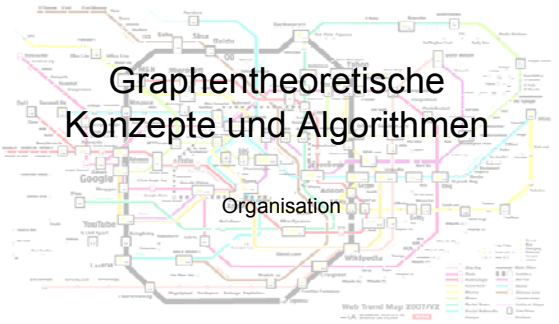


Graphentheoretische Konzepte und Algorithmen

Organisation

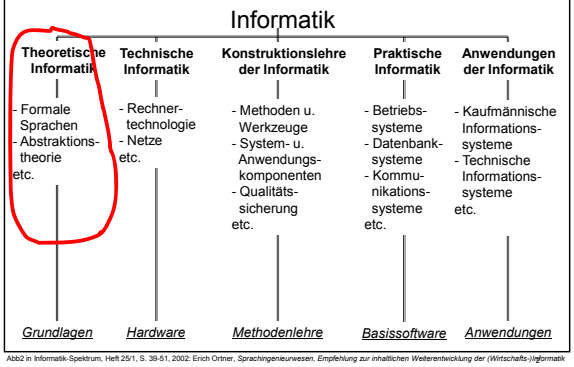


Abb2 in Informatik-Spektrum, Heft 25/1, S. 39-51, 2002. Erich Ortner, *Sprachingenieurwesen, Empfehlung zur inhaltlichen Weiterentwicklung der (Wirtschafts-)Informatik*

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

Informatik — Hochschule für angewandte Wissenschaften Hamburg

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 08^{15}-11^{30}
  - 4 Stunden (2 Viertel) Praktikum (4 Termine)
    - Raum 1165
    - Mittwoch 12^{30}-15^{45} je nach Gruppenzugehörigkeit
- Wie üblich
  - Vorlesung zur Vermittlung der *theoretischen* Grundlagen
  - Praktikum zur praktischen Erfahrung

Letzter Termin: 27.05

6

Informatik — Hochschule für angewandte Wissenschaften Hamburg

Back to the roots

*lateinisch **studere**:*
„(nach etwas) streben,
sich (um etwas) bemühen“)

8

Informatik — Hochschule für angewandte Wissenschaften Hamburg

Vorlesung? offen!

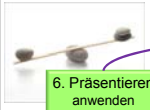
10

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?

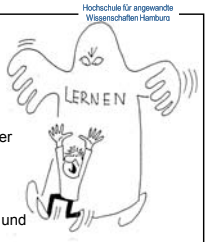


1. Informieren explorieren z.B. Artikel lesen
2. Planen z.B. Arbeitsschritte festlegen
3. Durchführen experimentieren z.B. erste Aufgaben bearbeiten
4. Kontrollieren z.B. testen und besprechen
5. Bewerten akkommodieren z.B. Kriterien prüfen
6. Präsentieren anwenden z.B. im Praktikum

Graphen-theoretische Konzepte und Algorithmen Praktikum

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 * 51?*
Defizitperspektive)
oder
Wie machst du es?
(Offene Aufgabe: *Wie rechnest du 49 * 51?*
Entwicklungsperspektive)

Was will man erfahren, wenn eine Aufgabe gestellt wird?

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 19^{00} Uhr**: Abgabe der geforderten Unterlagen (digital!)
  - Anwesenheitspflicht (**gesamte Praktikumszeit!**)
  - Erfolgreiche Bearbeitung **aller** Aufgaben
  - Einhaltung aller Termine

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

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.

Dokumentationskopf

Details siehe Vorlage

Team: <Teamnummer>, <Namen der Teammitglieder>

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.>

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 angeprochen
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.

Praktikumsaufgaben

- Konstruktoren:
  - new_AlGraph() // 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

Praktikumsaufgaben

- Selektoren:
  - isNil(Graph) // post: prüft, ob der Gragh 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

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

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

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) