ACM Vortrag
Dijkstra & Hamiltonkreis

von Rolf Schirm

am 22.06.2011

Dijkstra-Algorithmus

Beschreibung:

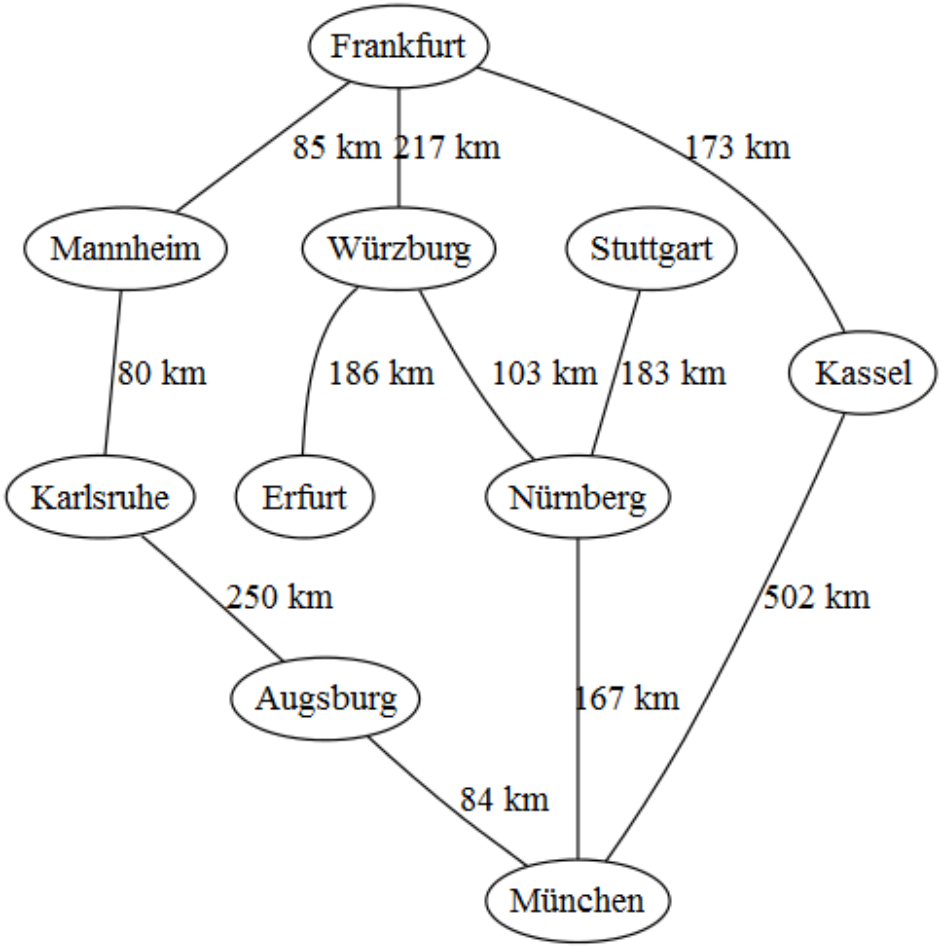
- Der Dijkstra-Algorithmus dient zur Berechnung des kürzesten Weges zwischen einem Startknoten und mehreren Endknoten in einem Graphen

- Vorraussetzung: Kantengewichtung darf nicht negativ sein!
- Die Kanten des Graphens können gerichtet oder ungerichtet sein
- Der Graph muss nicht zusammenhängend sein

Laufzeit:

  - $O(n^2 + m)$ mit normaler Liste
  - $O(n * \log(n) + m)$ mit Fibonacci-Heap (alle Operationen: $O(\log n)$)

