 | 2 |
| 3 | 3 |
| 4 | 4 |

# SCC (Wiederholung)



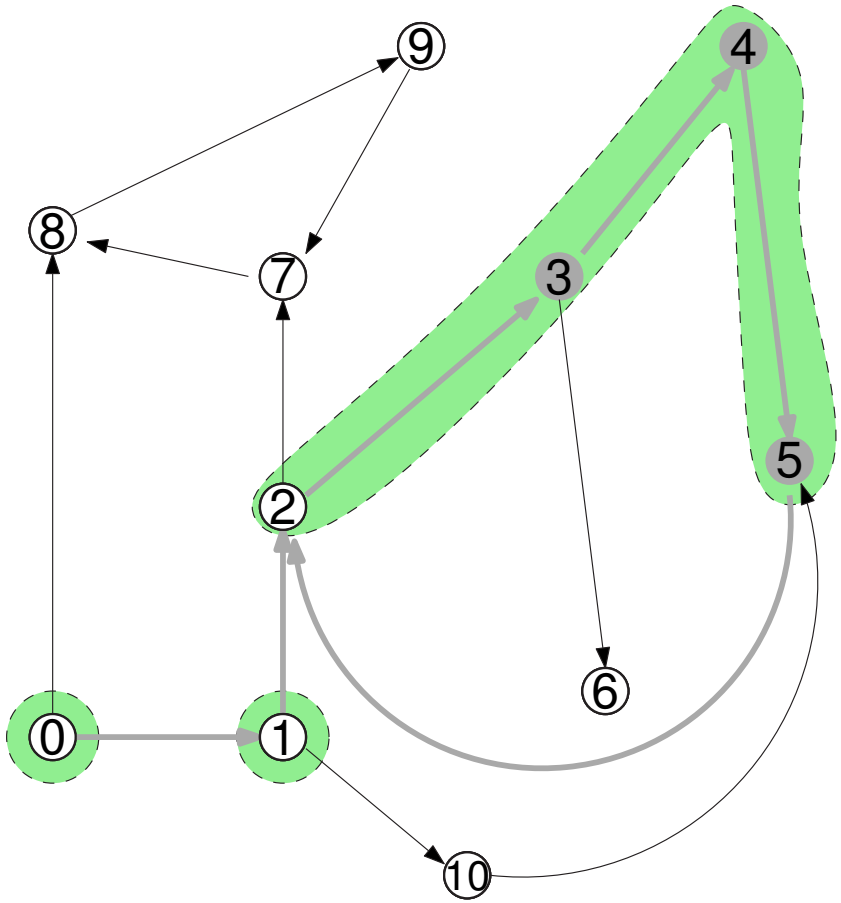

| oNodes | oReps |
|---|---|
| 0 | 0 |
| 1 | 1 |
| 2 | 2 |
| 3 | 3 |
| 4 | 4 |

# SCC (Wiederholung)




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

# SCC (Wiederholung)



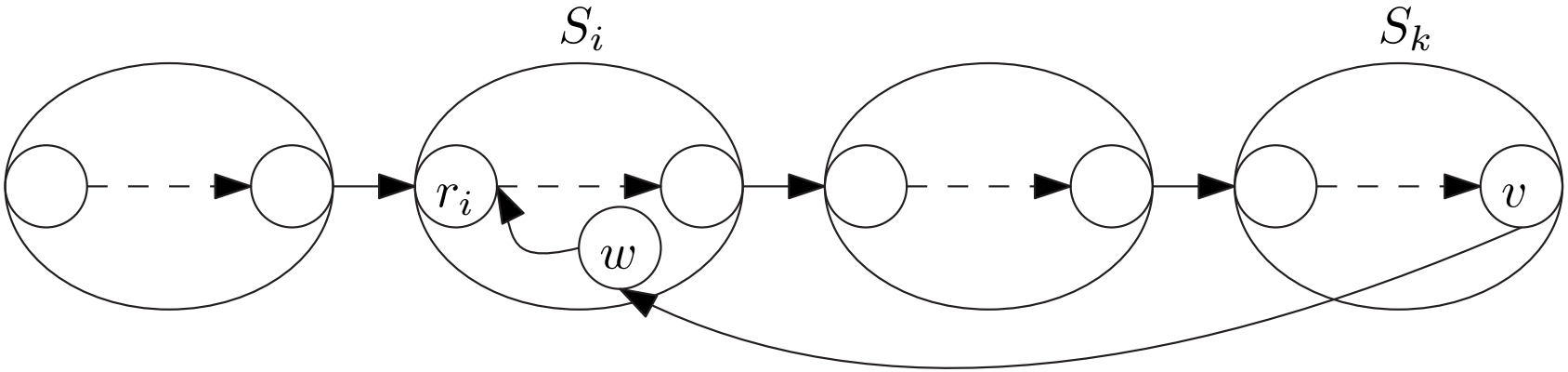

| oNodes | oReps |
|---|---|
| 0 | 0 |
| 1 | 1 |
| 2 | 2 |
| 3 | |
| 4 | |
| 5 | |

# 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



# SCC (Wiederholung)

**Invariante 2**

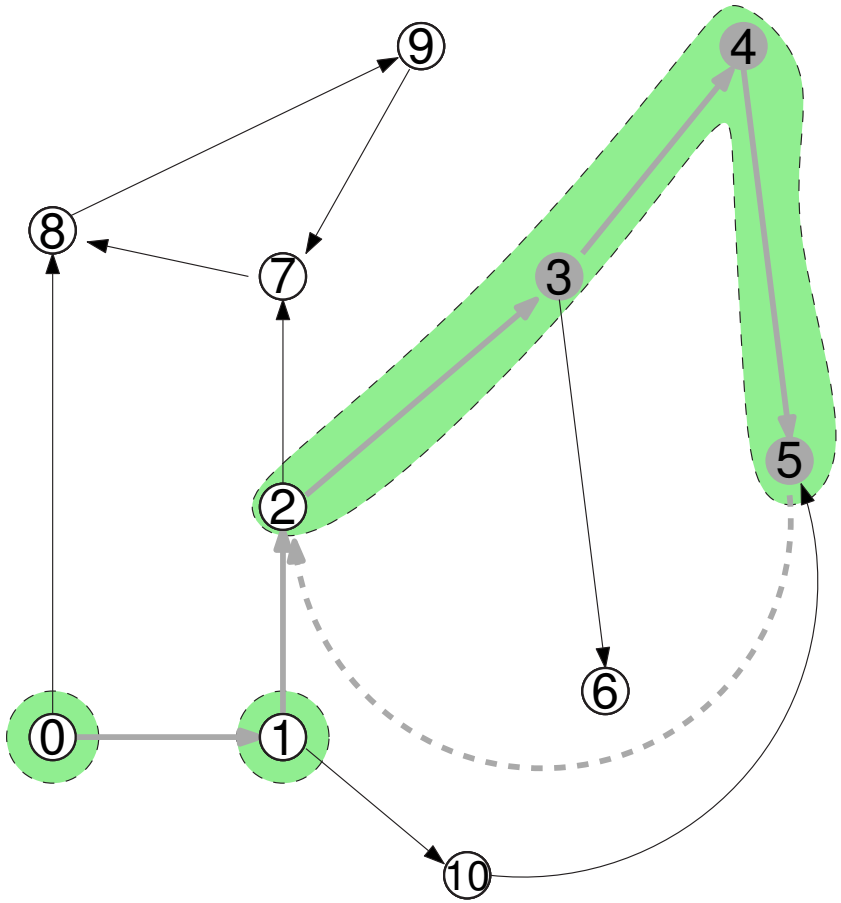
Offene Komponenten liegen auf Pfad.

- Repräsentant ist minimal auf Kreis

**Invariante 3**

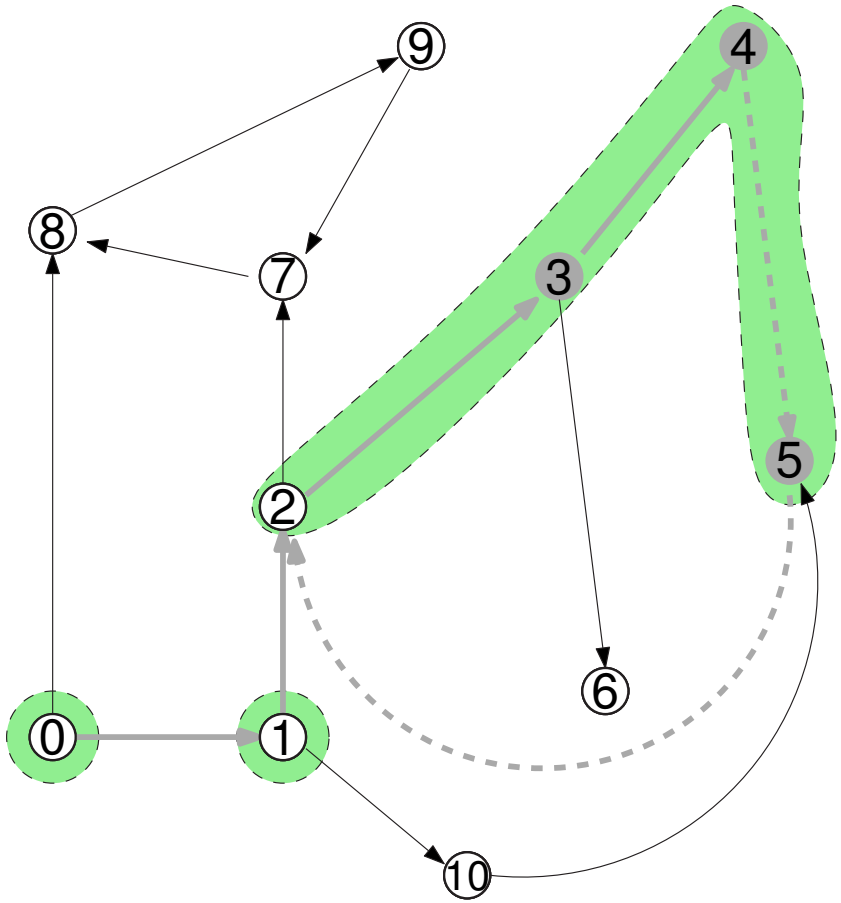
Repräsentanten partitionieren offene Komponenten bzgl. dfsNum.

# SCC (Wiederholung)



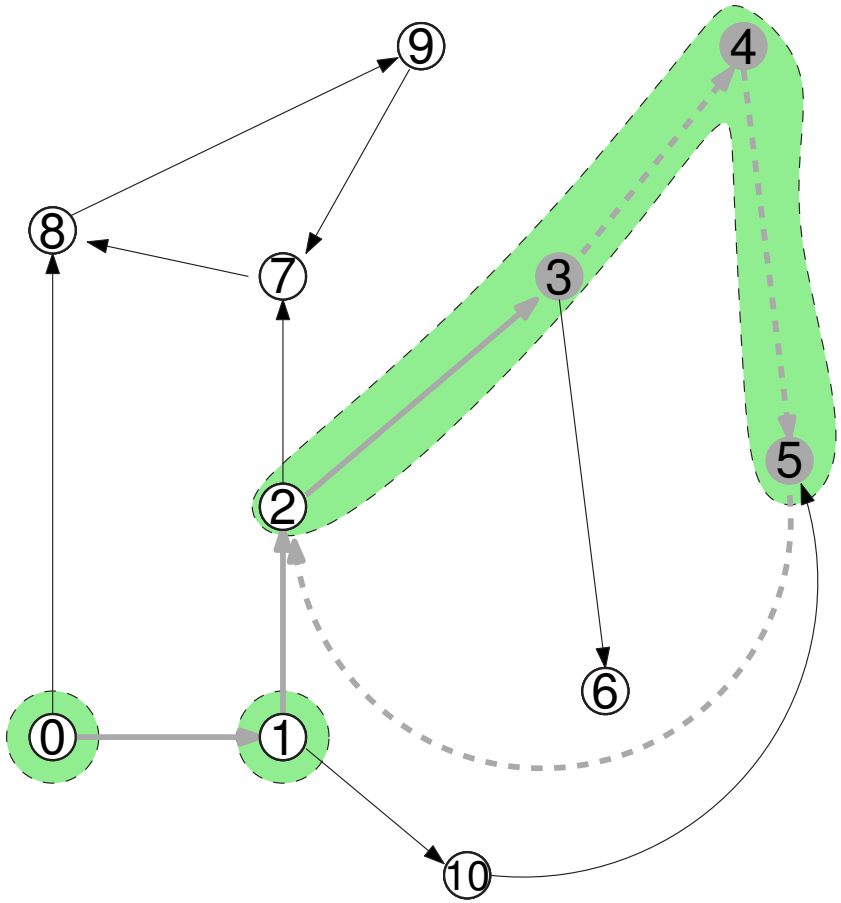

| oNodes | oReps |
|---|---|
| 0 | 0 |
| 1 | 1 |
| 2 | 2 |
| 3 | |
| 4 | |
| 5 | |

