

# Aufgabe

**Implementieren Sie den Dijkstra-Algorithmus. Verwenden Sie Adjazenzlisten zur Repräsentierung der Kanten und eine (heap-basierte) erweiterten Prioritätswarteschlange zur Verwaltung der Randknoten, d.h. der Knoten, die von  $S_k$  aus erreicht worden sind und Kosten kleiner  $\infty$  haben. In der Prioritätswarteschlange soll kein Knoten doppelt auftreten. Sie dürfen Ergebnisse aus vorherigen Aufgaben verwenden.**

# Rahmenwerk

<b>Dijkstra.java</b>	<b>Hauptprogram</b>
<b>Graph.java</b>	<b>Graph als Adjazenzliste</b>
<b>Edge.java</b>	<b>Kante im Graph</b>
<b>DijkstraResult.java</b>	<b>Datenstruktur für das Ergebnis eines Dijkstra Algorithmus</b>
<b>200nodes.graph</b>	<b>Beispielgraph mit 200 Knoten</b>
<b>10nodes.graph</b>	<b>Beispielgraph mit 10 Knoten</b>
<b>DijkstraSolver.java</b>	<b>zu implementieren: Dijkstra Algorithmus</b>

# Auswertung

- **Graphen mit 200, 1 000, 10 000 und 100 000 Knoten**
- **Dual PIII-1000 mit 1GHz unter Linux**

# Einreichungen

- **Ashwani Mehlem**
- **Daniel Brack, David Zills**
- **Jan Mussler**
- **Jan Scherer**
- **Lars Krecklau, Michael Gubesch**
- **Stephan Günnemann, Patrick Zimmer, Sabrina von Styp**
- **Sven Kulle**
- **Thorsten Sattler**
- **Thomas Siegbert, Marcus Kucay, Florian Klumb**
- **Volker Kamin, Martin Zimmermann, Martin Plücker**

# Ergebnisse: 200 Knoten

<b>Teilnehmer</b>	<b>Zeit</b>
<b>Ashwani Mehlem</b>	<b>31 ms</b>
<b>Daniel Brack, David Zils</b>	<b>–</b>
<b>Jan Mussler</b>	<b>18 ms</b>
<b>Jan Scherer</b>	<b>25 ms</b>
<b>Lars Krecklau, Michael Gubesch</b>	<b>20 ms</b>
<b>Stephan Günnemann, Patrick Zimmer, Sabrina von Styp</b>	<b>29 ms</b>
<b>Sven Kulle</b>	<b>21 ms</b>
<b>Torsten Sattler</b>	<b>–</b>
<b>Markus Kucay, Thomas Siegbert, Florian Klumb</b>	<b>40 ms</b>
<b>Volker Kamin, Martin Zimmermann, Martin Plücker</b>	<b>–</b>

# Ergebnisse: 1000 Knoten

<b>Teilnehmer</b>	<b>Zeit</b>
<b>Ashwani Mehlem</b>	–
<b>Daniel Brack, David Zils</b>	–
<b>Jan Mussler</b>	<b>36 ms</b>
<b>Jan Scherer</b>	<b>48 ms</b>
<b>Lars Krecklau, Michael Gubesch</b>	<b>42 ms</b>
<b>Stephan Günнемann, Patrick Zimmer, Sabrina von Styp</b>	<b>151 ms</b>
<b>Sven Kulle</b>	<b>45 ms</b>
<b>Torsten Sattler</b>	–
<b>Markus Kucay, Thomas Siegbert, Florian Klumb</b>	<b>59 ms</b>
<b>Volker Kamin, Martin Zimmermann, Martin Plücker</b>	–

# Ergebnisse: 10000 Knoten

<b>Teilnehmer</b>	<b>Zeit</b>
<b>Ashwani Mehlem</b>	–
<b>Daniel Brack, David Zils</b>	–
<b>Jan Mussler</b>	<b>79 ms</b>
<b>Jan Scherer</b>	<b>1009 ms</b>
<b>Lars Krecklau, Michael Gubesch</b>	<b>212 ms</b>
<b>Stephan Günнемann, Patrick Zimmer, Sabrina von Styp</b>	<b>96911 ms</b>
<b>Sven Kulle</b>	<b>1633 ms</b>
<b>Torsten Sattler</b>	–
<b>Markus Kucay, Thomas Siegbert, Florian Klumb</b>	<b>735 ms</b>
<b>Volker Kamin, Martin Zimmermann, Martin Plücker</b>	–

# Ergebnisse: 100000 Knoten

<b>Teilnehmer</b>	<b>Zeit</b>
<b>Ashwani Mehlem</b>	–
<b>Daniel Brack, David Zils</b>	–
<b>Jan Mussler</b>	514 ms
<b>Jan Scherer</b>	559506 ms
<b>Lars Krecklau, Michael Gubesch</b>	1435 ms
<b>Stephan Günnemann, Patrick Zimmer, Sabrina von Styp</b>	>24h
<b>Sven Kulle</b>	654886 ms
<b>Torsten Sattler</b>	–
<b>Markus Kucay, Thomas Siegbert, Florian Klumb</b>	345469 ms
<b>Volker Kamin, Martin Zimmermann, Martin Plücker</b>	–

# Ergebnis: Das schnellste Programm

**1. Platz:**

Jan Mussler

**2. Platz:**

Lars Krecklau, Michael Gubesch

**3. Platz:**

Markus Kucay, Thomas Siegbert, Florian Klumb

# Ergebnis: Das kürzeste Programm

```
cat *.java | tr -cd '(; ' | wc -c
```

<b>Teilnehmer</b>	<b>Länge</b>
<b>Jan Mussler</b>	<b>129</b>
<b>Jan Scherer</b>	<b>115</b>
<b>Lars Krecklau, Michael Gubesch</b>	<b>166</b>
<b>Stephan Günnemann, Patrick Zimmer, Sabrina von Styp</b>	<b>67</b>
<b>Sven Kulle</b>	<b>74</b>
<b>Markus Kucay, Thomas Siegbert, Florian Klumb</b>	<b>237</b>
<b>Ashwani Mehlem</b>	<b>103</b>
<b>Daniel Brack, David Zils</b>	<b>126</b>
<b>Torsten Sattler</b>	<b>130</b>
<b>Volker Kamin, Martin Zimmermann, Martin Plücker</b>	<b>91</b>

# Ergebnis: Das kürzeste Programm

## 1. Platz:

Stephan Günnemann, Patrick Zimmer, Sabrina von Styp

## 2. Platz:

Sven Kulle

## 3. Platz:

Jan Scherer

```

public DijkstraResult doIt(Graph g, int startNode) {
    PriorityQueue q=new PriorityQueue(N);

    for(int n=0;n<N;++n) result.setRank(n,0);
    q.Put(new HeapElement(startNode, startNode, 0));

    for(int k=0;k<N;++k) {
        HeapElement w=q.Min() ; q.GetMin();
        result.setRank(w.vertex,k);
        result.setCost(w.vertex,w.cost);
        result.setPred(w.vertex,w.pred);

        for(int i=0;i<g.get(w.vertex).size();++i) {
            Edge e=g.get(w.vertex,i);
            if(result.getRank(e.to)==0) {
                c=w.cost+e.cost;
                if(q.Contains(e.to)) {
                    if(c<q.Get(e.to).cost) {
                        q.Update(new HeapElement(e.to, w.vertex, c));
                    }
                } else {

```

```
q.Put(new HeapElement(e.to,w.vertex,c));
```

```
}
```

```
}
```

```
}
```

```
}
```

```
return result;
```

```
}
```

# Priority Queue

Die Priority Queue erlaubt die Operationen

- contains
- update
- get

Diese werden durch ein zusätzliches Feld realisiert, die für jeden Eintrag (Knotennummer) speichert, an welcher Stelle er auf dem Heap liegt. D.h. es gilt

```
heap[pos[v]].vertex=v or pos[v]=-1
```