

# Übung 3 – Algorithmen II

Moritz Laupichler, Hans-Peter Lehmann – {moritz.laupichler, hans-peter.lehmann}@kit.edu  
[http://algo2.iti.kit.edu/AlgorithmenII\\_WS22.php](http://algo2.iti.kit.edu/AlgorithmenII_WS22.php)

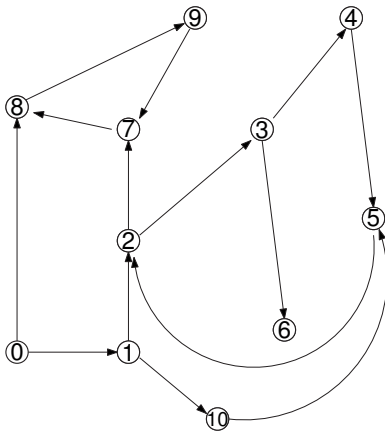
Institut für Theoretische Informatik - Algorithmik II

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

- SCCs mit DFS berechnen
- Besprechung Übungsblatt 1

# Starke Zusammenhangskomponenten



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

## Invariante 1

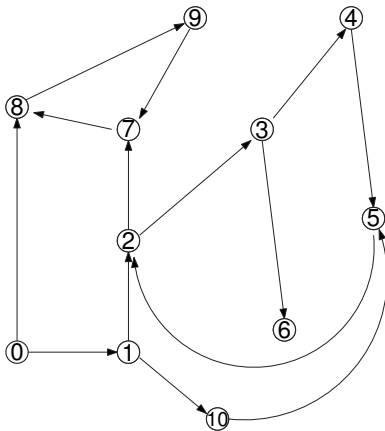
Keine Kanten von geschlossenen in offene Komponenten.

## Invariante 2

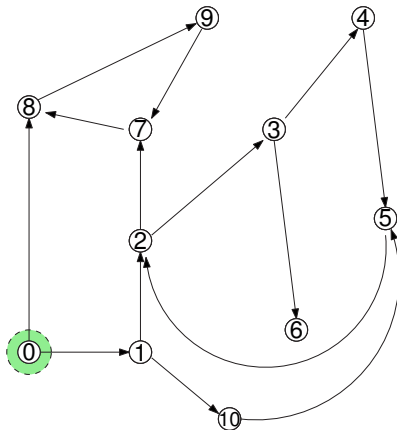
Offene Komponenten liegen auf Pfad.

## Invariante 3

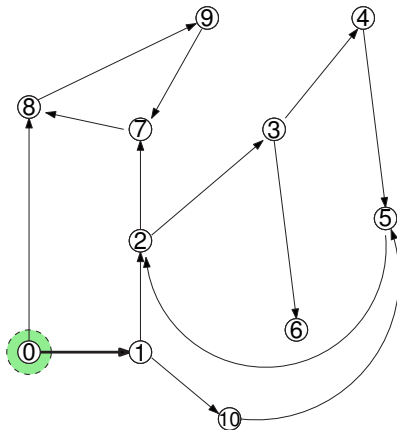
Repräsentanten partitionieren offene Komponenten bzgl. dfsNum.



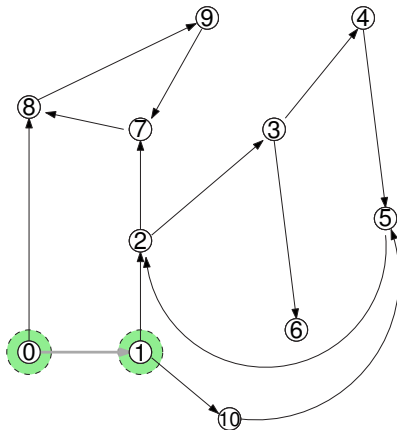
| oNodes | oReps |
|--------|-------|
|        |       |