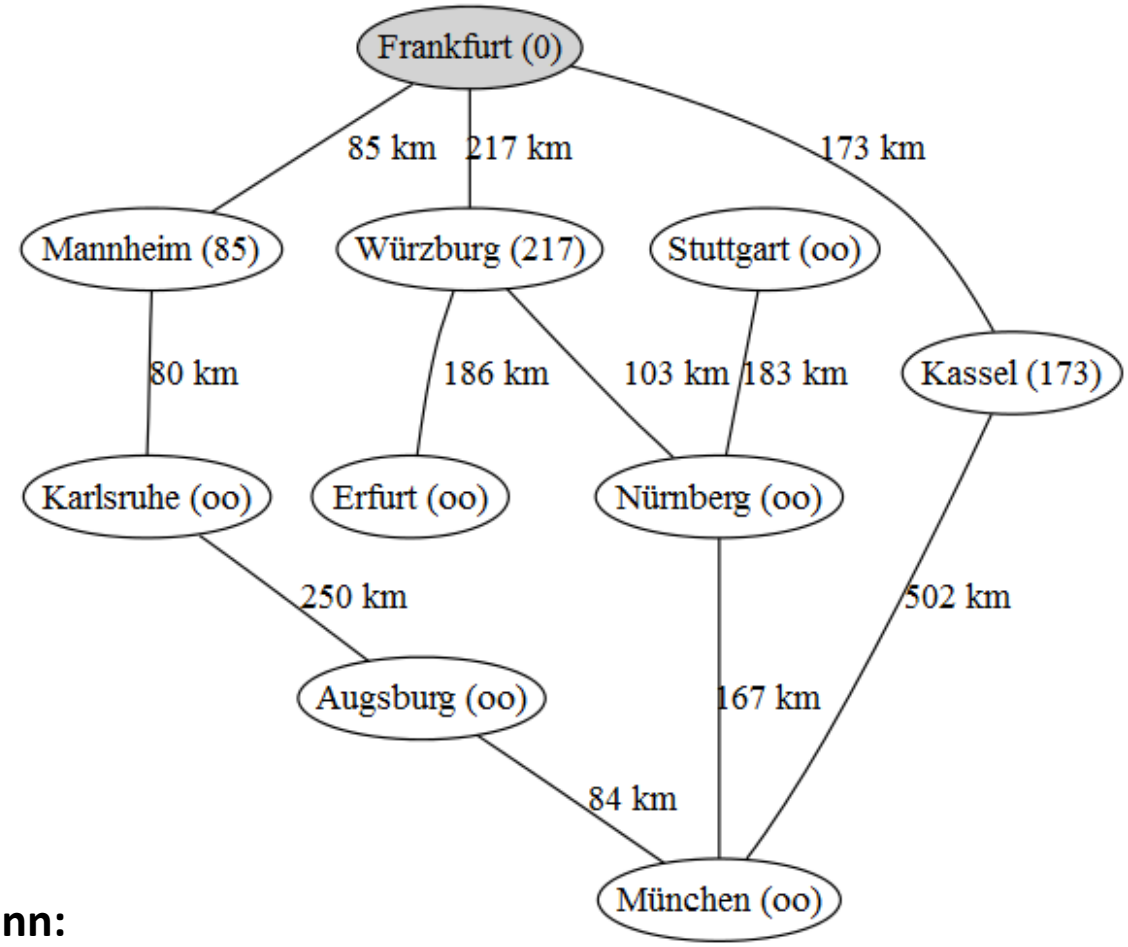
Beispiel: Frankfurt - München



Ersetze Wert wenn:

dist(start, current) + dist(current, next) < dist(start, next)