# SCC (Wiederholung)



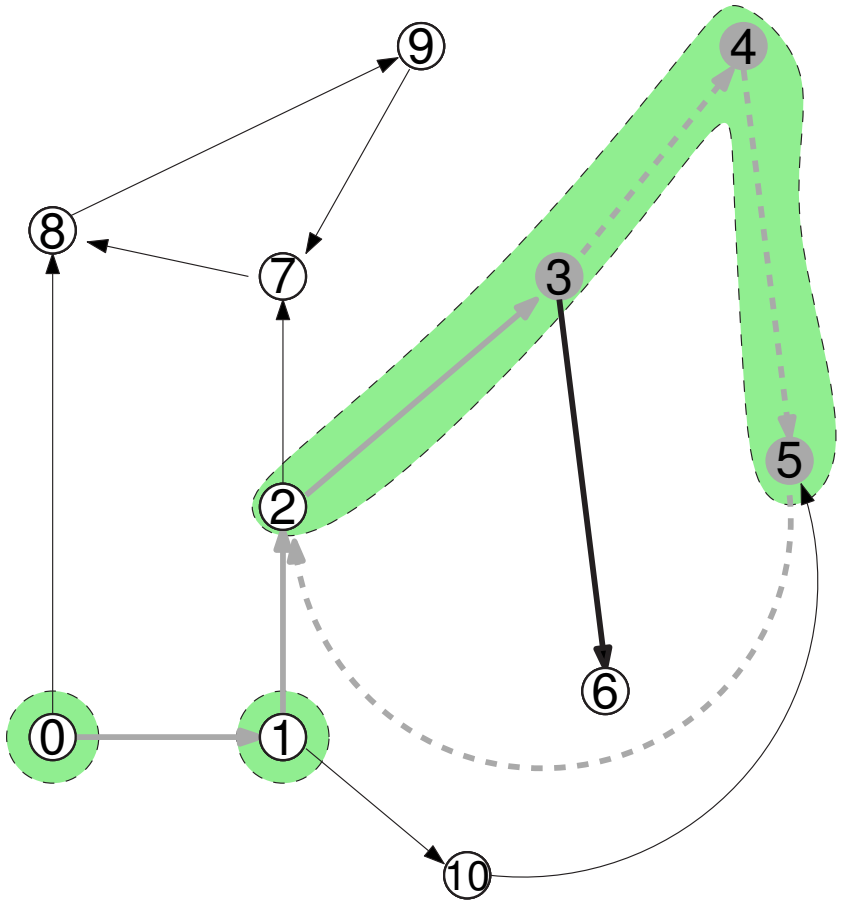

| oNodes | oReps |
|---|---|
| 0 | 0 |
| 1 | 1 |
| 2 | 2 |
| 3 | |
| 4 | |
| 5 | |

# SCC (Wiederholung)



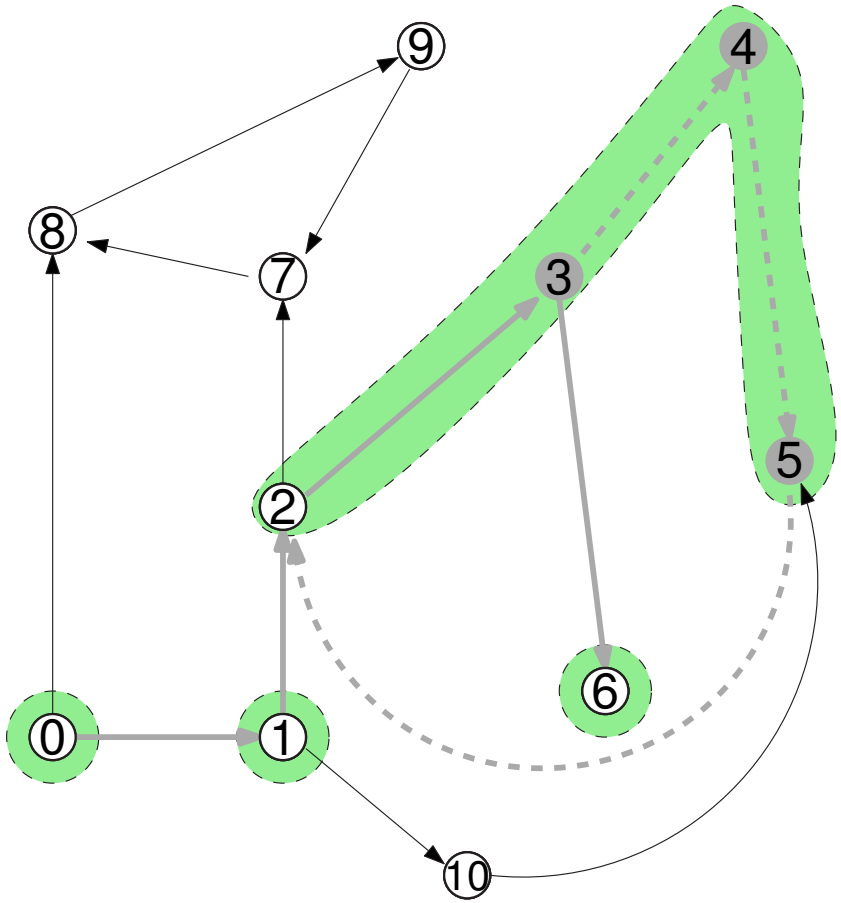

| oNodes | oReps |
|---|---|
| 0 | 0 |
| 1 | 1 |
| 2 | 2 |
| 3 | |
| 4 | |
| 5 | |

# SCC (Wiederholung)



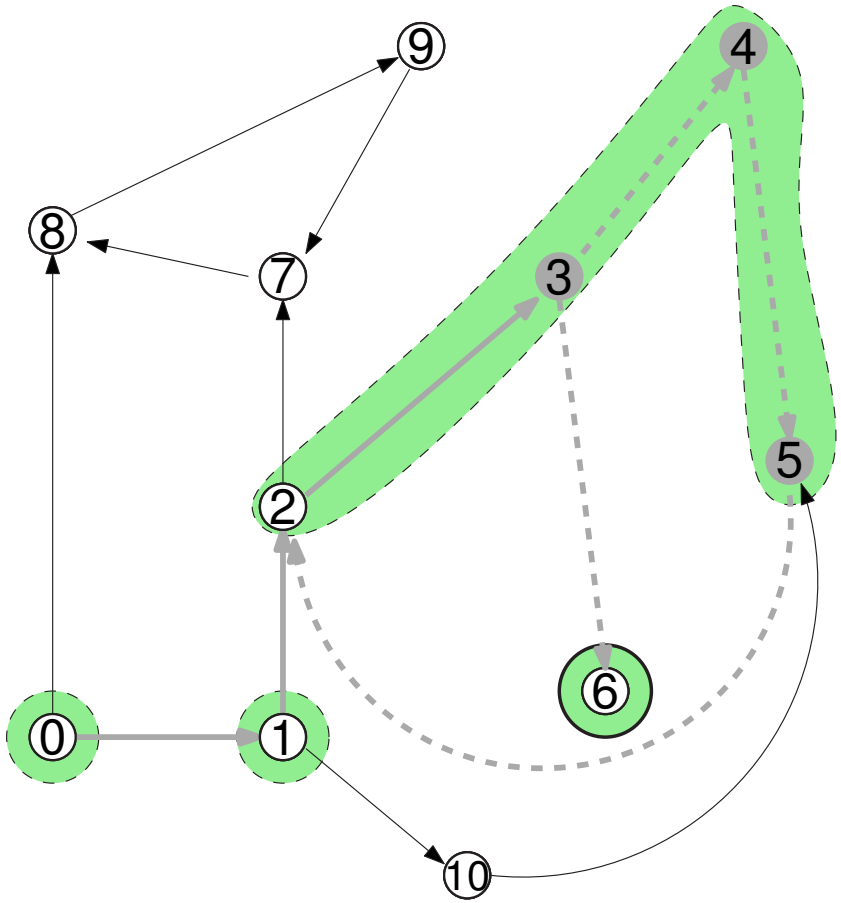

| oNodes | oReps |
|---|---|
| 0 | 0 |
| 1 | 1 |
| 2 | 2 |
| 3 | |
| 4 | |
| 5 | |

# SCC (Wiederholung)




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

# SCC (Wiederholung)




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

# 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.

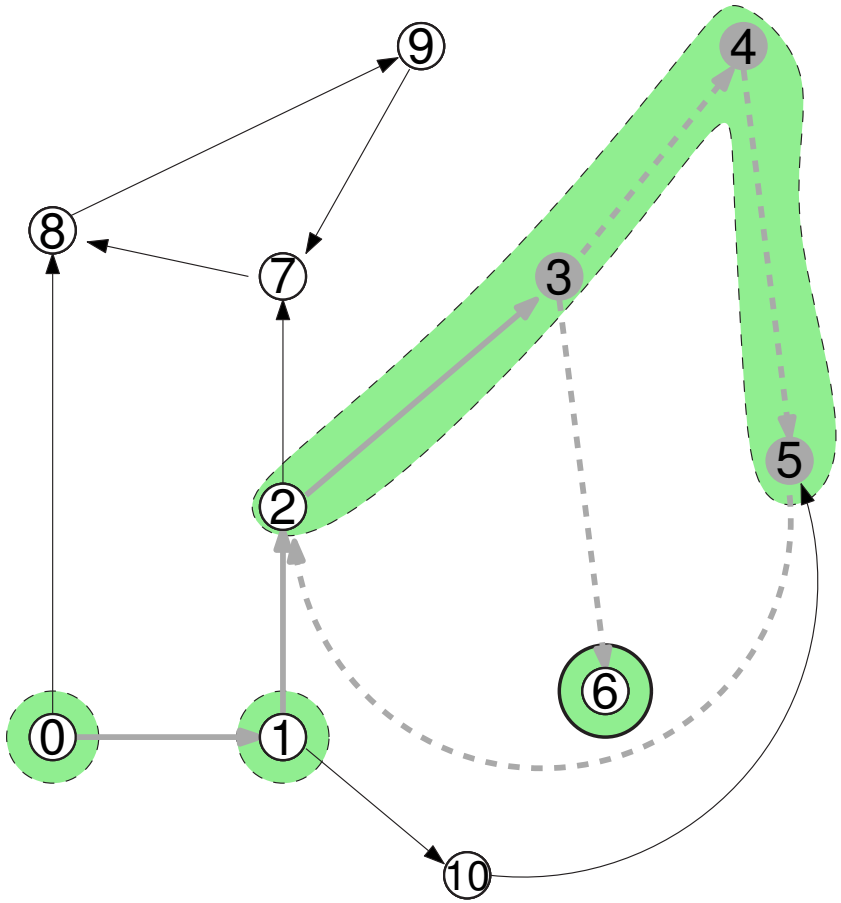
**Invariante 2**

Offene Komponenten liegen auf Pfad.

**Invariante 3**

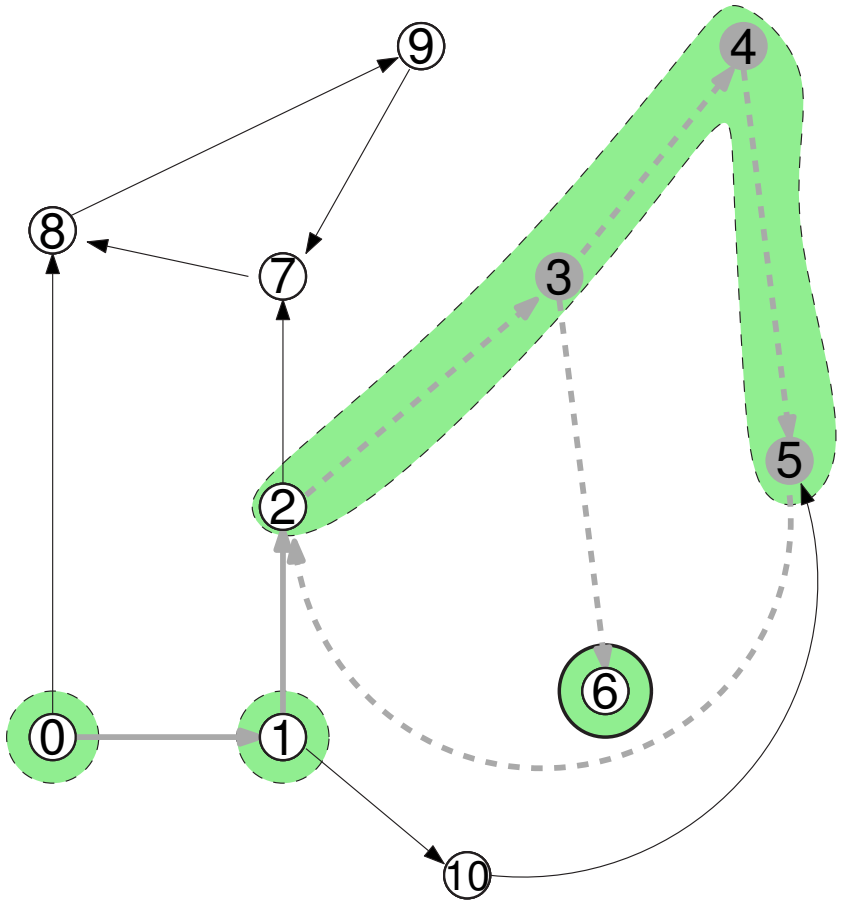
Repräsentanten partitionieren offene Komponenten bzgl. dfsNum.

# SCC (Wiederholung)



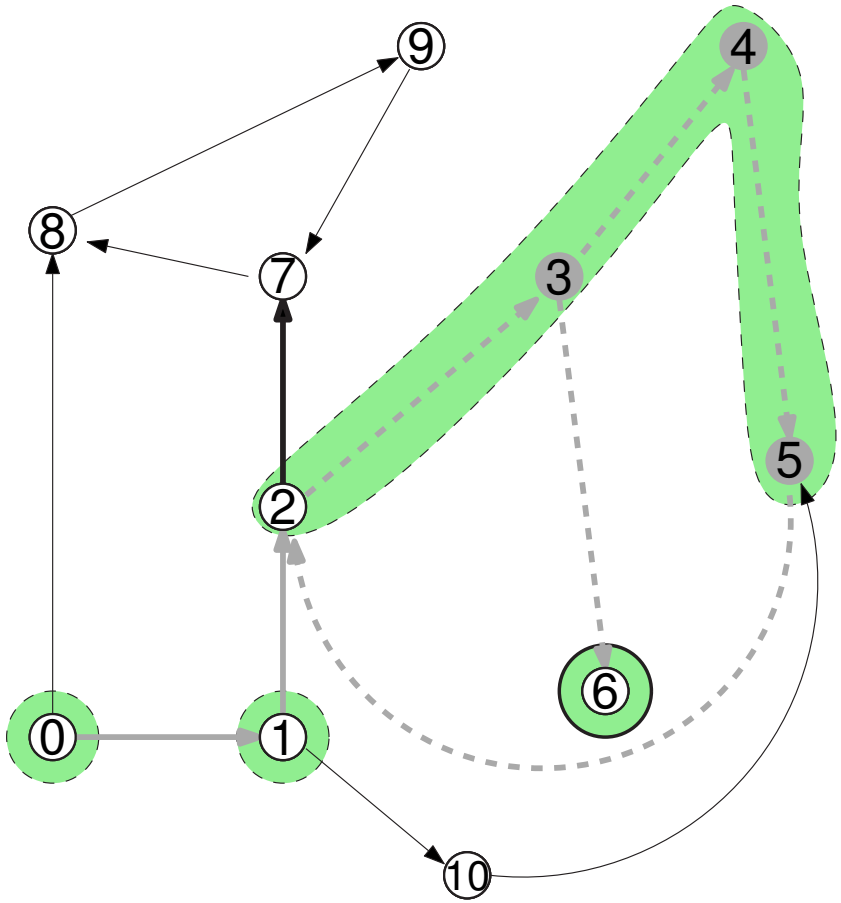

| oNodes | oReps |
|---|---|
| 0 | 0 |
| 1 | 1 |
| 2 | 2 |
| 3 | |
| 4 | |
| 5 | |

# SCC (Wiederholung)



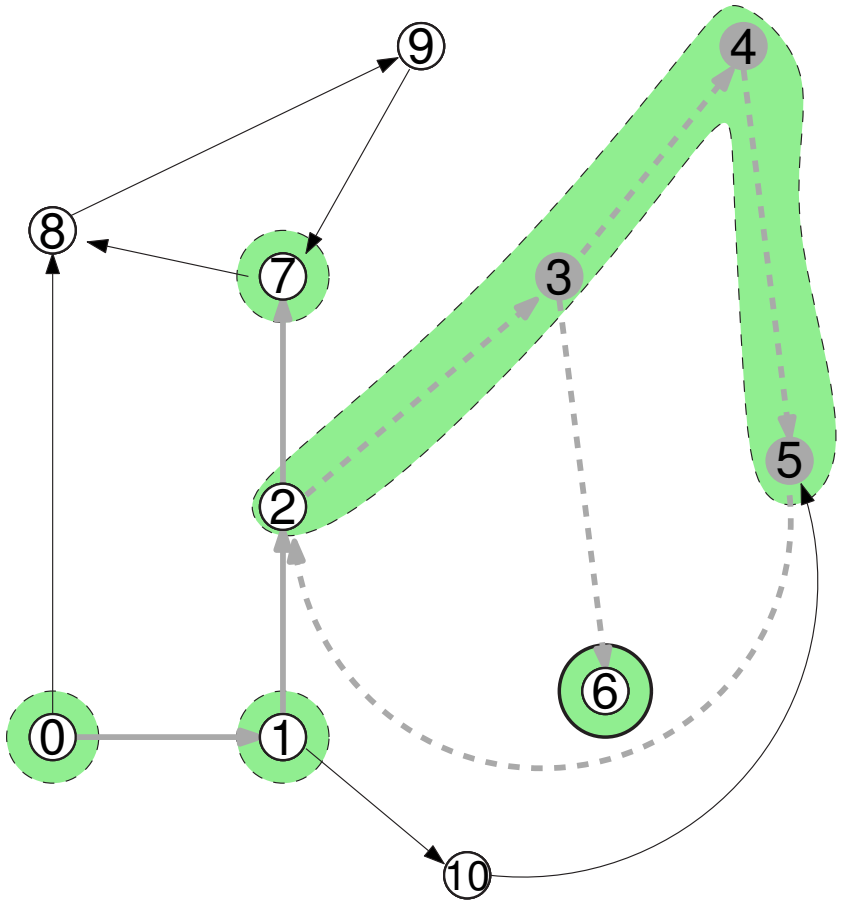

| oNodes | oReps |
|---|---|
| 0 | 0 |
| 1 | 1 |
| 2 | 2 |
| 3 | |
| 4 | |
| 5 | |

# SCC (Wiederholung)



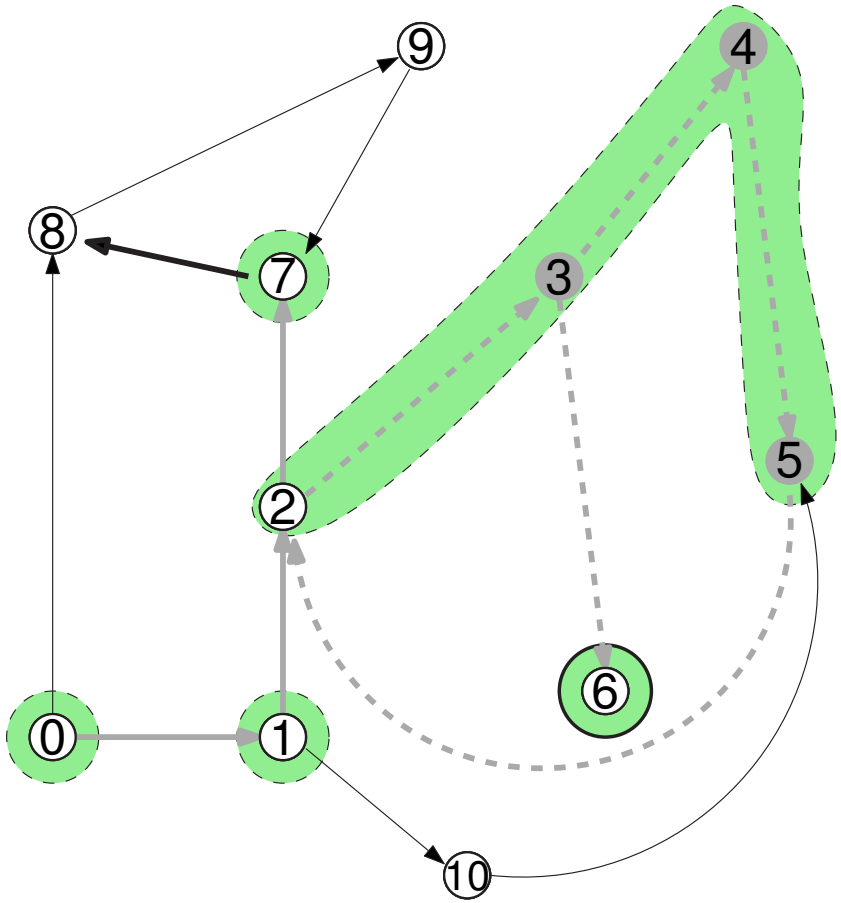

| oNodes | oReps |
|---|---|
| 0 | 0 |
| 1 | 1 |
| 2 | 2 |
| 3 | |
| 4 | |
| 5 | |

# SCC (Wiederholung)



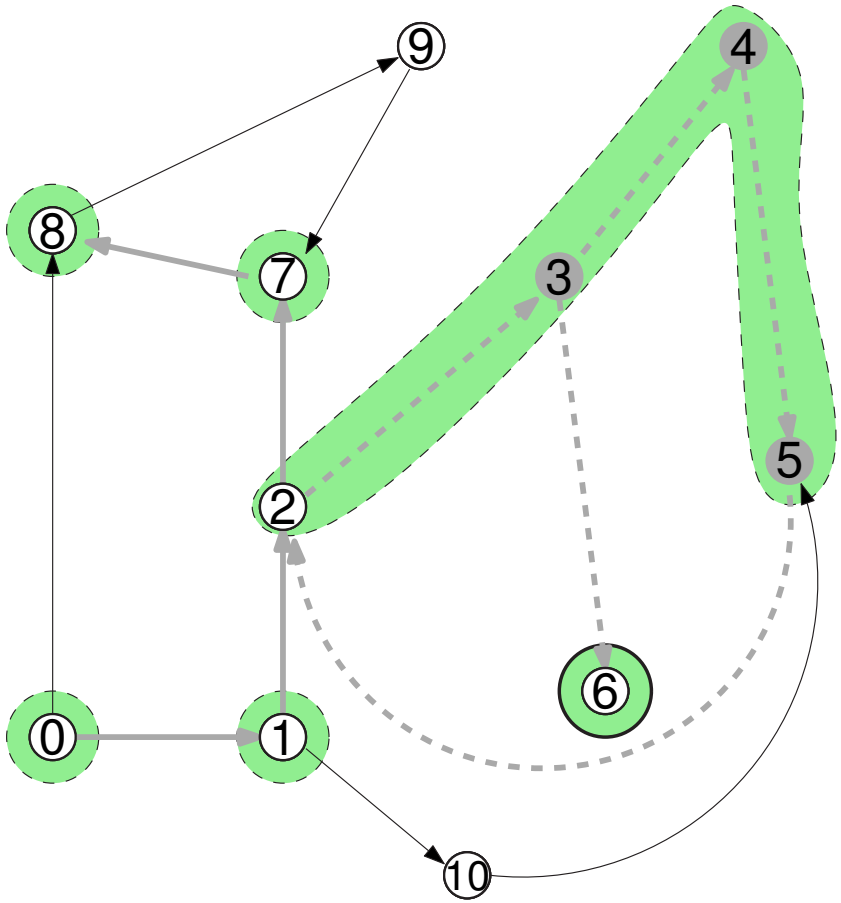

| oNodes | oReps |
|---|---|
| 0 | 0 |
| 1 | 1 |
| 2 | 2 |
| 3 | |
| 4 | |
| 5 | |
| 7 | 7 |

# SCC (Wiederholung)



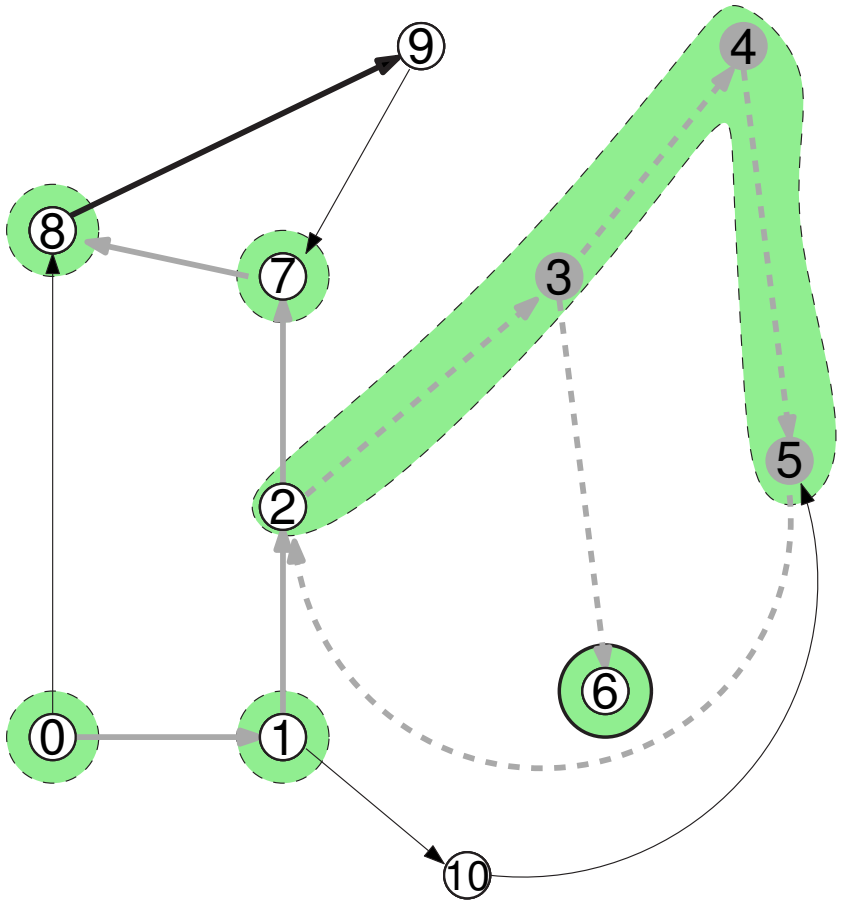

| oNodes | oReps |
|---|---|
| 0 | 0 |
| 1 | 1 |
| 2 | 2 |
| 3 | |
| 4 | |
| 5 | |
| 7 | 7 |

# SCC (Wiederholung)



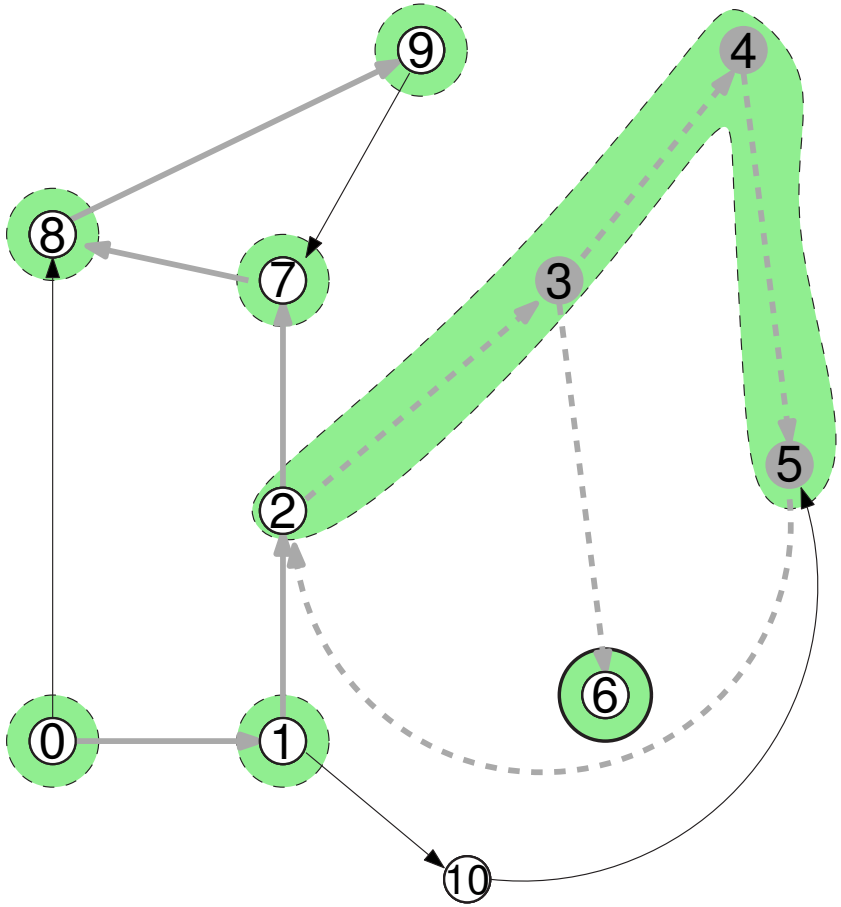

| oNodes | oReps |
|---|---|
| 0 | 0 |
| 1 | 1 |
| 2 | 2 |
| 3 | |
| 4 | |
| 5 | |
| 7 | 7 |
| 8 | 8 |

# SCC (Wiederholung)



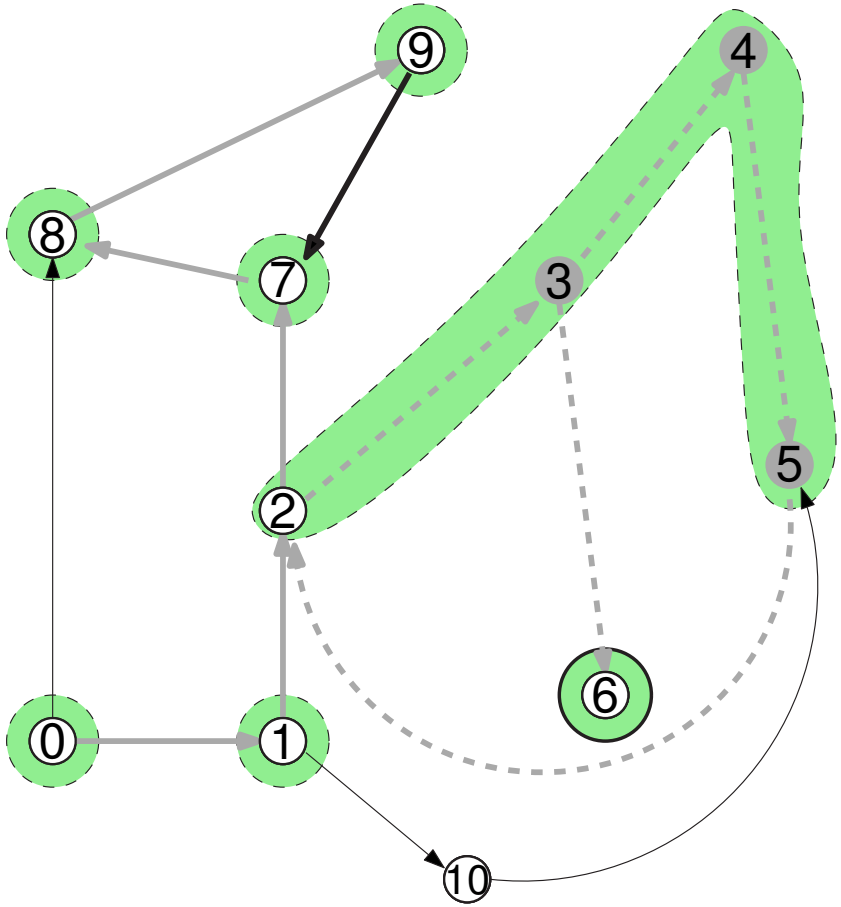

| oNodes | oReps |
|---|---|
| 0 | 0 |
| 1 | 1 |
| 2 | 2 |
| 3 | |
| 4 | |
| 5 | |
| 7 | 7 |
| 8 | 8 |

# SCC (Wiederholung)



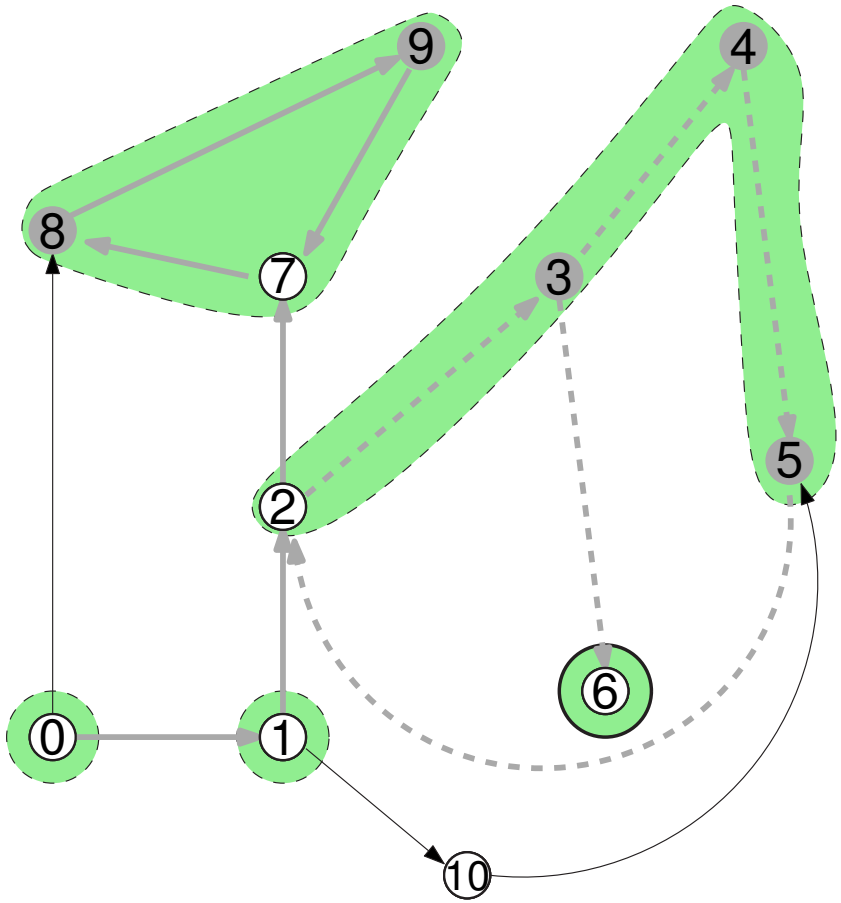

| oNodes | oReps |
|---|---|
| 0 | 0 |
| 1 | 1 |
| 2 | 2 |
| 3 | |
| 4 | |
| 5 | |
| 7 | 7 |
| 8 | 8 |
| 9 | 9 |

# SCC (Wiederholung)



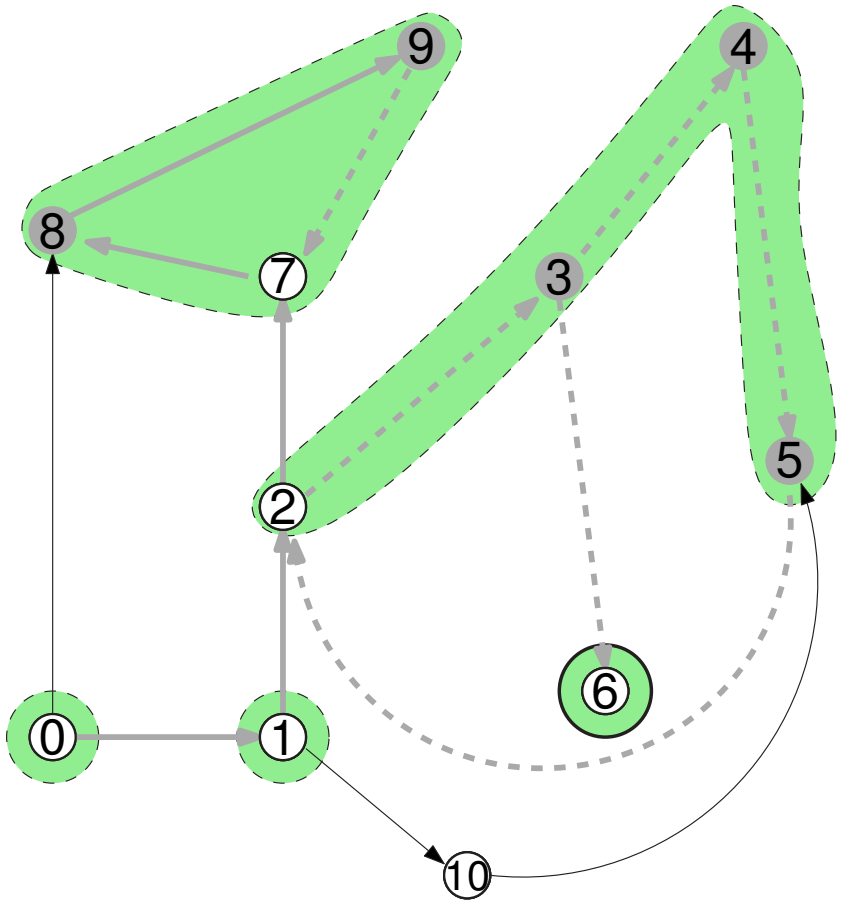

| oNodes | oReps |
|---|---|
| 0 | 0 |
| 1 | 1 |
| 2 | 2 |
| 3 | |
| 4 | |
| 5 | |
| 7 | 7 |
| 8 | 8 |
| 9 | 9 |

# SCC (Wiederholung)



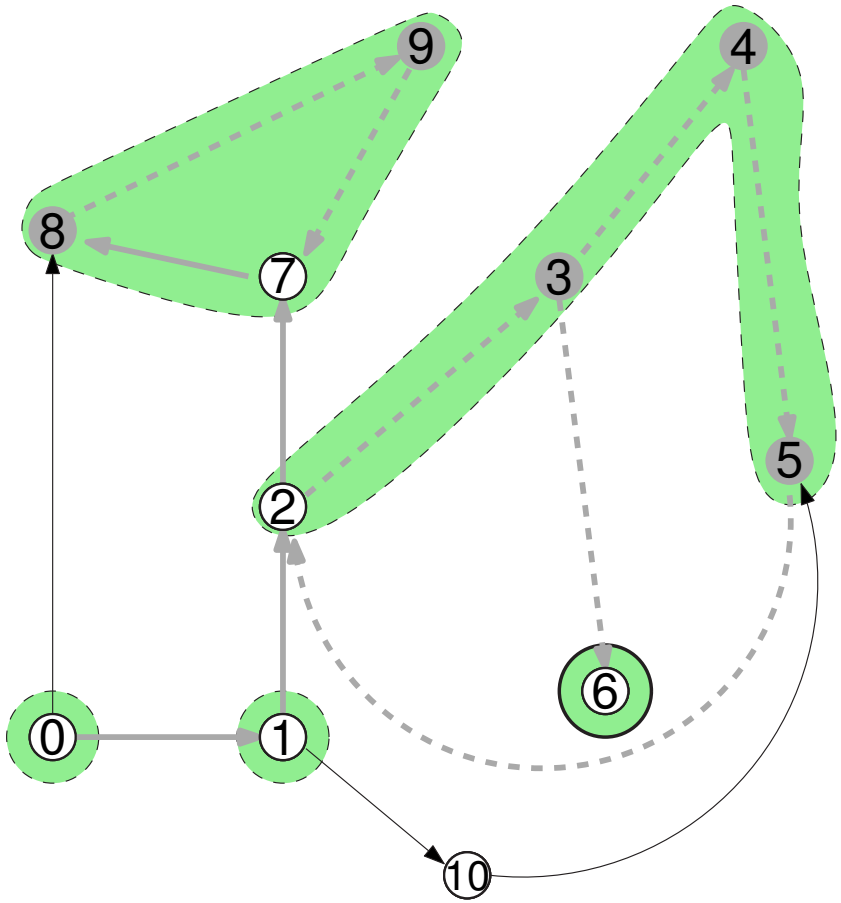

| oNodes | oReps |
|---|---|
| 0 | 0 |
| 1 | 1 |
| 2 | 2 |
| 3 | |
| 4 | |
| 5 | |
| 7 | 7 |
| 8 | |
| 9 | |