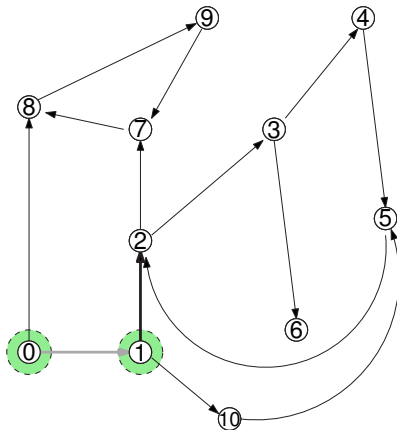
| oNodes | oReps |
|--------|-------|
| 0      | 0     |



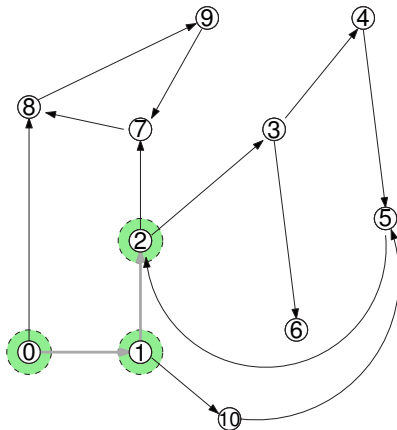
| oNodes | oReps |
|--------|-------|
| 0      | 0     |



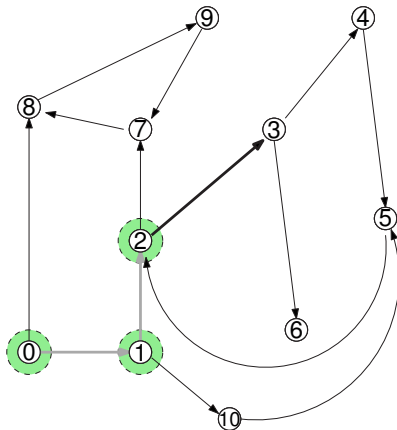
| oNodes | oReps |
|--------|-------|
| 0      | 0     |
| 1      | 1     |



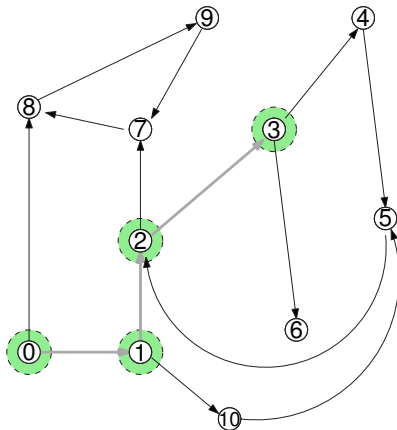
| oNodes | oReps |
|--------|-------|
| 0      | 0     |
| 1      | 1     |



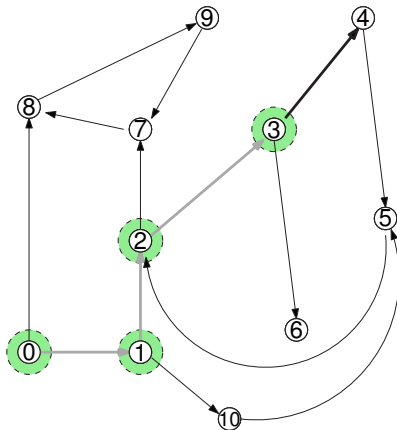
| oNodes | oReps |
|--------|-------|
| 0      | 0     |
| 1      | 1     |
| 2      | 2     |



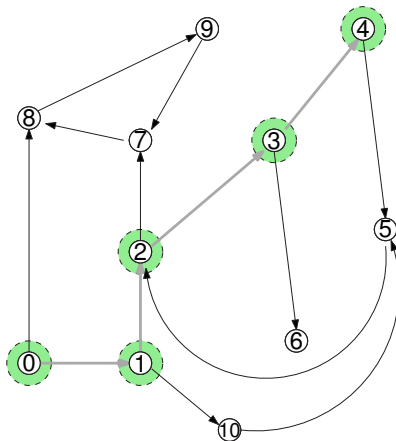
| oNodes | oReps |
|--------|-------|
| 0      | 0     |
| 1      | 1     |
| 2      | 2     |



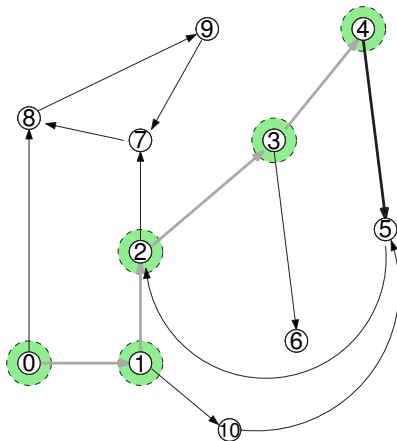
| oNodes | oReps |
|--------|-------|
| 0      | 0     |
| 1      | 1     |
| 2      | 2     |
| 3      | 3     |



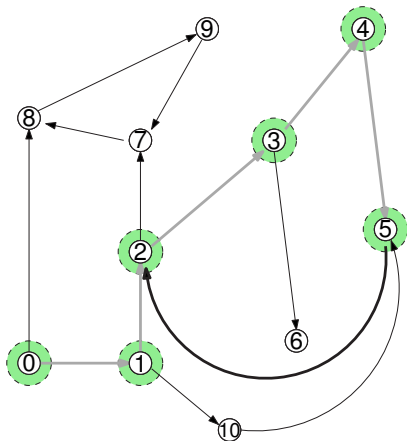
| oNodes | oReps |
|--------|-------|
| 0      | 0     |
| 1      | 1     |
| 2      | 2     |
| 3      | 3     |



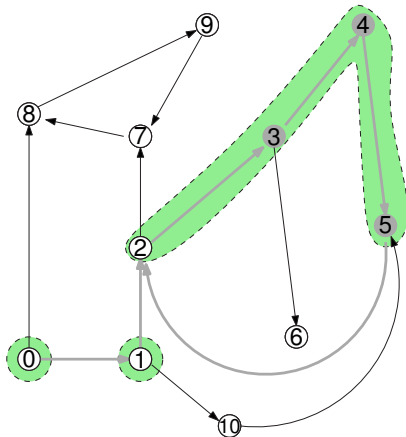
| oNodes | oReps |
|--------|-------|
| 0      | 0     |
| 1      | 1     |
| 2      | 2     |
| 3      | 3     |
| 4      | 4     |



| 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

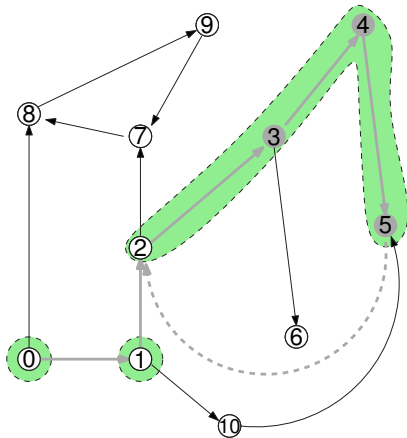
## Invariante 2

Offene Komponenten liegen auf Pfad.

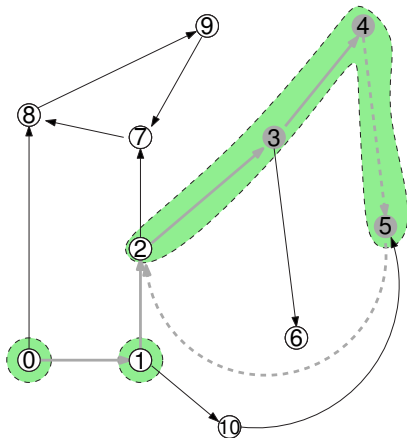
- Repräsentant ist minimal auf Kreis

## Invariante 3

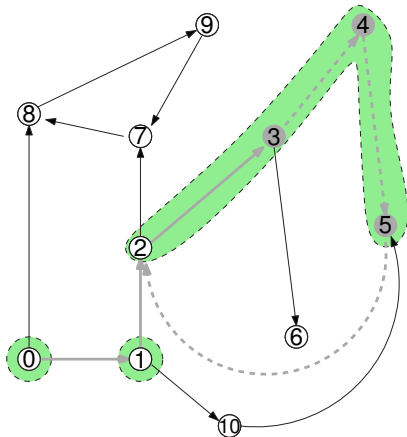
Repräsentanten partitionieren offene Komponenten bzgl. dfsNum.