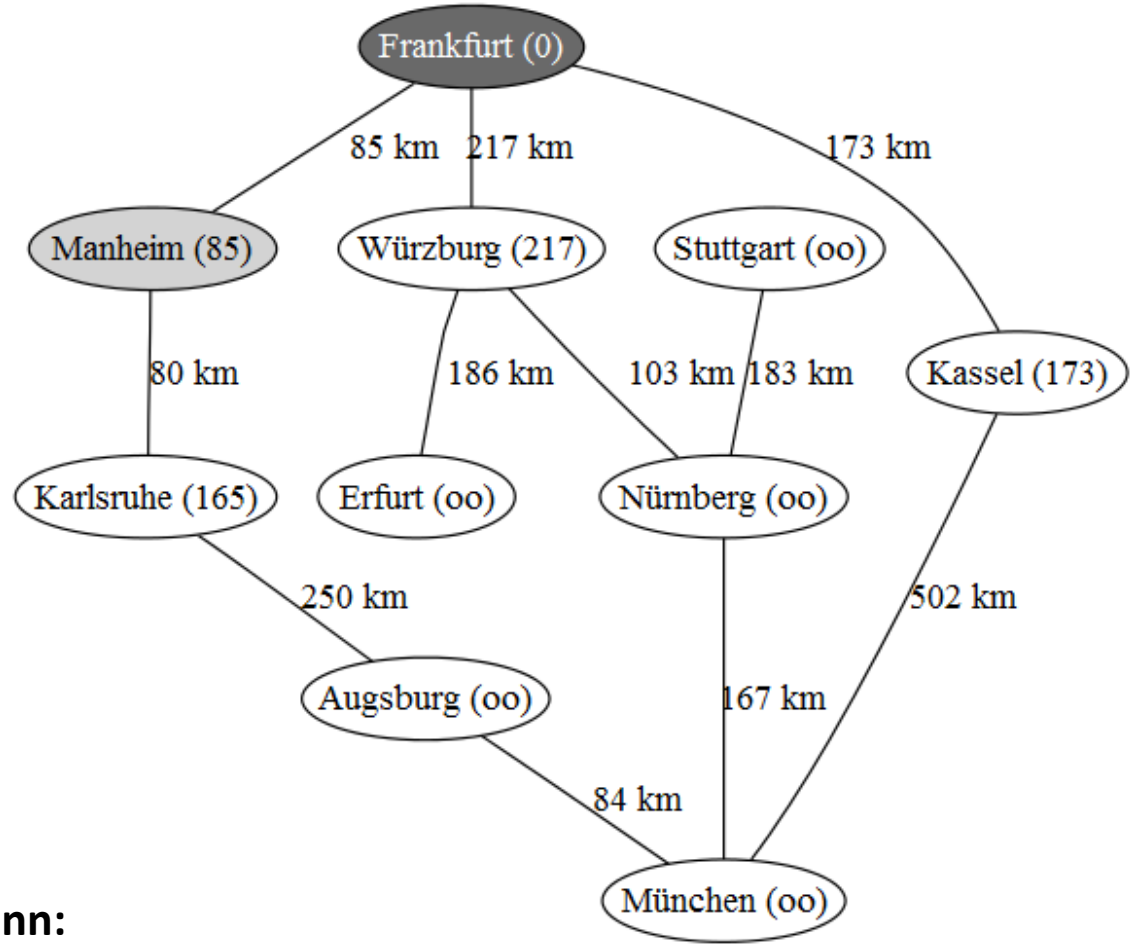
Beispiel: Frankfurt - München



Ersetze Wert wenn:

dist(start, current) + dist(current, next) < dist(start, next)