# SCC (Wiederholung)



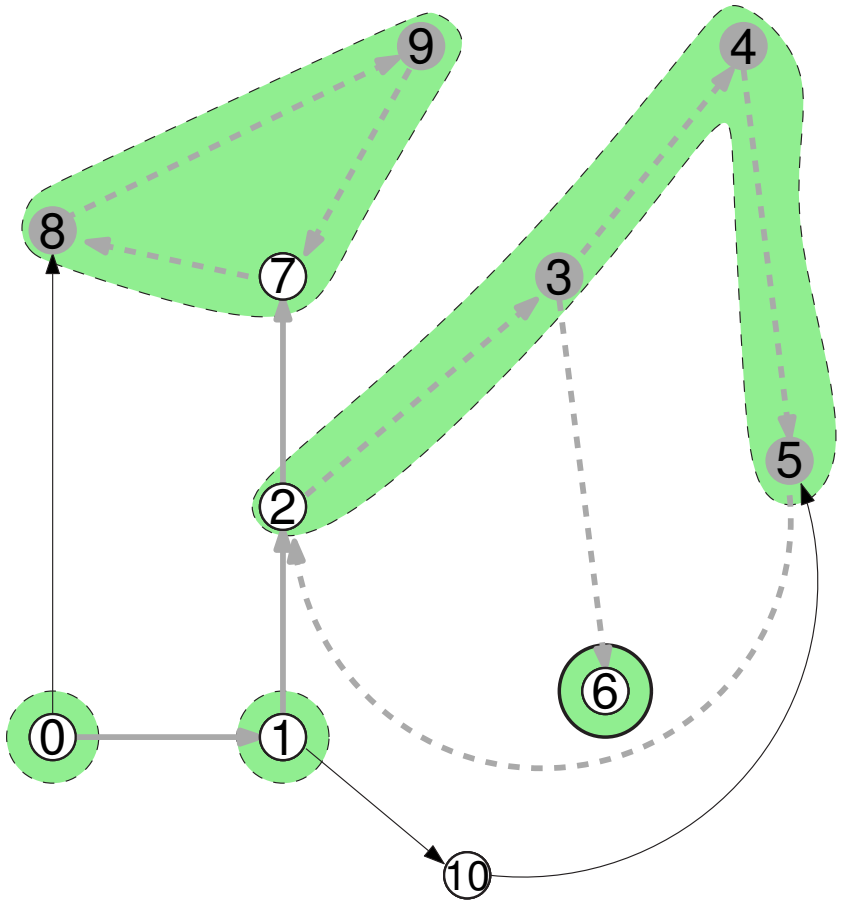

| oNodes | oReps |
|---|---|
| 0 | 0 |
| 1 | 1 |
| 2 | 2 |
| 3 | |
| 4 | |
| 5 | |
| 7 | 7 |
| 8 | |
| 9 | |

# SCC (Wiederholung)



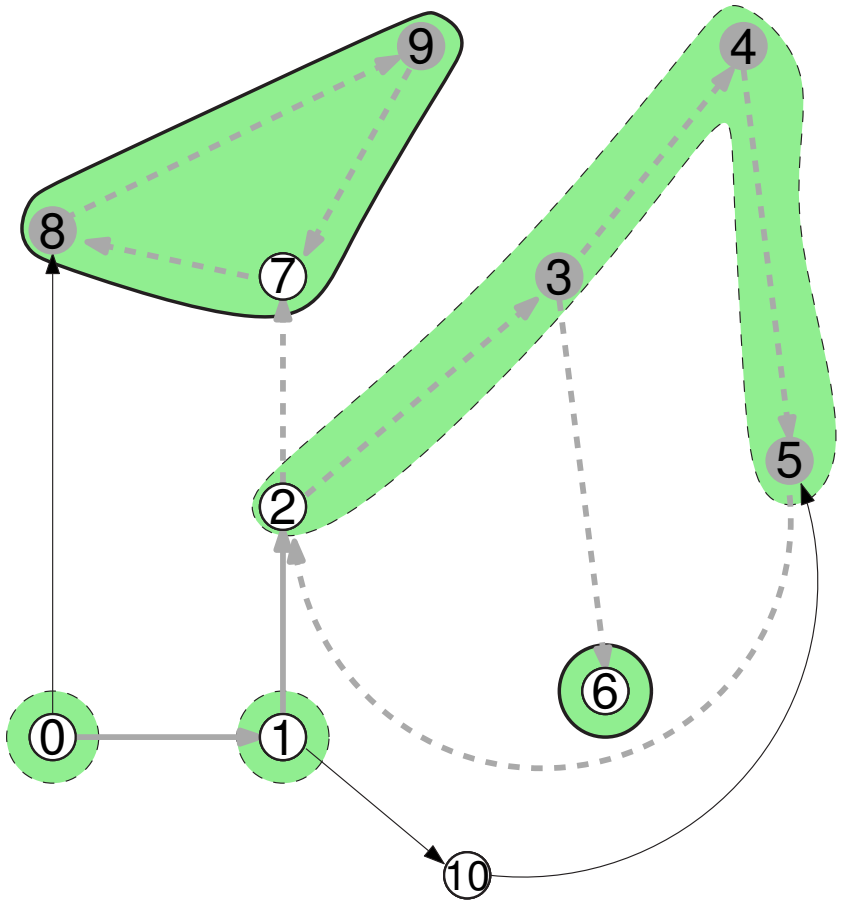

| oNodes | oReps |
|---|---|
| 0 | 0 |
| 1 | 1 |
| 2 | 2 |
| 3 | |
| 4 | |
| 5 | |
| 7 | 7 |
| 8 | |
| 9 | |

# SCC (Wiederholung)



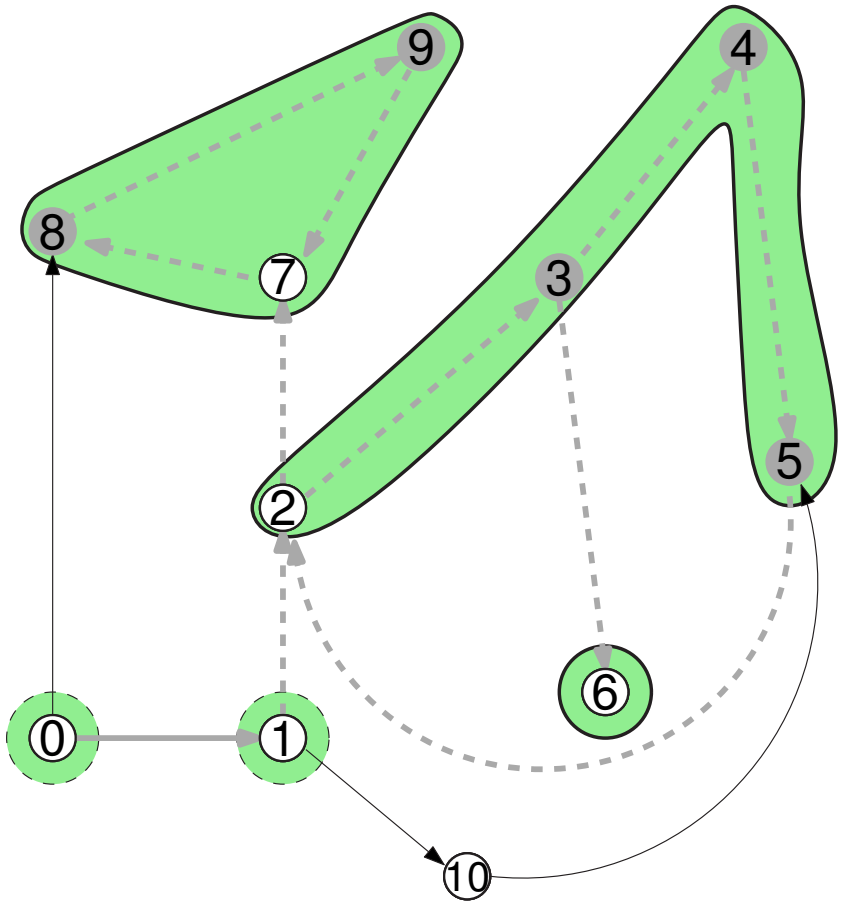

| oNodes | oReps |
|---|---|
| 0 | 0 |
| 1 | 1 |
| 2 | 2 |
| 3 | |
| 4 | |
| 5 | |
| 7 | 7 |
| 8 | |
| 9 | |

# SCC (Wiederholung)




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

# SCC (Wiederholung)




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

# Floyd Warshall: SCC als Speedup Technik

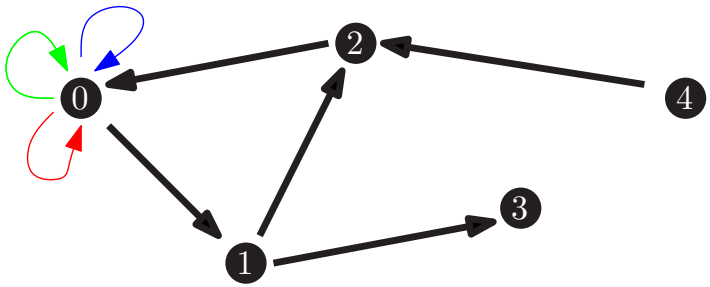
# Floyd Warshall

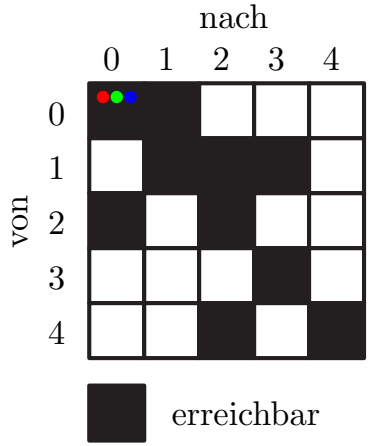
- kubischer Algorithmus
- berechnet transitive Hülle

  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
```

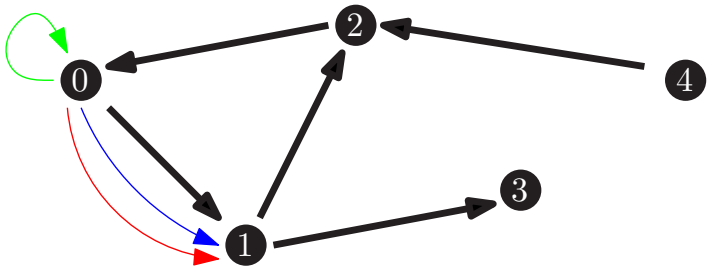
# Floyd Warshall

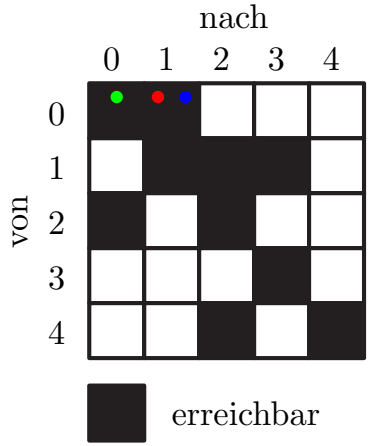



```
for int k = 0; k < n; ++k do
    for int i = 0; i < n; ++i do
        for int j = 0; j < n; ++j do
            array[i][j] = array[i][j] ||
            (array[i][k] && array[k][j]);
        end
    end
end
```

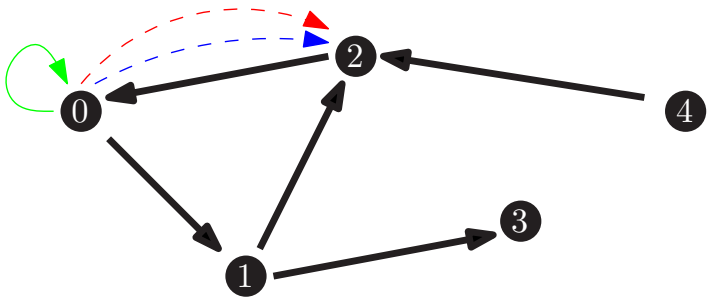
# Floyd Warshall

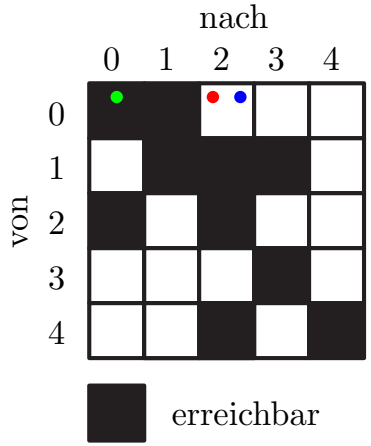



```
for int k = 0; k < n; ++k do
    for int i = 0; i < n; ++i do
        for int j = 0; j < n; ++j do
            array[i][j] = array[i][j] ||
            (array[i][k] && array[k][j]);
        end
    end
end
```

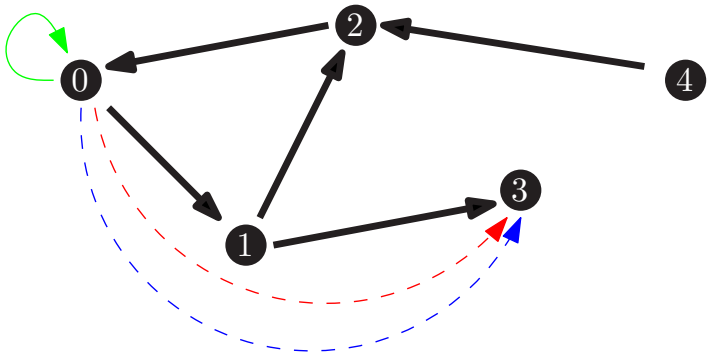
# Floyd Warshall

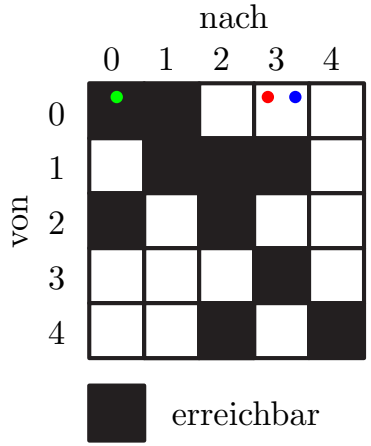