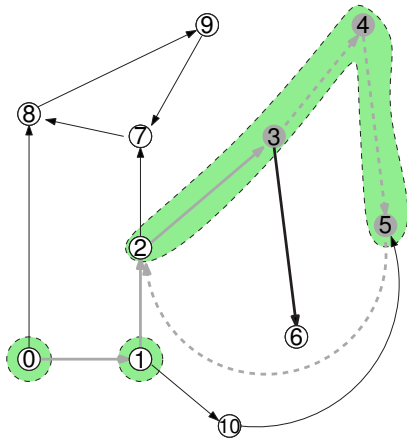
| oNodes | oReps |
|--------|-------|
| 0      | 0     |
| 1      | 1     |
| 2      | 2     |
| 3      |       |
| 4      |       |
| 5      |       |



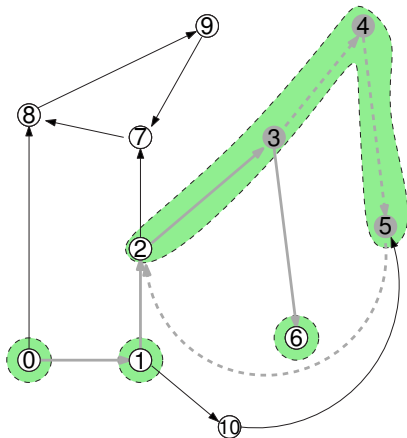
| oNodes | oReps |
|--------|-------|
| 0      | 0     |
| 1      | 1     |
| 2      | 2     |
| 3      |       |
| 4      |       |
| 5      |       |



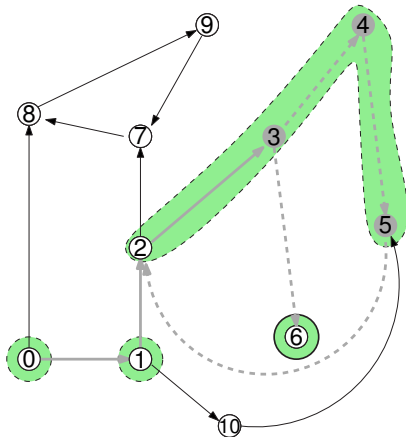
| oNodes | oReps |
|--------|-------|
| 0      | 0     |
| 1      | 1     |
| 2      | 2     |
| 3      |       |
| 4      |       |
| 5      |       |



| 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

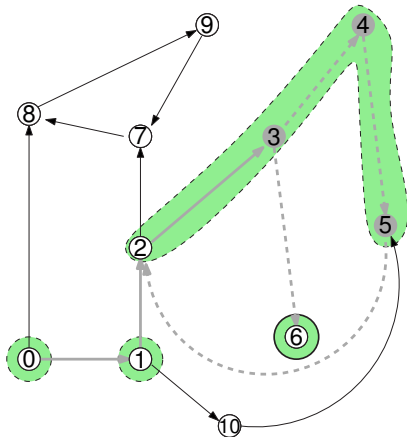
Keine Kanten von geschlossenen in offene Komponenten.

## Invariante 2

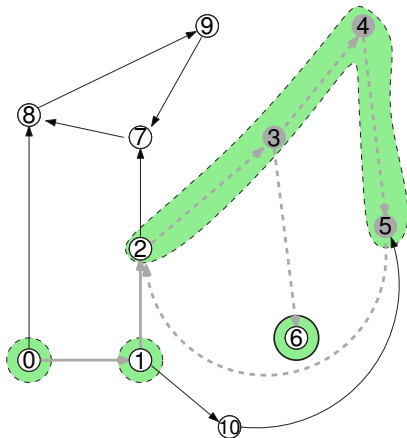
Offene Komponenten liegen auf Pfad.

## Invariante 3

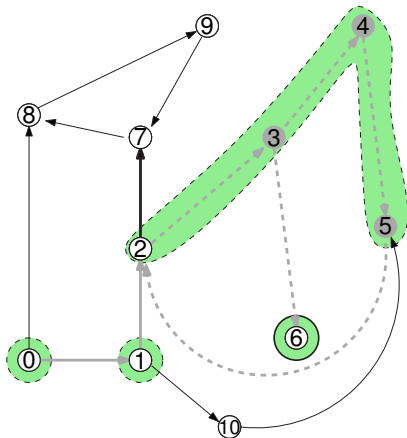
Repräsentanten partitionieren offene Komponenten bzgl. dfsNum.