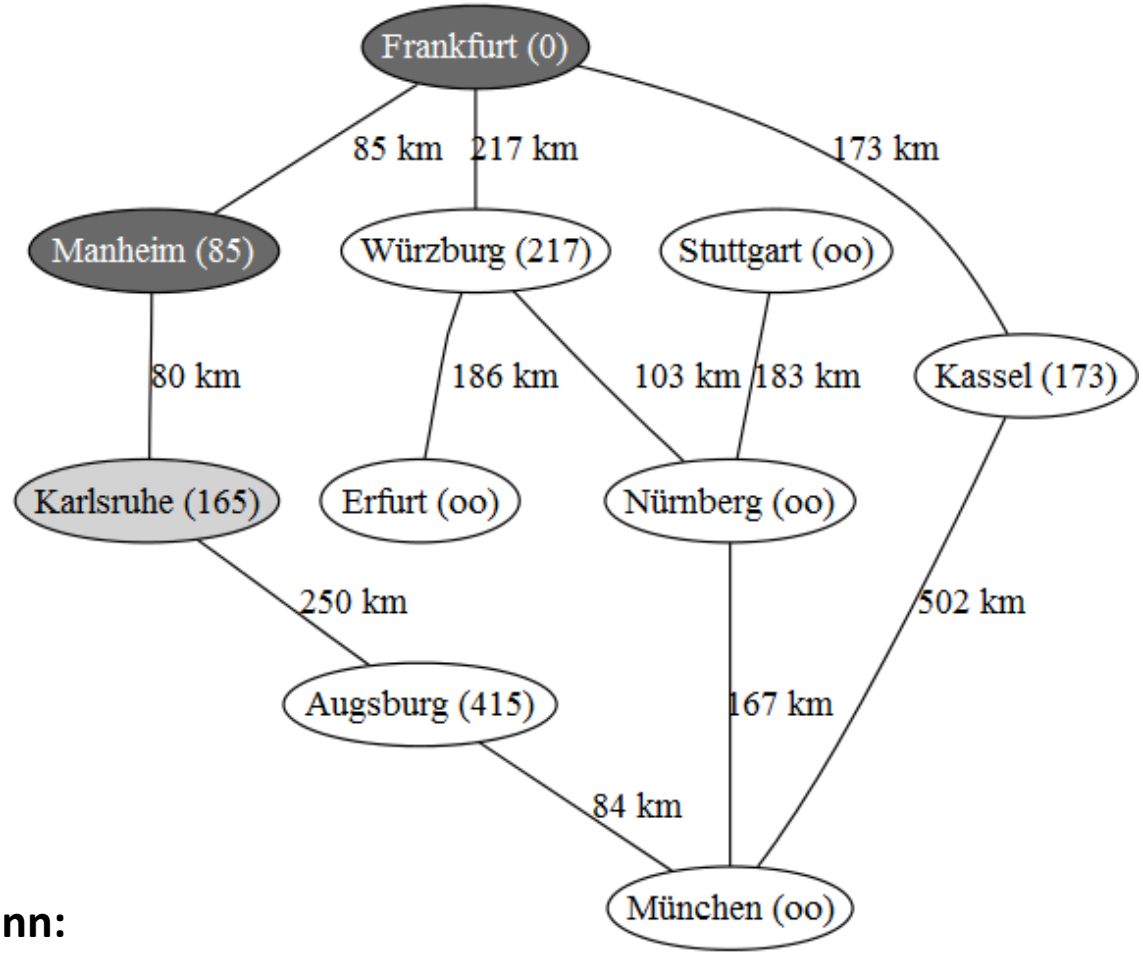
Beispiel: Frankfurt - München



Ersetze Wert wenn:

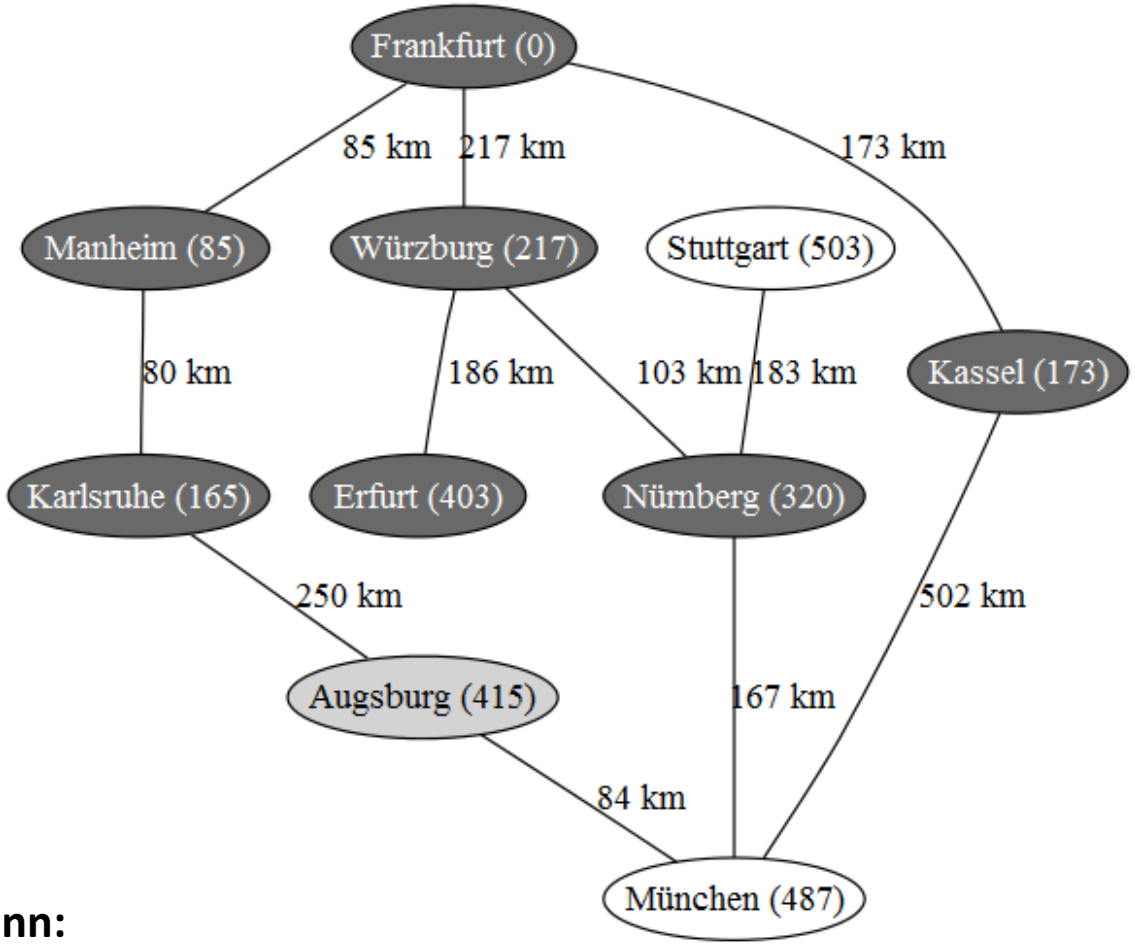
dist(start, current) + dist(current, next) < dist(start, next)

Beispiel: Frankfurt - München



Ersetze Wert wenn:
dist(start, current) + dist(current, next) < dist(start, next)