```
for int k = 0; k < n; ++k do
    for int i = 0; i < n; ++i do
        for int j = 0; j < n; ++j do
            array[i][j] = array[i][j] ||
            (array[i][k] && array[k][j]);
        end
    end
end
```

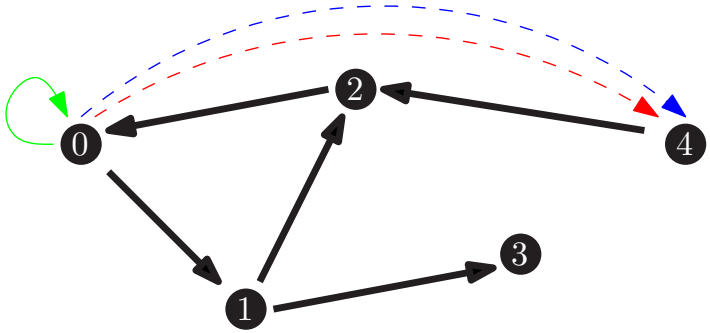
# Floyd Warshall

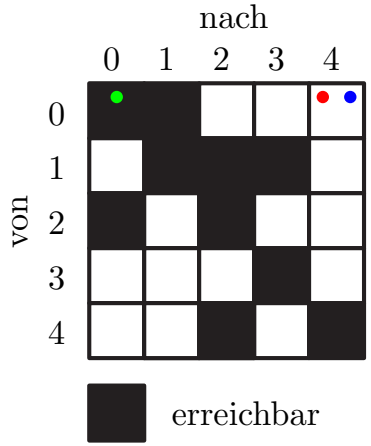



```
for int k = 0; k < n; ++k do
    for int i = 0; i < n; ++i do
        for int j = 0; j < n; ++j do
            array[i][j] = array[i][j] ||
            (array[i][k] && array[k][j]);
        end
    end
end
```

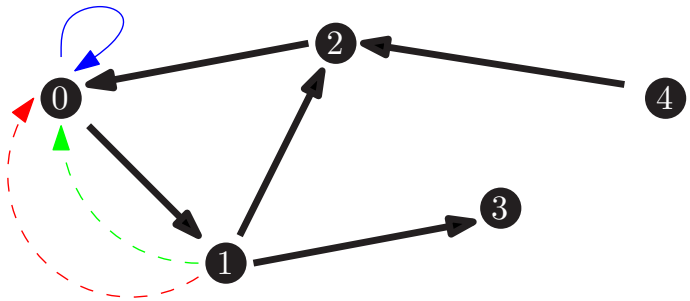
# Floyd Warshall

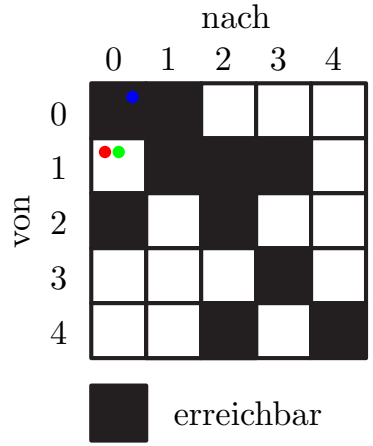



```
for int k = 0; k < n; ++k do
    for int i = 0; i < n; ++i do
        for int j = 0; j < n; ++j do
            array[i][j] = array[i][j] ||
            (array[i][k] && array[k][j]);
        end
    end
end
```

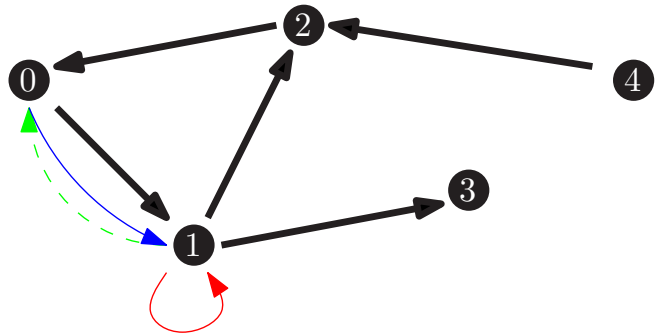
# Floyd Warshall

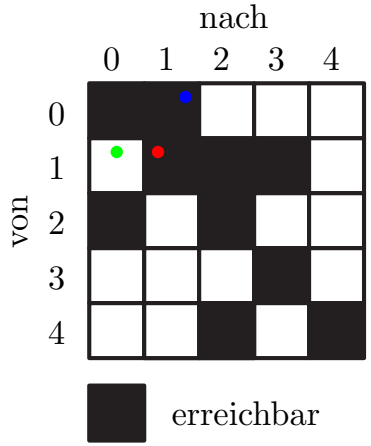



```
for int k = 0; k < n; ++k do
    for int i = 0; i < n; ++i do
        for int j = 0; j < n; ++j do
            array[i][j] = array[i][j] ||
            (array[i][k] && array[k][j]);
        end
    end
end
```

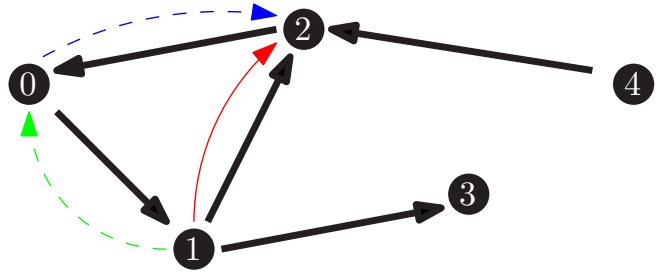
# Floyd Warshall

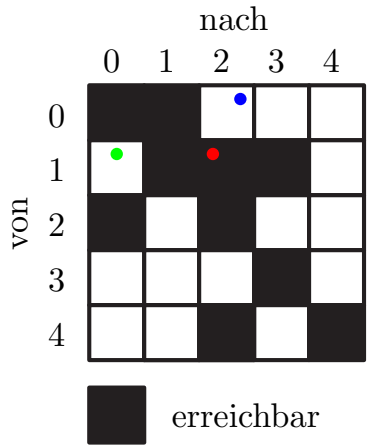



```
for int k = 0; k < n; ++k do
    for int i = 0; i < n; ++i do
        for int j = 0; j < n; ++j do
            array[i][j] = array[i][j] ||
            (array[i][k] && array[k][j]);
        end
    end
end
```

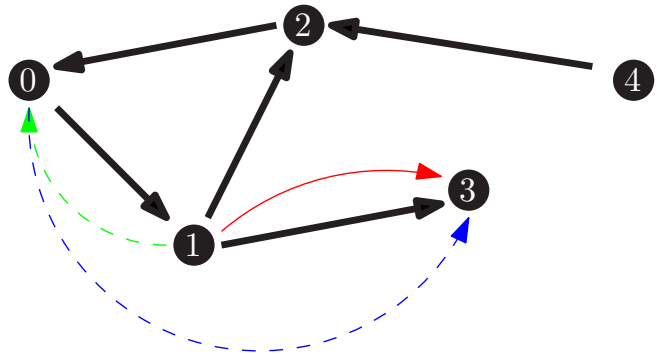
# Floyd Warshall

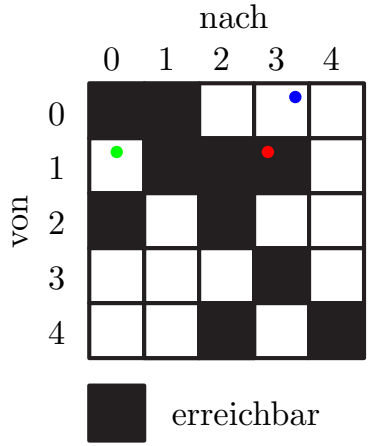



```
for int k = 0; k < n; ++k do
    for int i = 0; i < n; ++i do
        for int j = 0; j < n; ++j do
            array[i][j] = array[i][j] ||
            (array[i][k] && array[k][j]);
        end
    end
end
```

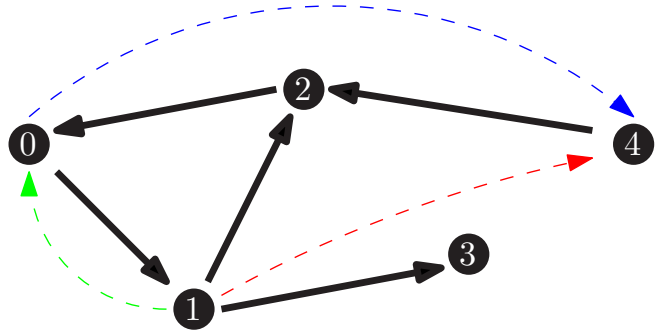
# Floyd Warshall

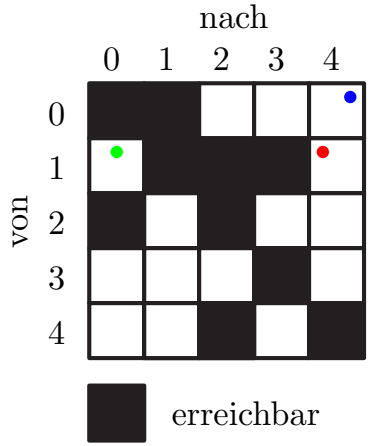



```
for int k = 0; k < n; ++k do
    for int i = 0; i < n; ++i do
        for int j = 0; j < n; ++j do
            array[i][j] = array[i][j] ||
            (array[i][k] && array[k][j]);
        end
    end
end
```

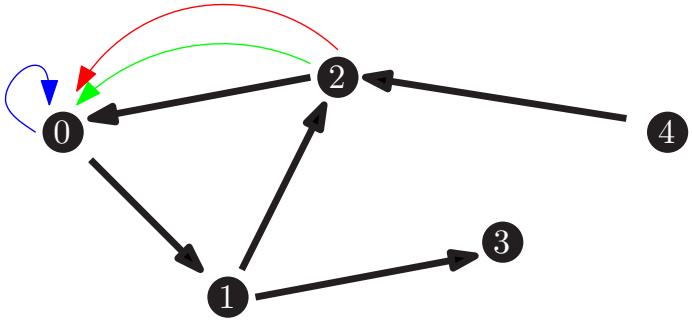
# Floyd Warshall

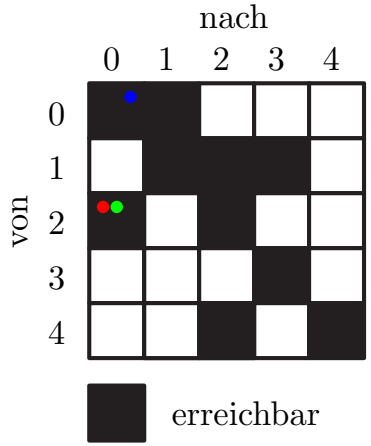



```
for int k = 0; k < n; ++k do
    for int i = 0; i < n; ++i do
        for int j = 0; j < n; ++j do
            array[i][j] = array[i][j] ||
            (array[i][k] && array[k][j]);
        end
    end
end
```

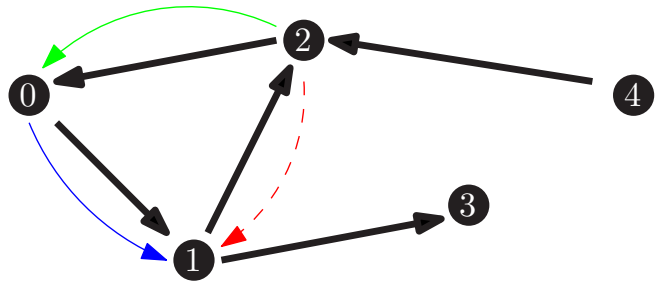
# Floyd Warshall

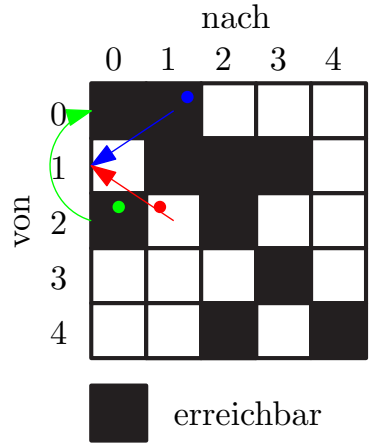



```
for int k = 0; k < n; ++k do
    for int i = 0; i < n; ++i do
        for int j = 0; j < n; ++j do
            array[i][j] = array[i][j] ||
            (array[i][k] && array[k][j]);
        end
    end
end
```

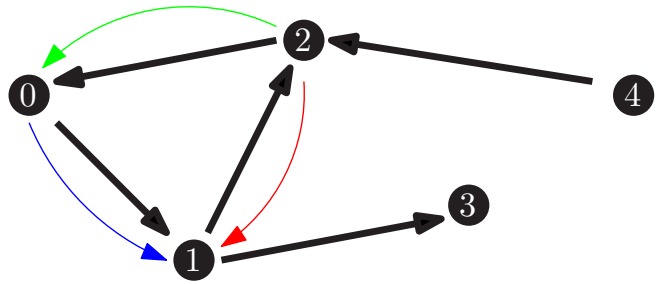
# Floyd Warshall

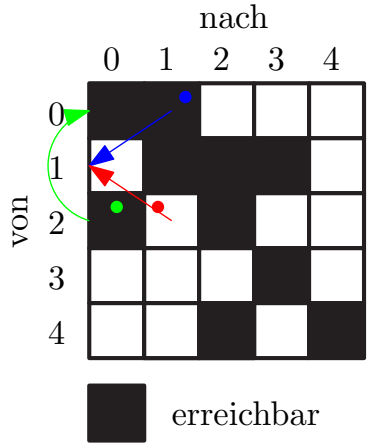



