# Aufgabe 4: Tourenprobleme

In dieser Aufgabe wird einmal ein Kantenbezogener Algorithmus und ein Eckenbezogener Algorithmus implementiert. Verglichen wird die Komplexität.

## Aufgabenstellung

Implementieren Sie nun unter Verwendung der ADT Graph folgende zwei Algorithmen, **so, wie sie in der Vorlesung vorgestellt wurden**:

1. **hierholzer**: Der Algorithmus von Hierholzer, so wie er in der Vorlesung vorgestellt wurde. Zur Überprüfung sollen in die Datei `HiHoTour.log` die gefundenen Teilzyklen und die finale Tour geschrieben werden, so dass der Ablauf des Algorithmus nachvollziehbar wird. **Schnittstelle**: `hierholzer:hierholzer(<Filename>)`; Rückgabewert: `[<Eulerkreis: [a,b,b,c,c,a]>]` sowie eine Datei `HiHoTour.log`;
   bzw. `hierholzer:hierholzer(<Graph>)`. Sowie `hierholzer:hierholzerT(<Graph>)`; mit obigem Rückgabewert, jedoch ohne weitere Ausgaben (keine log-Dateien oder io-Ausgaben) für die Zeitmessung.
2. **mede**: Den Algorithmus der Methode der Einführung der dichtesten Ecke, so wie er in der Vorlesung vorgestellt wurde. Zur Überprüfung sollen in die Datei `MeDETour.log` der aktueller Kreis mit Kosten und die dichteste Ecke geschrieben werden, so dass der Ablauf des Algorithmus nachvollziehbar wird. Die Startecke in der Schnittstelle dient lediglich für die Tests und beschränken den Algorithmus in seiner Funktionalität nicht!
   **Schnittstelle**: `mede:mede(<Filename>,<Startecke>)`; Rückgabewert: `{<Kosten der Hamiltontour>,<Hamiltontour: [a,5,b,3,c,5,d,1,a]>}` sowie eine Datei `MeDETour.log`;
   bzw. `mede:mede(<Graph>,<Startecke>)`. Sowie `mede:medeT(<Graph>,<Startecke>)`; mit obigem Rückgabewert, jedoch ohne weitere Ausgaben (keine log-Dateien oder io-Ausgaben) für die Zeitmessung.
3. Für die Laufzeitmessungen erstellen Sie bitte Versionen der Algorithmen ohne print-Ausgabe (`hierholzerT` bzw. `medeT`)! Führen Sie mit diesen Versionen aussagenkräftige Messungen durch (**Einheit: ms!**), mit der Sie die Laufzeitkomplexität der einzelnen Algorithmen nachweisen können. **Im Entwurf** sind für die Messungen bereits die Parameter der einzelnen Messungen anzugeben: welches Ziel (z.B. Laufzeitkomplexität) soll geprüft werden und welche Parameter (z.B. Anzahl Ecken, Grad der Ecken, Gewichte an den Kanten etc) werden in welchem Versuch mit welchen Erwartungen verändert. **Erstellen Sie aus den Messungen ein pdf,** indem die Messungen dokumetiert werden (Versuchsaufbau, Resultate, Interpretation, geforderte Nachweise etc.). Sofern Ihre Messungen die erwartete Komplexität **nicht bestätigen**, ist die Implementierung zu verbessern oder ausführlich herzuleiten, warum Ihre Implementierung eine andere Laufzeitkomplexität hat.

Auf der Hauptseite zur Vorlesung sind Graphen vorgegeben. Die Attributwerte stellen die Kosten der Kante dar. Die Graphen 09, 10 ,11, 12 sowie 13 (mede) und 01, 09, 10 sowie 13 (hierholzer) sind zum Test der Korrektheit zu verwenden. Führen Sie mit Ihren Algorithmen eine **Zeitmessung** durch (aufg4test.beam, Aufruf `aufg4test:zeitmessung()`) und **geben die damit erzeugte Datei (sowie die erzeugten log-Dateien) mit ab**. Für diesen Test werden die folgenden Graphen (im gleichen Ordner wie die beam Dateien) benötigt: testaufg4m.graph, testaufg4h.graph und graph_11.graph sowie graph_de.graph. Darüber hinaus sind für die Zeitmessung (**Einheit: ms!**) Graphen mittels dem Tool gengraph zu generieren und zu verwenden. (ausschließlich ungerichtet!). Zur Zeitmessung können Sie folgendes Tool verwenden: zeit4hm.beam. und zeit4mm.beam.

Die zugehörigen Dateien heißen `hierholzer.erl` und `mede.erl`. Weitere Dateien neben `adtgraph.erl` und `util.erl` sind nicht zulässig.

# Abnahme

**Der Entwurf** für diese Aufgabe ist in das Referat zu integrieren.

**Am Tag vor dem Referatstermin** ist bis spätestens **20:00 Uhr** (Achtung: Prüfungsrelevanter Termin!) die Abgabe des Referates zu erfolgen (schriftliche Ausarbeitung, Code, *.log Dateien etc., Präsentation). Nachreichungen sind nicht möglich.

Am Tag des Referates besteht **während aller Vortragstermine Anwesenheitspflicht**. Die Nutzung von Rechnern während der Vorträge ist für die Zuhörer*innen untersagt.

**Beachten Sie die Regularien zum Referat!**

# Graphentheoretische Konzepte und Algorithmen

Aufgabe 4







## Algorithmus von Hierholzer

***Algorithmus von Hierholzer**: Gegeben sei ein eulerscher Graph G = (V,E). Ausgabe ist die Eulertour K.*

***Schritt 1**: Man wähle einen beliebigen Knoten $v_0$ in G und konstruiere von $v_0$ ausgehend einen Unterkreis (Eulerkreis) K in G, der keine Kante in G zweimal durchläuft.*

***Schritt 2**: Wenn K eine Eulertour ist, gehe zu 4. Andernfalls: gehe zu Schritt 3.*

***Schritt 3**: 1. Vernachlässige alle Kanten des Eulerkreises K.*
2. *Eine beliebige Ecke von K, zu der nicht vernachlässigte Kanten inzident sind, startet man einen weiteren Eulerkreis K' (analog Schritt 1).*
3. *Füge in K den zweiten Kreis K' ein, indem die Startecke von K' in K durch alle Ecken von K' ersetzt wird. Gehe nun zu Schritt 2.*

***Schritt 4**: Gebe K als Eulertour aus.*

# Schnittstellen

```
hierholzer:hierholzer(<Filename>);
```

Rückgabewert:

```
[<Eulerkreis: [a,b,b,c,c,a]>]
```

sowie eine Datei `HiHoTour.log` mit den Teilzyklen;

Für zB Laufzeitmessung:

```
hierholzer:hierholzer(<Graph>);
hierholzer:hierholzerT(<Graph>);
```

Benötigte Dateien: *hierholzer.erl* und *adtgraph.erl* sowie *util.erl*







# Methode der Einführung der dichtesten Ecke

*Der Abstand einer Ecke v von einer Kantenfolge W ist definiert als d(v,W) = min{d(v,u) | u ist eine Ecke von W}, wobei d(v,u) die Kosten dieser Kante sind. Die Ecke v, die nicht in W liegt, wird als* ***dichteste Ecke*** *zu W bezeichnet, wenn für jede andere nicht in W befindliche Ecke x d(v,W) ≤ d(x,W) gilt.*

***Methode der Einführung der dichtesten Ecke****: Gegeben sei ein Graph $K_n = (V,E)$ mit $V= \{v_1, v_2, \ldots, v_n\}$. Der aktuelle gewählte Kreis $W_i$ wird stets neu nummeriert und ist gegeben durch $W_i = u_1u_2 \ldots u_iu_1$, $u_j \in V$ . Diese neue Nummerierung verändert nicht die Reihenfolge der bisher gewählten Ecken!*

*Schritt 1: Man wähle eine beliebige Ecke $u_1 = v_j \in V$ als Startecke und setze $W_1 = u_1$.*

*Schritt 2: Aus der Zahl der n−i Ecken, die bisher noch nicht gewählt worden sind, ermittle man eine Ecke $u_{i+1} = v_k$, die am dichtesten zu $W_i$ liegt. Sei $W_i = u_1u_2 \ldots u_iu_1$. Man bestimme dann, welche der Kantenfolgen (Kreise) $u_1u_{i+1}u_2u_3 \ldots u_iu_1$, $u_1u_2u_{i+1}u_3 \ldots u_iu_1$, …, $u_1u_2u_3 \ldots u_iu_{i+1}u_1$ die kürzeste ist. Es sei $W_{i+1}$ diese kürzeste Kantenfolge. Man nummeriere sie, wenn nötig, neu als $u_1u_2 \ldots u_{i+1}u_1$. Man setze i:= i + 1.*

*Schritt 3: Wenn $W_i$ alle Ecken beinhaltet, beende den Algorithmus, sonst führe Schritt 2 aus.*

# Schnittstellen

`mede:mede(<Filename>,<Startecke>)`

Rückgabewert:

`{<Kosten der Tour>,<Tour: [a,5,b,3,c,5,d,1,a]>}`
sowie eine Datei `MeDETour.log` mit aktuellem Kreis, dessen Kosten und der dichtesten Ecke;

Für zB Laufzeitmessung:

```
mede:mede(<Graph>,<Startecke>);
mede:medeT(<Graph>,<Startecke>);
```

mede ist eine Form „Nearest Insertion Heuristic“, nicht zu verwechseln mit „Nearest Neighbor Heuristic“

Benötigte Dateien: *mede.erl* und *adtgraph.erl, util.erl*







# Durchzuführende Tests

```
1> aufg4test:zeitmessung().
Graph mit 97 Ecken und 4656 Kanten aus 'testaufg4m.graph' importiert.
Graph mit 987 Ecken und 5922 Kanten aus 'testaufg4h.graph' importiert.
        hierholzer
.............
        mede
.............
        >>--<<
Graph mit 23 Eckеn und 44 Kanten aus 'graph_de.graph' importiert.
Graph mit 6 Ecken und 15 Kanten aus 'graph_11.graph' importiert.
        hierholzer
        mede
        >>--<<
ok
2>
```

MeDETour.log
results814542.log
HiHoTour.log

```
hierholzer: 259...ms, im Durchschnitt: 199.461538ms
    Eulertour A (11844)[1,2,2,3,3,4,4,5,5,2,2,272,272,857,
877,838,838,839,839,840,840,896,896,895,895,926,926,925,92
...
485,485,751,751,44,44,987,987,1];
mede: 3750ms, im Durchschnitt: 288.461538ms
    Hamiltontour A (195/1086)[1,11,5,11,63,12,50,11,65,11,
14,12,54,11,42,11,29,11,66,11,72,11,37,12,71,11,40,11,17,1
...
11,97,11,22,11,61,11,74,11,8,11,9,11,36,11,1];
Gesamtzeit über alle Tests: 6344ms
    hierholzer Eulertour B (88)[1099,90449,90449,76199,761
...
28199,28199,44149,44149,33699,33699,49090,49090,44149,4414
    mede Hamiltontour B (13/52)[11,5,55,5,66,18,22,6,33,3,
11,97,11,22,11,61,11,74,11,8,11,9,11,36,11,1] mit Kosten 1086
  Aktuelle beste Tour zur dichtesten Ecke 55: [11,5,55,5,11] mit Kosten 10
  Aktuelle beste Tour zur dichtesten Ecke 66: [11,5,55,5,66,15,11] mit Kosten 25
  Aktuelle beste Tour zur dichtesten Ecke 33: [11,5,55,5,66,10,33,10,11] mit Kos
  Aktuelle beste Tour zur dichtesten Ecke 44: [11,5,55,5,66,10,33,3,44,15,11] mi
  Aktuelle beste Tour zur dichtesten Ecke 22: [11,5,55,5,66,18,22,6,33,3,44,15,1
Finale Tour: [11,5,55,5,66,18,22,6,33,3,44,15,11] mit Kosten 52
```

```
485,485,751,751,44,44,987,987,1]
    Neue Teiltour gefunden: [1099,90449,90449,6
    Neue Teiltour gefunden: [1099,12099,12099,4
20099,20099,12099,12099,18119,18119,23569,23569
30159,30159,49090,49090,28199,28199,44149,44149
44149,44149,34119,34119,1099]
    Neue Teiltour gefunden: [90449,76199,76199,
60599,44149,44149,50999,50999,66119,66119,67659
55129,60599,60599,68199,68199,76199,76199,70199
30999,90449]
inale Tour: [1099,90449,90449,76199,76199,67659
60599,60599,44149,44149,50999,50999,66119,66119
55129,55129,60599,60599,68199,68199,76199,76199
```

Benötigt: testaufg4m.graph
testaufg4h.graph
graph_11.graph
graph_de.graph