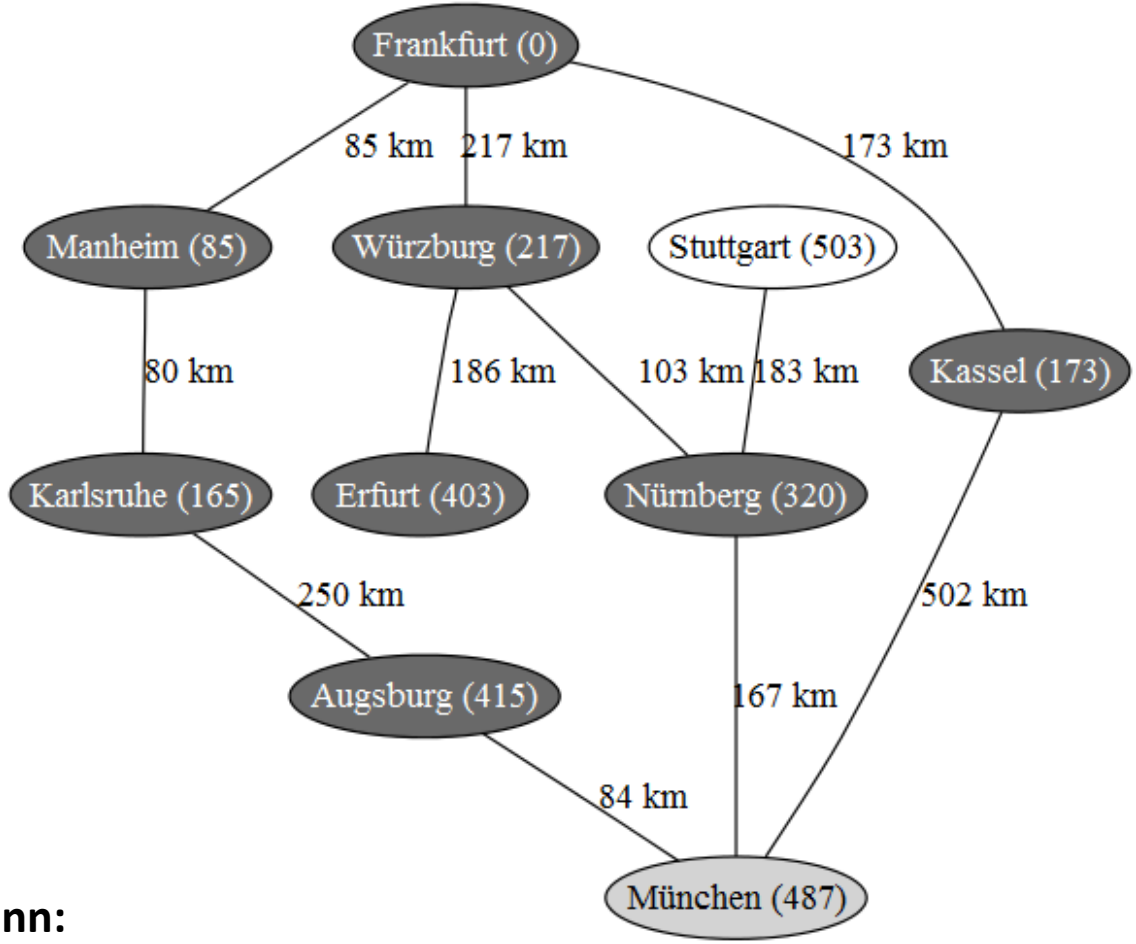
Beispiel: Frankfurt - München



Ersetze Wert wenn:

dist(start, current) + dist(current, next) < dist(start, next)

Beispiel: Frankfurt - München



Ersetze Wert wenn:

dist(start, current) + dist(current, next) < dist(start, next)

Dijkstra - Beispielproblem

ACM Problem: 523 Minimum Transport Cost

Input: Mehrere Testfälle mit jeweils:

- Einer Kostenmatrix der Form N x N
- Zusatzkosten für die Durchfahrt einer Stadt (nicht bei Start- und Zielstadt)
- Mehrere Paare von Start- und Zielknoten

Output: Pro Wertepaar von Start- und Zielknoten:

```
From [Startknoten] to [Zielknoten] :
Path: [Startknoten]-->[…]-->[…]-->[Zielknoten]
Total cost : [Minimale Kosten]
```

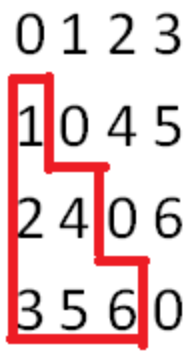
Dijkstra - Verbesserungen

Minimierung des Speicherbedarfs bei der Kostenmatrix :

Speichern der unteren linken Dreiecksmatrix

Vorraussetzung: ungerichteter Graph (Symmetrische Matrix)

Beispiel:




```
// Zugriff auf ein Array Element
int get(int[][] array, int row, int col) {
    if(row == col)
        return 0;
    if(row < col)
        return array[col][row];
    return array[row][col];
}
```

Dijkstra - Verbesserungen

Änderung der Datenstruktur für den Zugriff auf den minimalen Knoten zu Beginn jedes Schleifendurchlaufs