```
for int k = 0; k < n; ++k do
    for int i = 0; i < n; ++i do
        for int j = 0; j < n; ++j do
            array[i][j] = array[i][j] ||
            (array[i][k] && array[k][j]);
        end
    end
end
```

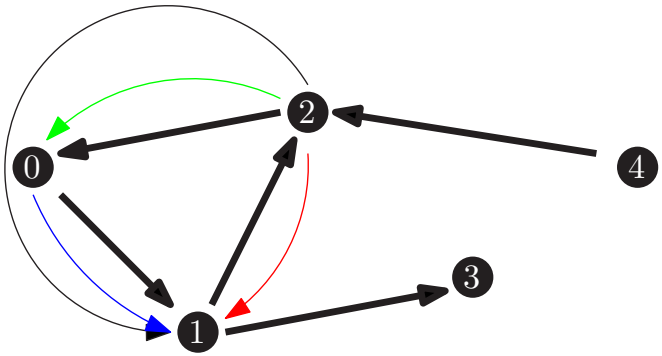
# Floyd Warshall

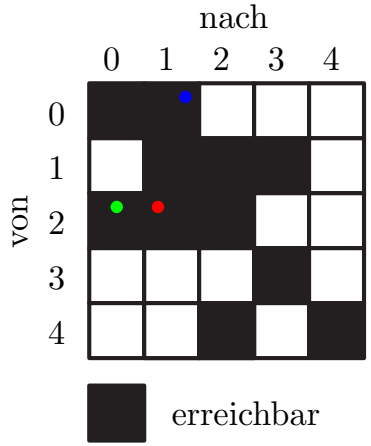



```
for int k = 0; k < n; ++k do
    for int i = 0; i < n; ++i do
        for int j = 0; j < n; ++j do
            array[i][j] = array[i][j] ||
            (array[i][k] && array[k][j]);
        end
    end
end
```

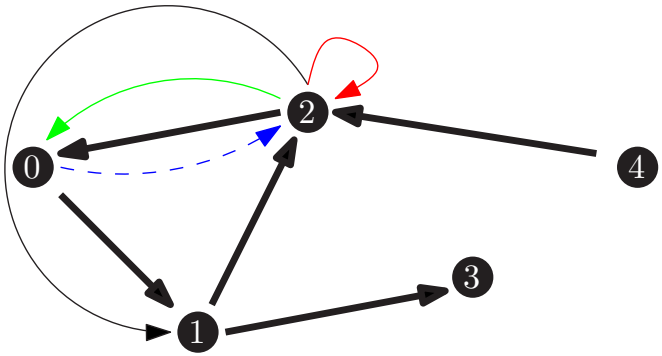
# Floyd Warshall

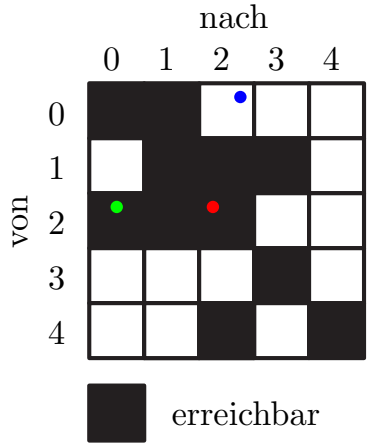



```
for int k = 0; k < n; ++k do
    for int i = 0; i < n; ++i do
        for int j = 0; j < n; ++j do
            array[i][j] = array[i][j] ||
            (array[i][k] && array[k][j]);
        end
    end
end
```

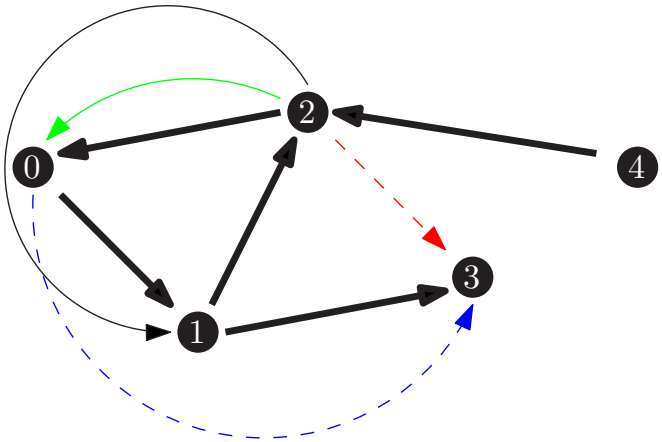
# Floyd Warshall

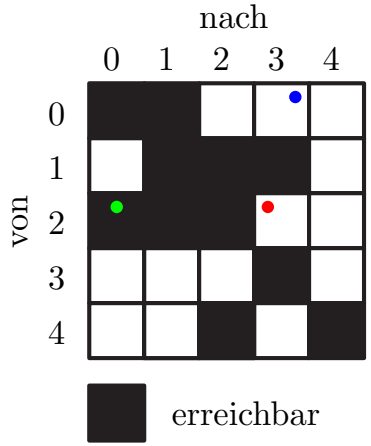



```
for int k = 0; k < n; ++k do
    for int i = 0; i < n; ++i do
        for int j = 0; j < n; ++j do
            array[i][j] = array[i][j] ||
            (array[i][k] && array[k][j]);
        end
    end
end
```

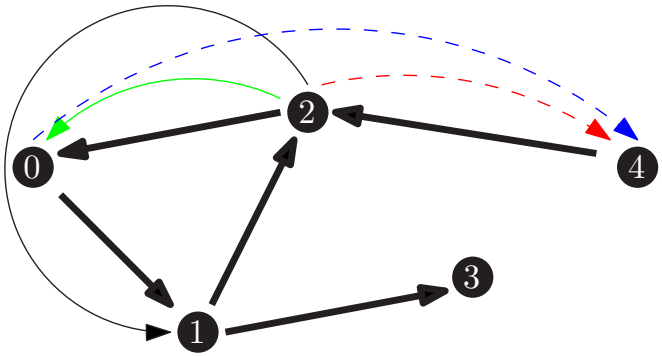
# Floyd Warshall

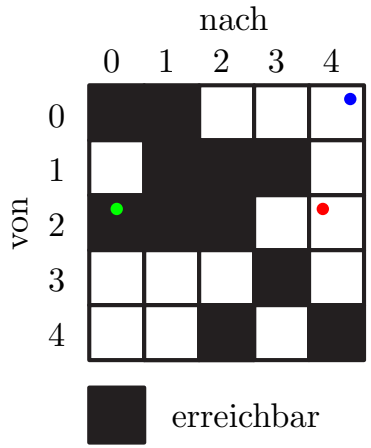



```
for int k = 0; k < n; ++k do
    for int i = 0; i < n; ++i do
        for int j = 0; j < n; ++j do
            array[i][j] = array[i][j] ||
            (array[i][k] && array[k][j]);
        end
    end
end
```

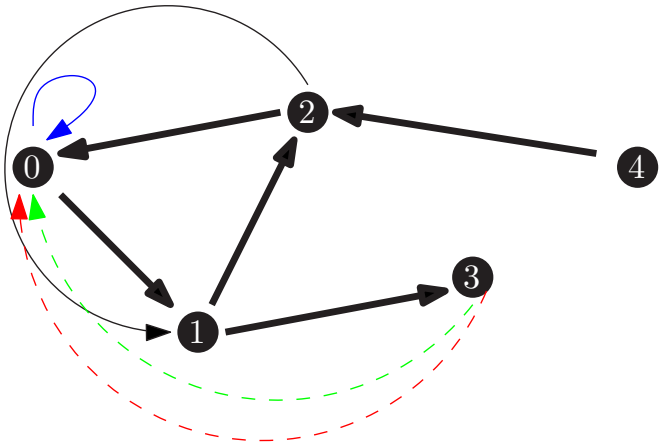
# Floyd Warshall

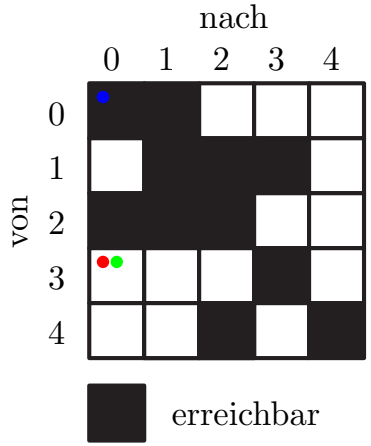



```
for int k = 0; k < n; ++k do
    for int i = 0; i < n; ++i do
        for int j = 0; j < n; ++j do
            array[i][j] = array[i][j] ||
            (array[i][k] && array[k][j]);
        end
    end
end
```

# Floyd Warshall



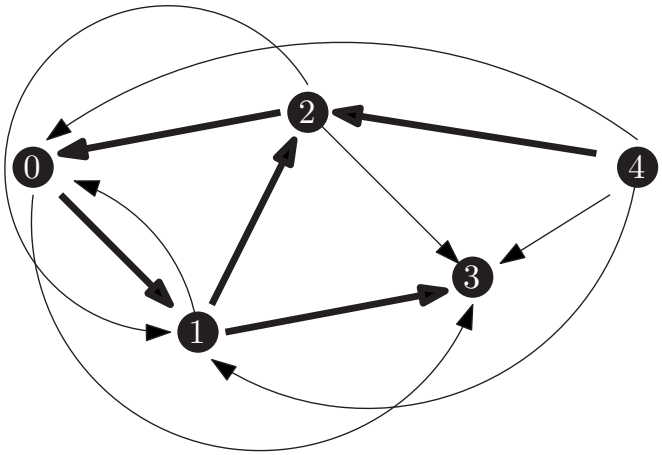



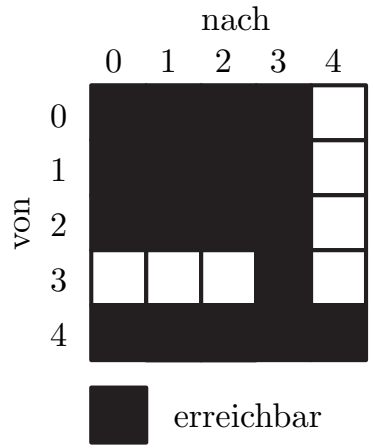

```
for int k = 0; k < n; ++k do
    for int i = 0; i < n; ++i do
        for int j = 0; j < n; ++j do
            array[i][j] = array[i][j] ||
            (array[i][k] && array[k][j]);
        end
    end
end
```

# Floyd Warshall

107 Iterationen später . . .

# Floyd Warshall



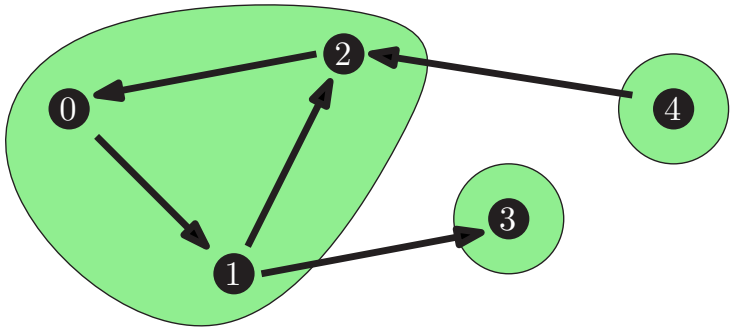



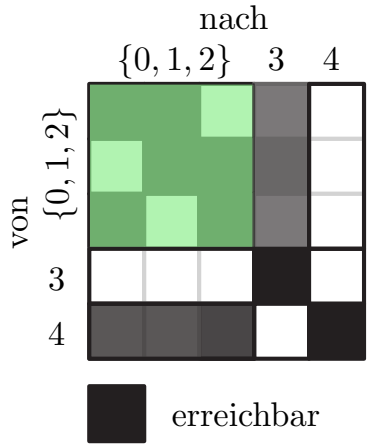

```
for int k = 0; k < n; ++k do
    for int i = 0; i < n; ++i do
        for int j = 0; j < n; ++j do
            array[i][j] = array[i][j] ||
            (array[i][k] && array[k][j]);
        end
    end
end
```

# Floyd Warshall und SCC

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

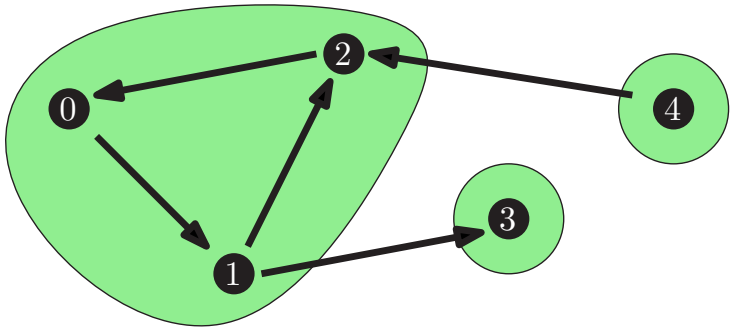
# Floyd Warshall und SCC

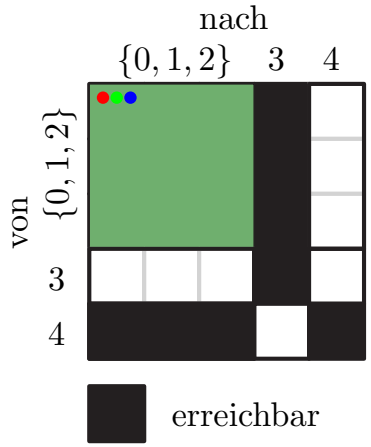



```
for int k = 0; k < n; ++k do
    for int i = 0; i < n; ++i do
        for int j = 0; j < n; ++j do
            array[i][j] = array[i][j] ||
            (array[i][k] && array[k][j]);
        end
    end
end
```

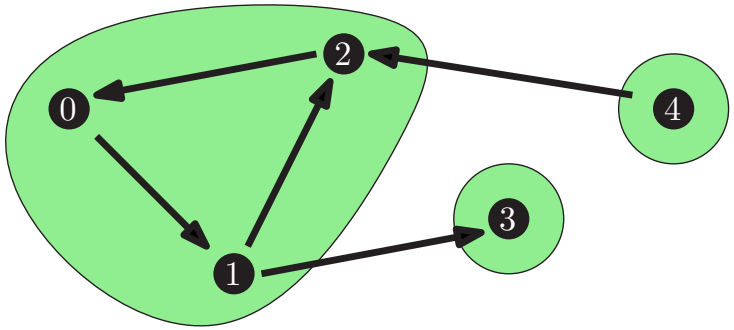
# Floyd Warshall und SCC

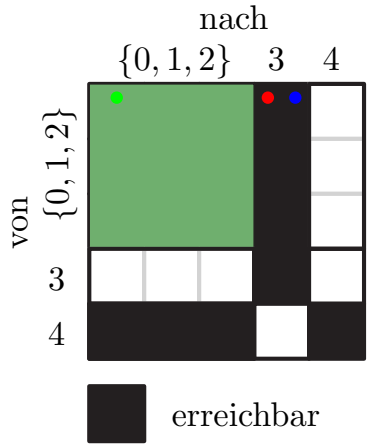



```
for int k = 0; k < n; ++k do
    for int i = 0; i < n; ++i do
        for int j = 0; j < n; ++j do
            array[i][j] = array[i][j] ||
            (array[i][k] && array[k][j]);
        end
    end
end
```

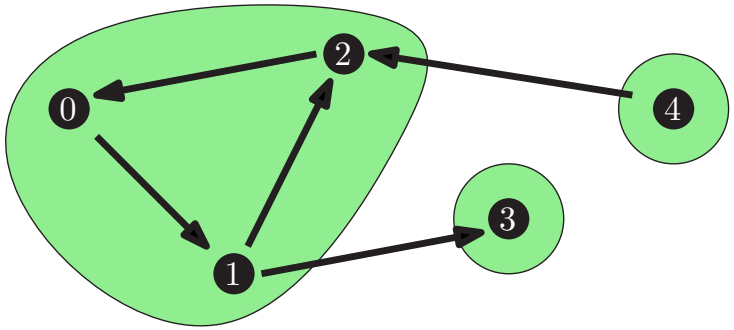
# Floyd Warshall und SCC

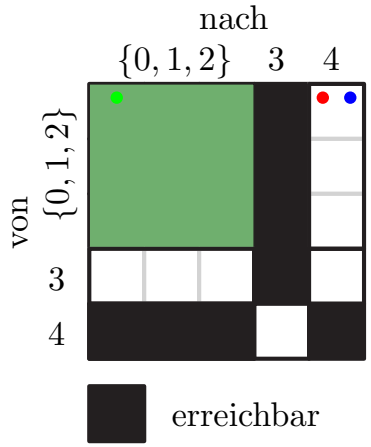



```
for int k = 0; k < n; ++k do
    for int i = 0; i < n; ++i do
        for int j = 0; j < n; ++j do
            array[i][j] = array[i][j] ||
            (array[i][k] && array[k][j]);
        end
    end
end
```

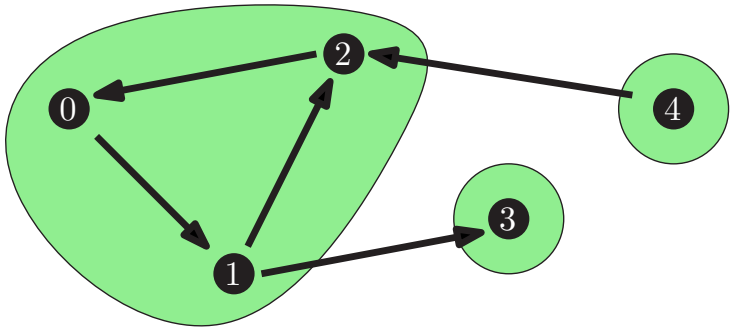
# Floyd Warshall und SCC

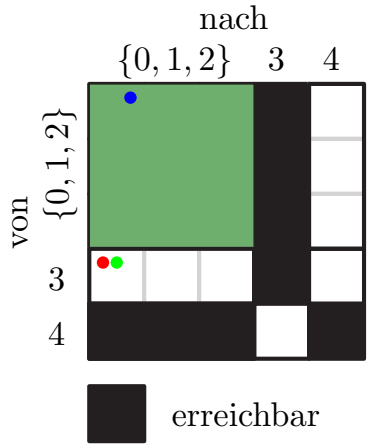



```
for int k = 0; k < n; ++k do
    for int i = 0; i < n; ++i do
        for int j = 0; j < n; ++j do
            array[i][j] = array[i][j] ||
            (array[i][k] && array[k][j]);
        end
    end
end
```

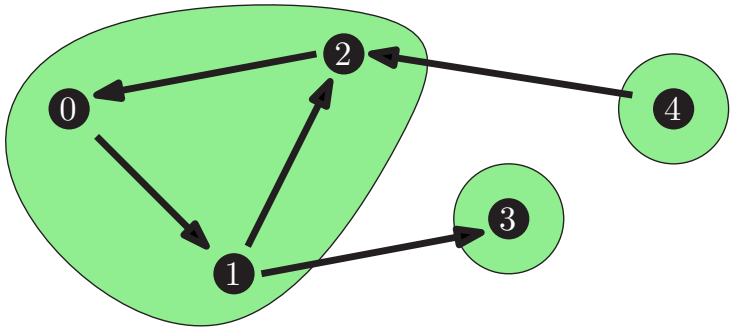
# Floyd Warshall und SCC

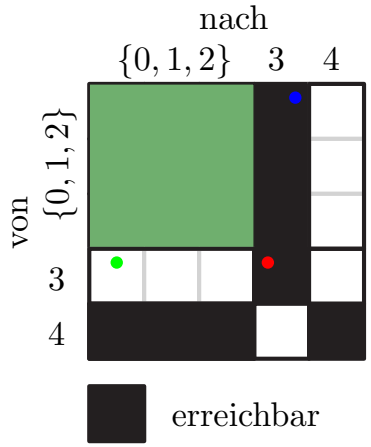



```
for int k = 0; k < n; ++k do
    for int i = 0; i < n; ++i do
        for int j = 0; j < n; ++j do
            array[i][j] = array[i][j] ||
            (array[i][k] && array[k][j]);
        end
    end
end
```

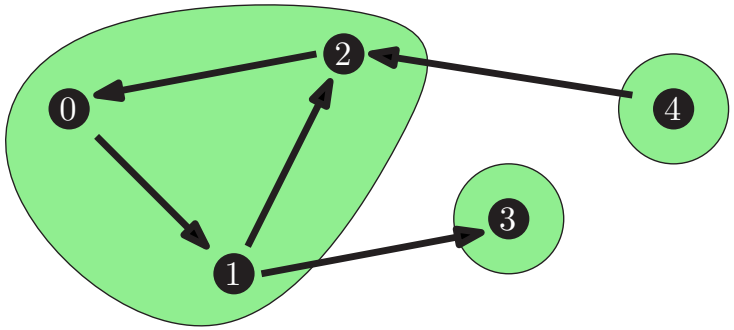
# Floyd Warshall und SCC

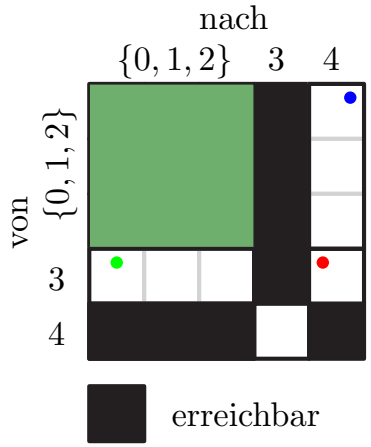



```
for int k = 0; k < n; ++k do
    for int i = 0; i < n; ++i do
        for int j = 0; j < n; ++j do
            array[i][j] = array[i][j] ||
            (array[i][k] && array[k][j]);
        end
    end
end
```

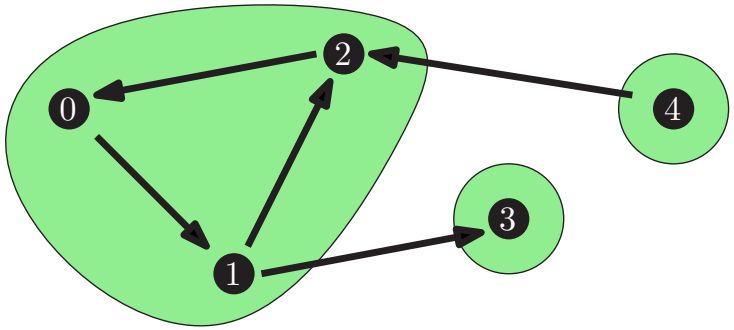
# Floyd Warshall und SCC

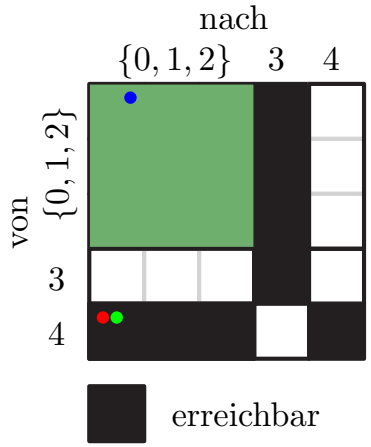



```
for int k = 0; k < n; ++k do
    for int i = 0; i < n; ++i do
        for int j = 0; j < n; ++j do
            array[i][j] = array[i][j] ||
            (array[i][k] && array[k][j]);
        end
    end
end
```

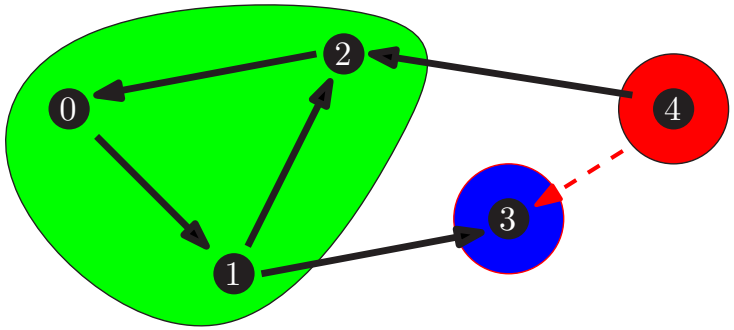
# Floyd Warshall und SCC

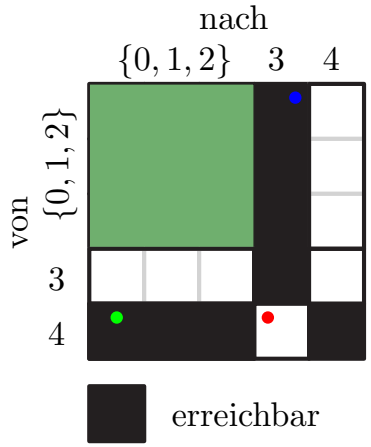



```
for int k = 0; k < n; ++k do
    for int i = 0; i < n; ++i do
        for int j = 0; j < n; ++j do
            array[i][j] = array[i][j] ||
            (array[i][k] && array[k][j]);
        end
    end
end
```

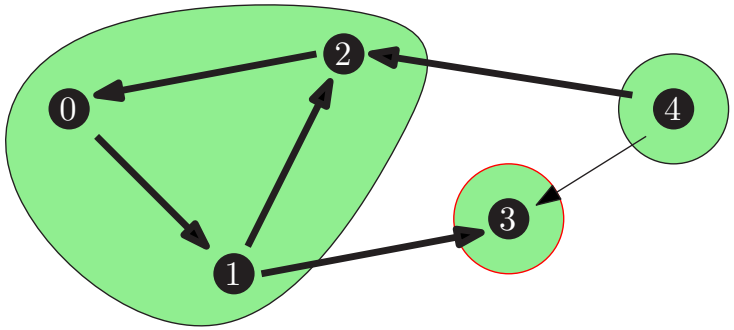
# Floyd Warshall und SCC

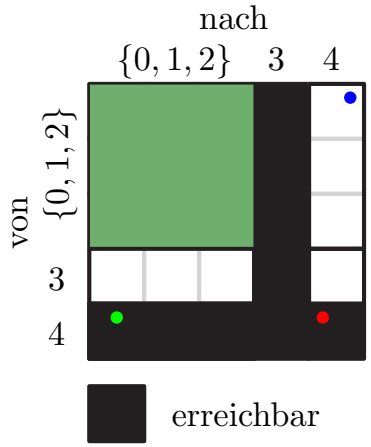



```
for int k = 0; k < n; ++k do
    for int i = 0; i < n; ++i do
        for int j = 0; j < n; ++j do
            array[i][j] = array[i][j] ||
            (array[i][k] && array[k][j]);
        end
    end
end
```

# 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!