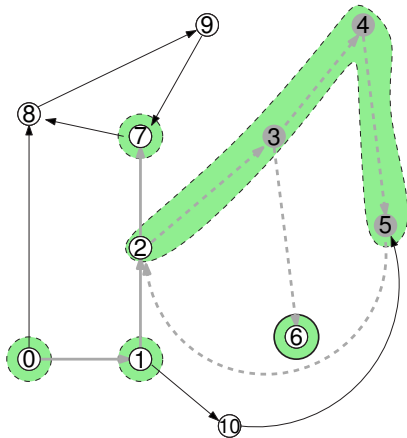
| oNodes | oReps |
|--------|-------|
| 0      | 0     |
| 1      | 1     |
| 2      | 2     |
| 3      |       |
| 4      |       |
| 5      |       |



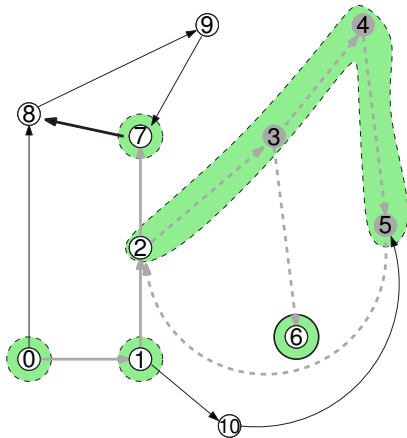
| oNodes | oReps |
|--------|-------|
| 0      | 0     |
| 1      | 1     |
| 2      | 2     |
| 3      |       |
| 4      |       |
| 5      |       |



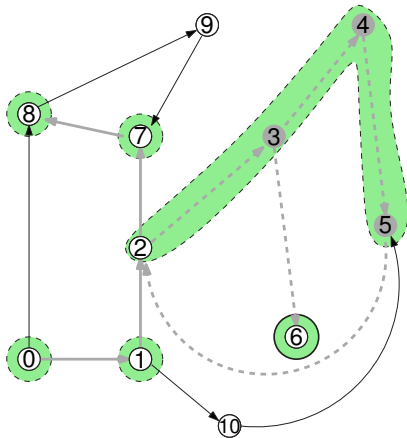
| oNodes | oReps |
|--------|-------|
| 0      | 0     |
| 1      | 1     |
| 2      | 2     |
| 3      |       |
| 4      |       |
| 5      |       |



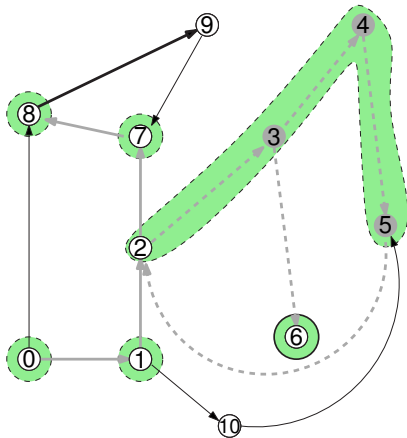
| oNodes | oReps |
|--------|-------|
| 0      | 0     |
| 1      | 1     |
| 2      | 2     |
| 3      |       |
| 4      |       |
| 5      |       |
| 7      | 7     |



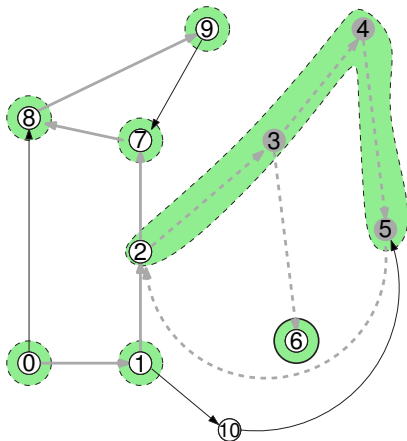
| oNodes | oReps |
|--------|-------|
| 0      | 0     |
| 1      | 1     |
| 2      | 2     |
| 3      |       |
| 4      |       |
| 5      |       |
| 7      | 7     |



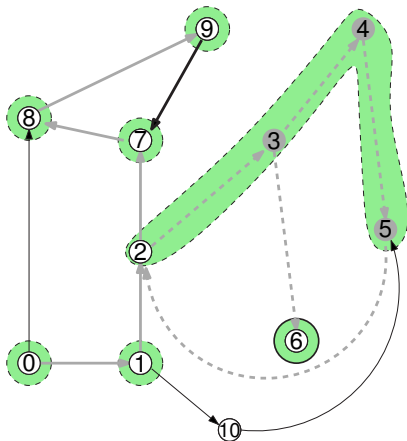
| oNodes | oReps |
|--------|-------|
| 0      | 0     |
| 1      | 1     |
| 2      | 2     |
| 3      |       |
| 4      |       |
| 5      |       |
| 7      | 7     |
| 8      | 8     |



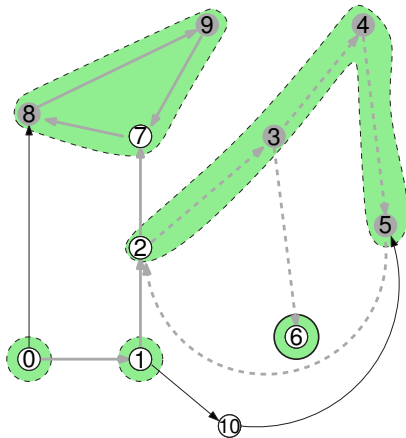
| oNodes | oReps |
|--------|-------|
| 0      | 0     |
| 1      | 1     |
| 2      | 2     |
| 3      |       |
| 4      |       |
| 5      |       |
| 7      | 7     |
| 8      | 8     |



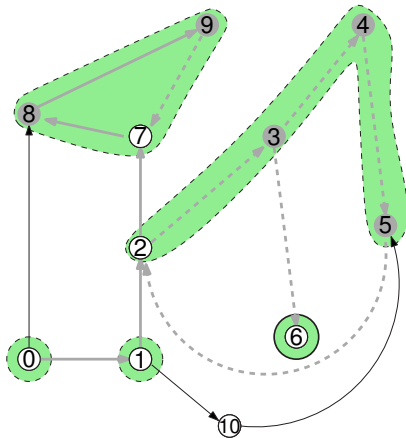
| oNodes | oReps |
|--------|-------|
| 0      | 0     |
| 1      | 1     |
| 2      | 2     |
| 3      |       |
| 4      |       |
| 5      |       |
| 7      | 7     |
| 8      | 8     |
| 9      | 9     |



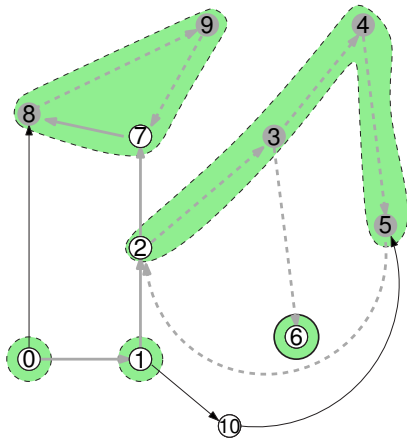
| oNodes | oReps |
|--------|-------|
| 0      | 0     |
| 1      | 1     |
| 2      | 2     |
| 3      |       |
| 4      |       |
| 5      |       |
| 7      | 7     |
| 8      | 8     |
| 9      | 9     |



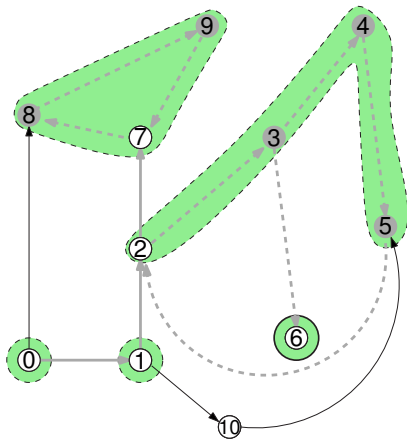
| oNodes | oReps |
|--------|-------|
| 0      | 0     |
| 1      | 1     |
| 2      | 2     |
| 3      |       |
| 4      |       |
| 5      |       |
| 7      | 7     |
| 8      |       |
| 9      |       |



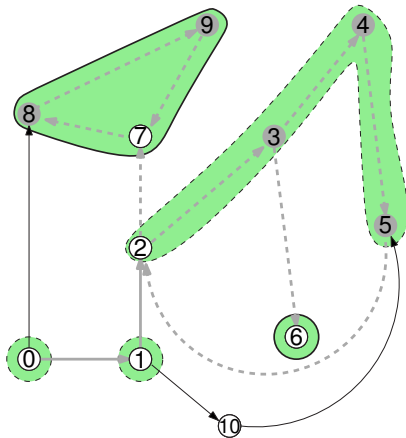
| oNodes | oReps |
|--------|-------|
| 0      | 0     |
| 1      | 1     |
| 2      | 2     |
| 3      |       |
| 4      |       |
| 5      |       |
| 7      | 7     |
| 8      |       |
| 9      |       |



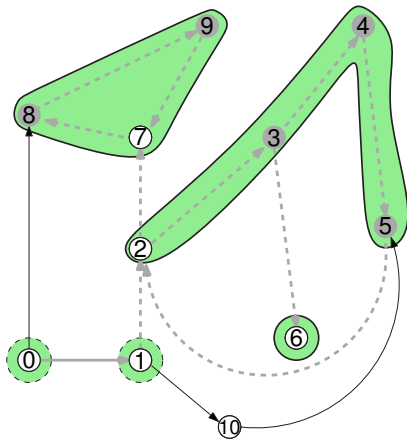
| oNodes | oReps |
|--------|-------|
| 0      | 0     |
| 1      | 1     |
| 2      | 2     |
| 3      |       |
| 4      |       |
| 5      |       |
| 7      | 7     |
| 8      |       |
| 9      |       |



| 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 5 (Entwurf: Datenstrukturen)

1. Erweitern Sie die Datenstruktur *Pairing Heap* um die Operation `increaseKey(h: Handle, k: Key)`. Ihre Operation sollte amortisiert  $O(\log n)$  Laufzeit benötigen. Geben Sie Pseudocode an. Wie würden Sie bei einem *Binary Heap* vorgehen?
2. Entwerfen Sie eine Datenstruktur welche die Operationen `insert` in  $O(\log n)$ , Median bestimmen in  $O(1)$  und Median entfernen in  $O(\log n)$  unterstützt. Eine Beschreibung in Worten ist ausreichend.

## Aufgabe 7 (Analyse: Laufzeit von Dijkstra's Algorithmus)

Gegeben sei ein gerichteter Graph  $G = (V, E)$  mit  $|V| = n$  und  $|E| = m$ , sowie eine Kantengewichtungsfunktion  $c : E \rightarrow \mathbb{R}_0^+$ .

1. Beweisen Sie die Behauptung aus der Vorlesung, dass für  $m = \Omega(n \log n \log \log n)$  Dijkstra's Algorithmus mit einem *binary heap* eine durchschnittliche Laufzeit von  $O(m)$  besitzt.
2. Eine spezielle *Priority Queue* habe folgende Laufzeiteigenschaften:
  - insert:  $O(\log n)$
  - decreaseKey:  $O(1)$
  - deleteMin:  $O(\sqrt{m})$

(ob eine Datenstruktur mit diesen Eigenschaften existiert und Dijkstra's Algorithmus mit ihr korrekt arbeitet, ist eine andere Frage, aber wir nehmen für diese Aufgabe an es ginge :-))

Geben Sie eine kleinste obere Schranke für die Laufzeit von Dijkstra's Algorithmus unter Verwendung dieser *Priority Queue* an. Unter welcher Bedingung an das Verhältnis der Anzahl Knoten  $n$  zu Kanten  $m$  wird die Laufzeit linear in der Eingabegröße? Die Eingabe erfolgt in Form einer Adjazenzliste.

# Ende!



# Feierabend!