Operation	Lineare Liste	Sortierte Liste	(Min-)Heap	Unbalancierter Binärbaum	Fibonacci-Heap
insert	O(1)	O(n)	O(log n)	O(log n)*	O(1)
accessMin	O(1)	O(1)	O(1)	O(1)	O(1)
extractMin	O(n)	O(1)	O(log n)	O(1)	O(log n)*
decreaseKey	O(1)	O(n)	O(log n)	O(log n)*	O(1)*
remove	O(n)	O(n)	O(log n)	O(1)¹	O(log n)*
merge	O(1)	O(n + m)	O(m log(n+m))	O(n + m)	O(1)

Implementierung eines Fibonacci-Heaps in Java erhältlich unter: http://www.jgrapht.org

Dijkstra - Verbesserungen

Direkten Zugriff auf alle existierenden Kanten eines Knotens:

$\Rightarrow$ Knoten-Kanten-Matrix

Implementierung:

```
int[][] edges = new int[nodes][nodes];
int[] edgeCount = new int[nodes];
```

Beispiel: Iterator über alle Kanten des Knotens startNode:

```
for(int i = 0; i < edgeCount[startNode]; i++) {
        final int finishNode = edges[startNode][i];
        final int edgeCost = cost[startNode][finishNode];
}
```

Hamiltonkreisproblem

Beschreibung:

- Ein Hamiltonkreis ist ein geschlossener Pfad in einem Graphen, der jeden Knoten genau einmal enthält.

Laufzeit:

- O(n!) => NP-Vollständig

- Bekannte Aufgabenstellung: Traveling Salesman Problem

Hamiltonkreis - Beispielproblem

ACM Problem: 775 Hamiltonian Cycle

Input: Mehrere Testfälle mit jeweils:

- Anzahl der Knoten des Graphen
- Mehrere Kanten zwischen zwei Knoten ohne Gewichtung

Output: Hamiltonkreis beginnend und endend mit dem Knoten 1

Beispiele:

```
1 2 4 3 1
1 3 2 5 4 6 1
```

Hamiltonkreis - Implementierung

```
// path = {0, 1, ..., nodes - 1};
boolean hasNext = true;
while (hasNext) {
	// Überprüfen ob der Pfad einen Zyklus besitzt
	final int position = hasCycle(nodes, path, containsEdge);
	if (position < 0) {
		// Ausgabe der Lösung
		printSolution(path, nodes);
		hasNext = false;
	} else {
		// Weiterschalten zum nächsten Pfad
		hasNext = nextPath(nodes, path, position);

		// Keine weiteren Möglichkeiten vorhanden, kein Zyklus  existiert
		if (!hasNext) {
			System.out.println("N");
		}
	}
}
```

Hamiltonkreis - Implementierung

```
private static int hasCycle(int nodes, int[] path, boolean[][] containsEdge) {
    final int lastIndex = nodes - 1;
    for (int i = 0; i < lastIndex; i++) {
        if (!containsEdge[path[i]][path[i + 1]]) {
            return i;
        }
    }

    // Geschlossener Kreis
    if(containsEdge[path[lastIndex]][path[0]]) {
        return -1;
    } else {
        return lastIndex;
    }
}
```

Hamiltonkreis - Implementierung

```
private static boolean nextPath(int nodes, int[] path, int position) {
    for (int i = position; i > 0; i--) {
        // Suche in path rechts von i nach der nächst größeren Zahl
        final int index = getNextGreaterNumber(path, nodes, i);
        if (index >= 0) {
            int temp = path[i];
            path[i] = path[index];
            path[index] = temp;
            Arrays.sort(path, i + 1, nodes);
            return true;
        }
    }
    return false;
}
```

Hamiltonkreis - Implementierung

```
private static int getNextGreaterNumber(int[] path, int nodes, int i) {
    int oldNumber = path[i];
    int newNumber = nodes;
    int index = -1;

    for (int j = i + 1; j < nodes; j++) {
        int number = path[j];
        if (number < newNumber && number > oldNumber) {
            newNumber = number;
            index = j;
        }
    }
    return index;
}
```

Vielen Dank für die Aufmerksamkeit

Fragen?