

Informatik Hochschule für angewandte Wissenschaften Hamburg

Arbeitsaufwand

- Laut Modulhandbuch für das Semester (16 Wochen):
 - **Vorlesung:** 42h
 - **Praktika:** 16h
 - **Eigenstudium:** 126h

- Real** (Feiertags bedingt) für das Semester (15 Wochen):
 - Vorlesung/Woche: 3h (11 Termine)
 - Praktika/3Wochen: 3h (4 Termine)
 - Eigenstudium/Woche: 6,3h (15 Termine)

Hier 1h ≙ 60 Minuten, Im Modulhandbuch 1h ≙ 45 Minuten

Organisation der Veranstaltung

- www.informatik.haw-hamburg.de/~klauck/graphkoal.html
- 4 Stunden (2 Viertel) pro Woche
 - 4 Stunden Vorlesung (11 Termine)
 - ♦ Raum 1281
 - ♦ Dienstag 08¹⁵–11³⁰ **Letzter Termin: 27.05**
 - 4 Stunden (2 Viertel) Praktikum (4 Termine)
 - ♦ Raum 1165
 - ♦ Mittwoch 12³⁰–15⁴⁵ je nach Gruppenzugehörigkeit
- Wie üblich
 - Vorlesung zur Vermittlung der *theoretischen* Grundlagen
 - Praktikum zur praktischen Erfahrung



6

Back to the roots



8

Vorlesung? offen!



10

Vorlesung? Lernen?

- Lernen ist ein **aktiver und konstruktiver Prozess**
- Lernen als **individueller und sozialer Prozess**
 - Individualität: Entwicklungsstand, Herausforderung, Interesse, **Begeisterung**
 - Sozialität: Aufgaben, Probleme, sozialer Vergleich, Vorbild
- **Intrinsische Lernmotivation:** Lernmotivation ergibt sich aus den eigenen (vagen) Lebenswünschen und den Herausforderungen der Umwelt
- Individualität des Lernenden anerkennen
 - Vorhandene Lernfreude nutzen
 - Der eigenen Lernmotivation eine Chance geben
 - **Selbstkontrolle:** Eigene Lernziele, Wochenpläne, Lerntagebücher etc.
- Motivation: "Du sollst wollen!"
 - Freiheit und Eigenverantwortung kann man nicht befehlen
 - Mit Noten kann man nicht wirklich Lernen "motivieren"
- **Selbstgesteuertes Lernen** hat einen langfristigen und flexibel nutzbaren Lernertrag

11

Wie geht das?



12

Lernen!

- **Semi-reale Aufgaben formulieren,** kontextualisiertes Lernen
- Den **Umgang mit Problemen** lernen lassen (Der Weg ist auch ein Ziel: Lernen lernen)
- **Authentische Fragen** stellen
 - Richtig fragen will gelernt sein
 - Prüfungsfragen unterminieren (untergraben und zerstören) das Selbstbewusstsein
- Stimulation des Lernens durch **Themen, Aufgaben, Probleme und Fragen**
 - Themen finden: Gespräche mit Anderen, Medien, Bibliothek, WWW
 - Aufgaben finden und formulieren: Literatur, WWW, selbst formulieren
 - Probleme aufgreifen und diskutieren
 - Fragen: Pseudo-Fragen und Prüfungsfragen vermeiden!!
- Vorlesungen sind nur wirksam, wenn die Lernenden mitspielen!

13

Wozu Aufgaben?

Machst du es richtig?

(Geschlossene Aufgabe: *Wie viel ist $49 \cdot 51$?*
Defizitperspektive)

oder

Wie machst du es?

(Offene Aufgabe: *Wie rechnest du $49 \cdot 51$?*
Entwicklungsperspektive)

Was will man erfahren, wenn eine
Aufgabe gestellt wird?



14

Praktikum / PVL

- Gruppenaufteilung: **2 Studierende** in einem Team (**Meldung bis zum 21.03.14!**)
- **PVL-Bedingungen**
 - **Baut stark auf vorbereitende Arbeit** auf!
 - **Empfehlung:** **vor** dem Praktikumstermin: Ziele festlegen, Arbeitsplan erstellen etc.; das Praktikum als Präsentation/Anwendung nutzen
 - **Details** siehe Aufgabenstellungen!
 - Bis **Sonntag Abend vor** dem Praktikumstermin: erste Skizze zusenden.
 - **Während des Praktikumstermins**: Befragung (ca. 15 min pro Team)
 - **Während des Praktikumstermins**: Disputation mit anderen Teams, dazu möglichst konkurrierende / unterschiedliche Ansätze der einzelnen Teams.
 - Die Frage ist **nicht ob** die Anforderungen erfüllt sind, **sondern wie!**
 - Am **gleichen Tag bis 19⁰⁰ Uhr**: Abgabe der geforderten Unterlagen (digital!)
 - Anwesenheitspflicht (**gesamte Praktikumszeit!**)
 - Erfolgreiche Bearbeitung **aller** Aufgaben
 - Einhaltung aller Termine



15

Praktikumsaufgaben

Erfolgreiche Bearbeitung **aller** Aufgaben:

- **Ausführliche Dokumentation** des Codes und rechtzeitige Abgabe (als *.zip < 1MB):
 - Eine **korrekte Implementierung**, die der formalen Beschreibung entspricht (keine Warnings/Errors beim Start)
 - **Kommentierung** der zentralen Eigenschaften/Ereignisse etc. im Code
 - **Ausführliche Beschreibung** zum Start der Software (ausführbare Datei ist mitzuliefern), ggf. zur **Architektur**
- **Anwesenheitspflicht** (180 Minuten bzw. 3 Zeitstunden)
- **Erfolgreiche Besprechung** der Aufgabe
- **Weitere Details:** siehe jeweilige Aufgabenstellung

16

Praktikumsaufgaben

- ♦ **Programmiersprache** Erlang/OTP
- ♦ **Themenschwerpunkte** siehe Aufgabenstellung
- ♦ **Teams** (je zwei) **verantwortlich für den gesamten Code** der Aufgabe: Architektur und Programmcode müssen gut (frei) erklärbar sein
- ♦ Alle **Lösungen aller Aufgaben** müssen **als eine Lösung** gleichzeitig laufen können
- ♦ **Übernahme von Code Dritter:** Ist maximal bei einer Aufgabe mit entsprechender Begründung (siehe Dokumentationskopf) zulässig.

17

Dokumentationskopf

Team: <Teamnummer>, <Namen der Teammitglieder> **Details siehe Vorlage**

Aufgabenaufteilung:

1. <Aufgaben, für die Teammitglied 1 verantwortlich ist>
2. <Aufgaben, für die Teammitglied 2 verantwortlich ist>

Quellenangaben: <Angabe von wesentlichen Quellen> <Namentliche Nennung von Studierenden der HAW, von denen Quellcode übernommen wurde>

Begründung für Codeübernahme: <Wurde Quellcode übernommen, ist hier ausführlich zu begründen, warum dies gemacht wird, warum diese Quelle ausgewählt wurde und wie der dadurch verlorene Lerneffekt auf andere Art und Weise sichergestellt wird.>

Bearbeitungszeitraum: <Datum und Dauer der Bearbeitung an der Aufgabe von allen Teammitgliedern>

Aktueller Stand: <Welche Teile der Software sind fertig inklusive Tests, welche sind fertig, aber noch nicht getestet, welche müssen noch implementiert werden>

Skizze: <Entwurfsskizze nach den Vorgaben gemäß Aufgabenstellung.>

18

Praktikumsaufgaben

Vier Aufgaben:

ADT Graph: Implementierung einer ADT Graph in Erlang/OTP. Diese wird in den nachfolgenden Aufgaben evtl. mit anderen Gruppen ausgetauscht werden!

Ecken: werden über eine Identität `V_ID` angesprochen

Kanten: sind Tupel zweier Ecken: `{V_ID1, V_ID2}`

Attribute: sind Schlüssel/Wert Paare der Form: `{Key, Value}`

Graph: besteht aus einer Menge von Ecken und einer Menge von Kanten. Der Nullgraph ist der initiale Graph (kein leerer Graph!)

Fehler: als allgemeine Fehlermeldung wird nil als Rückgabewert festgelegt.

19

Praktikumsaufgaben

- **Konstruktoen:**
 - `new_AIGraph()` // post: erzeugt einen initialen Nullgraph
// returns: Ein neuer Nullgraph
 - `addVertex(NewItem,Graph)` // post: fügt dem Graphen eine neue Ecke mit Identität
//NewItem (V_ID) zu.
// returns: modifizierter Graph
 - `deleteVertex(V_ID,Graph)` // post: löscht die Ecke mit ID V_ID aus dem Graphen
// und alle zu ihr inzidenten Kanten.
// returns: modifizierter Graph
 - `addEdgeU(V_ID1, V_ID2,Graph)` // post: fügt dem Graphen eine neue ungerichtete Kante
// zwischen V_ID1 und V_ID2 zu.
// returns: modifizierter Graph
 - `addEdgeD(V_ID1, V_ID2,Graph)` // post: fügt dem Graphen eine neue gerichtete Kante
// von V_ID1 nach V_ID2 zu.
// returns: modifizierter Graph
 - `deleteEdge(V_ID1, V_ID2,Graph)` // post: löscht die Kante zwischen den beiden gegebenen
// Ecken; wenn er gerichtet ist, die von V_ID1
// nach V_ID2

20

Praktikumsaufgaben

- **Selektoren:**
 - `isNil(Graph)` // post: prüft, ob der Graph ein Nullgraph ist
// returns: boolean
 - `getIncident(V_ID1,Graph)` // post: ermittelt alle zur Ecke V_ID1 inzidenten Kanten
// returns: Liste der Kanten
 - `getAdjacent(V_ID1,Graph)` // post: ermittelt alle zur Ecke V_ID1 adjazenten Ecken
// returns: Liste von Tupeln {s/t/u, Ecken-ID}
// wobei s bzw. t angibt, ob diese Ecke source oder
// target zu V_ID1 ist oder u für ungerichtet
 - `getVertexes(Graph)` // post: ermittelt alle Ecken des Graphen
// returns: Liste der Ecken-IDs
 - `getEdges(Graph)` // post: ermittelt alle Kanten des Graphen
// returns: Liste der Kanten

21

Praktikumsaufgaben

- **Selektoren:**
 - `getValE({V_ID1,V_ID2},Attr,Graph)` // post: ermittelt den Attributwert von Attr der Kante
// {V_ID1,V_ID2}
// returns: Attributwert, nil im Fehlerfall
 - `getValV(V_ID1,Attr,Graph)` // post: ermittelt den Attributwert von Attr der Ecke V_ID1
// returns: Attributwert, nil im Fehlerfall
 - `getAttrV(V_ID1,Graph)` // post: ermittelt alle Attribute der Ecke V_ID1
// returns: Liste der Attribute
 - `getAttrE({V_ID1,V_ID2},Graph)` // post: ermittelt alle Attribute der Kante {V_ID1,V_ID2}
// returns: Liste der Attribute

22

Praktikumsaufgaben

- **Mutator:**
 - `setValE(V_ID1,V_ID2,Attr,Val,Graph)` // post: setzt den Attributwert von Attr der Kante
// (V_ID1,V_ID2) auf Val, unabhängig davon, ob Attr
// bekannt ist oder nicht
// returns: modifizierter Graph
 - `setValV(V_ID1,Attr,Val,Graph)` // post: setzt den Attributwert von Attr der Ecke V_ID1 auf
// Val, unabhängig davon, ob Attr bekannt ist oder nicht
// returns: modifizierter Graph
- **Import/Export:**
 - `importGraph(Datei,Attr)` // post: importiert aus Datei graph den Graphen.
// ist Attr = cost haben Kanten in dieser Datei einen
// Attributwert (ggf. Wird der zweite Ignoriert!)
// Ist Attr = maxis haben die Kanten zwei Attributwerte
// Die Namen sind als V_IDs zu verwenden.
// returns: importierten Graph
 - `exportGraph(Graph,Datei)` // post: exportiert den Graphen in Datei.graph
// returns: true im Erfolgsfall, sonst nil

23

Praktikumsaufgaben

- (2) **Optimale Wege:** aufbauend auf Aufgabe (1) sind zwei Algorithmen für optimale Wege zu implementieren: *Bellman-Ford* und *Floyd-Warshall*, die beide auf den gleichen Graphen arbeiten! Beide Algorithmen sollen zum Vergleich über Möglichkeiten zur Messung der Anzahl der Zugriffe auf den Graphen verfügen. Die Kanten haben das reservierte Attribut `cost`, um die Kosten dort zu speichern.
- (3) **Flußprobleme:** aufbauend auf Aufgabe (1) sind zwei Algorithmen für Flußprobleme zu implementieren: *Ford und Fulkerson* sowie *Edmonds und Karp*, die beide auf den gleichen Graphen arbeiten! Beide Algorithmen sollen zum Vergleich über Möglichkeiten zur Messung der Anzahl der Zugriffe auf den Graphen verfügen. Die Kanten haben das reservierte Attribut `maxis`, indem als Tupel die Kapazität (`max`) und der aktuelle Fluß (`is`) gehalten werden: `{max,is}`.
- (4) **Tourenprobleme:** aufbauend auf Aufgabe (1) ist jeweils ein Algorithmus für Tourenprobleme zu implementieren: *Algorithmus von Hierholzer* (Eulertour) und die *Methode der Einführung der dichtesten Ecke* (Hamiltonkreise). Beide Algorithmen sollen zum Vergleich über Möglichkeiten zur Messung der Anzahl der Zugriffe auf den Graphen verfügen. Die Kanten haben das reservierte Attribut `cost`, um die Kosten dort zu speichern.

Beachten Sie: Als Quelle für die Algorithmen haben sich Fachbücher bestens bewährt. Die Verwendung von Wikipedia oder vorhandenem Code führte oft zu nicht lauffähigem Code!

24

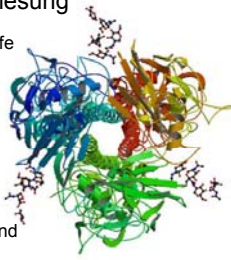
Darstellung von Graphen

- Graphen sind über Dateien (*.graph) einzulesen. Dabei ist folgendes Format zu verwenden:
#<gerichtet | ungerichtet>
<Name Ecke1>,<Name Ecke2>,<Kantenattribut1>,<Kantenattribut2>
Dabei sind die Namen der Ecken als Zeichenketten zu erfassen und die Attribute als ganze Zahlen.
- Beispiel:
#gerichtet
Hamburg,Bremen,123
Hamburg,Berlin,289
- Beispiel:
#ungerichtet
v1,v2,123,456
v3,v4,289,888

25

Inhalt der Vorlesung

- Graphentheoretische Grundbegriffe
- Suchstrategien, Kürzeste Wege
- Bäume, Wälder
- Flüsse und Tourenprobleme
- Grundlegende Eigenschaften von Petri-Netzen
- Berechenbarkeit, Erreichbarkeit und Erzeugbarkeit von Petri-Netzen



26

Termin-/Themenplan



Datum	VL 12:30-15:45	Thema
18.03.	VL1	Organisatorisches, Erlang/OTP
25.03.	VL2	Grundbegriffe
01.04.	VL3 p01	Und/Oder-Graphen, Zusammenhang, Speicherung von Graphen
08.04.	VL4 p01	Optimale Wege I: DFS, BFS, Dijkstra Algorithmus
15.04.	VL5 p01	Optimale Wege II: Dijkstra, FIFO/LIFO, Floyd-Warshall Algorithmus
22.04.	VL6 p02	Bäume und Wälder: spannende Bäume, Kruskal/Greedy, Prim Algorithmus
29.04.	VL7 p02	Flußprobleme: Max-flow/Min-cut, Ford und Fulkerson Algorithmus
06.05.	VL8 p02	Flußprobleme: Zuordnungs- und Transportprobleme
13.05.	VL9 p02	Tourenprobleme: Eulertour, chinesisches Briefträgerproblem
20.05.	VL10 p03	Tourenprobleme: Hamiltonsche Kreise, Handlungsreisendenproblem
27.05.	VL11 p03	Petri-Netze: Grundbegriffe, B/E-, S/T-, P/E-Systeme

27

Anforderungen

- Mathematische Grundlagen (MG/MGÜ)

insbesondere den Teil **Grundlagen**, und die Themen **Beweistechniken** und **vollständige Induktion** des Teils Techniken (ggf. unbedingt nacharbeiten!)

- Logik und Berechenbarkeit (LB/LBP)

insbesondere **induktive Definitionen** und **Rekursion** (ggf. unbedingt nacharbeiten!)



29

Basis-Literatur

- C. Klauck, C. Maas: *Graphentheorie und Operations Research für Studierende der Informatik*, HAW Hamburg, als *.pdf
- S.O. Krumke, H. Noltemeier: *Graphentheoretische Konzepte und Algorithmen*, Teubner, als *.pdf
- J. Clark, D.A. Holton: *Graphentheorie*, Spektrum, Originalausgabe als *.pdf
- R. Diestel: *Graphentheorie*, Springer-Verlag, als *.pdf
- A. Habel: *Graphersetzungssysteme*, Universität Oldenburg, als *.pdf
- H.J. Kreowski: *Algorithmen auf Graphen*, Universität Bremen, als *.pdf
- J. Padberg: *Petrinetze*, HAW Hamburg, als *.pdf
- V. Turau: *Algorithmische Graphentheorie*, Oldenburg, bei Books.Google
- L. Volkmann: *Graphen an allen Ecken und Kanten*, RWTH Aachen, als *.pdf
- K. Reich: *Konstruktivistische Didaktik*, Beltz, bei Books.Google



Zum Kauf
empfohlen!

30

Aufgabenstellung

Seien $A := \{m, a, f, i\}$ und $B := \{a, r, k\}$.

- (a) Welche der Relationen $R_i, i \in \{1, 2, 3, 4\}$ mit $\text{Typ } A \times B$ sind eine Abbildung bzw. keine Abbildung? (Begründung/Lösungsweg für jede Relation nicht vergessen!)

$$R_1 := \{(m, a), (a, a), (f, r), (i, k)\} \quad R_2 := \{(m, a), (f, a), (a, a), (i, a)\}$$

$$R_3 := \{(m, a), (a, r), (i, k), (a, a)\} \quad R_4 := \{(m, r), (a, k), (f, a), (f, k)\}$$

- (b) Geben Sie für die Relationen R_i , die eine Abbildung sind, an, welche der Eigenschaften injektiv, surjektiv und/oder bijektiv sie besitzen bzw. sie nicht besitzen. (Begründung/Lösungsweg nicht vergessen!)

Bei dieser Klausur (90 Minuten Lösungszeit) und vorher unbekannter Aufgabe:
 für sehr gute Bewertung *minimal* 3P in 3,8 Minuten Lösungszeit
 für ausreichende Bewertung *minimal* 3P in 7,6 Minuten Lösungszeit
 für sehr gute Bewertung *statistisch* 3P in 5,4 Minuten Lösungszeit
 für ausreichende Bewertung *statistisch* 3P in 11,2 Minuten Lösungszeit

31

Lösungen

b) R_1 ist eine Abbildung, da R_1 injektiv ist, da $\forall a \in B$ mit $a = a$ ein $m \in A$ existiert, das a abbildet.
 R_2 ist eine Abbildung, da $\forall a \in B$ mit $a = a$ ein $m \in A$ existiert, das a abbildet.
 R_3 ist eine Abbildung, da $\forall a \in B$ mit $a = a$ ein $m \in A$ existiert, das a abbildet.
 R_4 ist keine Abbildung, da $r \in B$ nicht abgebildet wird.

(c) R_1 ist eine Abbildung, da $\forall a \in B$ mit $a = a$ ein $m \in A$ existiert, das a abbildet.
 R_2 ist eine Abbildung, da $\forall a \in B$ mit $a = a$ ein $m \in A$ existiert, das a abbildet.
 R_3 ist eine Abbildung, da $\forall a \in B$ mit $a = a$ ein $m \in A$ existiert, das a abbildet.
 R_4 ist keine Abbildung, da $r \in B$ nicht abgebildet wird.

Kein Lösungsweg,
Keine Bearbeitungspunkte

32

Musterlösung

(b) (3P) R_1 ist nicht injektiv, da $m \neq a, m, a \in A$ ist, jedoch beide auf $a \in B$ abgebildet werden. R_1 ist surjektiv, da es zu jedem Element $x \in B$ ein Element $y \in A$ gibt mit $(y, x) \in R_1$. R_1 ist nicht bijektiv, da es nicht injektiv ist.

R_2 ist nicht injektiv, da alle Elemente aus A auf $a \in B$ abgebildet werden. R_2 ist nicht surjektiv, da es zu den Elementen $r, k \in B$ kein Element $y \in A$ gibt mit $(y, r) \in R_2$ bzw. $(y, k) \in R_2$. R_2 ist nicht bijektiv, da es weder injektiv noch surjektiv ist.

Bei dieser Klausur (90 Minuten Lösungszeit):

für sehr gute Bewertung	3P in 7,5 Minuten Lösungszeit
für ausreichende Bewertung	3P in 15 Minuten Lösungszeit

33

Aufgabenstellung

Wartegraph (3 Punkte)

Zur Verklemmungsentdeckung in verteilten Transaktionssystemen gibt es den Global Deadlock Detector (GDD), der die lokalen Wartegraphen zu einem globalen Wartegraphen zusammensetzt.

- (a) Kann es vorkommen, dass der GDD eine Verklemmung entdeckt, die real nicht existiert?
- (b) Kann es vorkommen, dass der GDD eine tatsächliche Verklemmung nicht entdeckt?

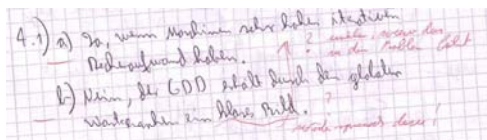
Begründen Sie jeweils Ihre Antwort.

Bei dieser Klausur (90 Minuten Lösungszeit) und vorher unbekannter Aufgabe:

für sehr gute Bewertung <i>minimal</i>	3P in 3,8 Minuten Lösungszeit
für ausreichende Bewertung <i>minimal</i>	3P in 7,6 Minuten Lösungszeit
für sehr gute Bewertung <i>statistisch</i>	3P in 5 Minuten Lösungszeit
für ausreichende Bewertung <i>statistisch</i>	3P in 10 Minuten Lösungszeit

34

Lösung



Oberflächliche Antwort ohne konkreteren Bezug zur Aufgabenstellung führt oft zu **Widersprüchen**. Im Beispiel:

- Antwort unter a), das ein falscher Eindruck entsteht, wird durch hohen iterativen Rechenaufwand begründet, was in dem Sinne zu keinem klaren Bild führt.
- Antwort unter b), das kein falscher Eindruck entsteht, wird durch ein klares Bild begründet, was in dem Sinne keine falschen Eindrücke zulassen würde.

35

Lösungen

Aufgabe 4

a) Ja, das kann vorkommen.
Wenn der GDD alle relevanten Wartegruppen hat, aber die zu unterschiedlichen Zeiten da sein angekommen sind, kann es sein, dass er ein Phantom Deadlock entdeckt, der aber in der Realität gar nicht vorhanden ist.

A4) 1.) b) Nein, wenn es wirklich eine Verklemmung bereits eingetreten ist, kann diese auch erkannt werden, da die betroffenen Prozesse da nicht aus diesem Dead Lock hinaus kommen. * = von selbst.

36

Musterlösung

(a) Ja, es kann passieren. Wegen der Verzögerungen der Nachrichten kann der Wartegruppen im GDD der aktuelle Situation nicht immer entsprechen, d.h. er kann eine Mischung aus neueren und älteren Daten sein. Eine Transaktion, wegen der eine Verklemmung in dem GDD entdeckt wurde, kann schon abgebrochen sein und trotzdem kann der GDD eine andere Transaktion wegen dieser entdeckten (Phantom-)Verklemmung abbrechen.

Bei Verwendung von Zwei-Phasen-Sperren kann dies nicht passieren. Abhilfe kann auch eine verteilte Ermittlung des Graphen sein, z.B. durch Erfassung eines globalen Systemzustandes.

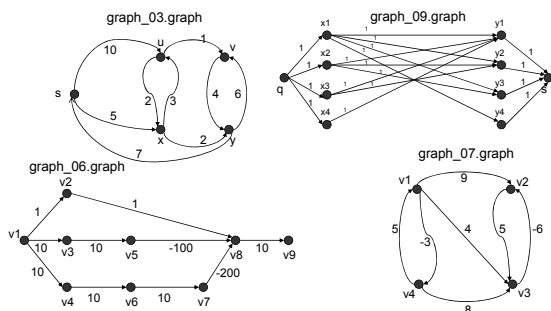
(b) Nein. Eine Verklemmung kann nicht von alleine verschwinden und wird deshalb nach einer bestimmten Zeit entdeckt.

Bei dieser Klausur (90 Minuten Lösungszeit):

für sehr gute Bewertung 3P in 7,1 Minuten Lösungszeit
für ausreichende Bewertung 3P in 14,2 Minuten Lösungszeit

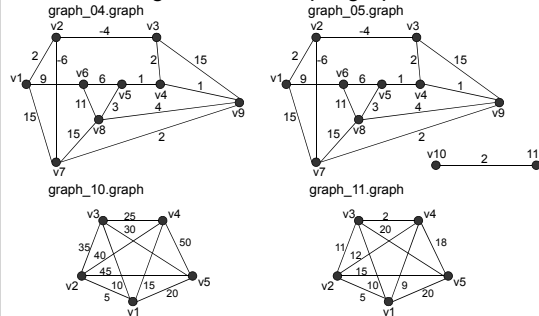
37

Gerichtete Beispielgraphen



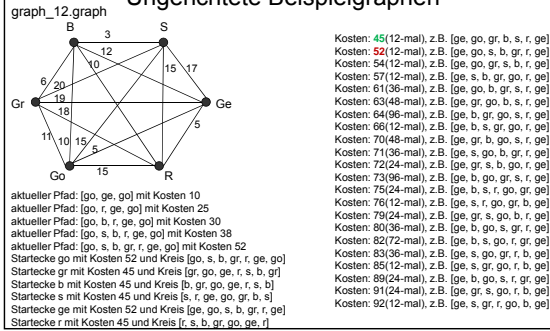
38

Ungerichtete Beispielgraphen



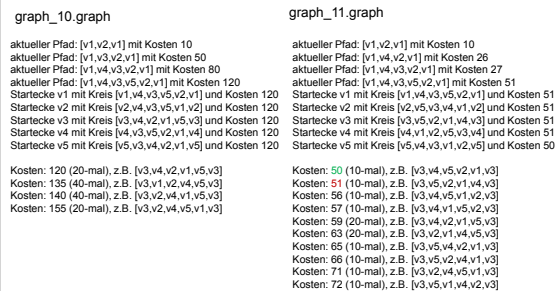
39

Ungerichtete Beispielgraphen



40

Ungerichtete Beispielgraphen



41

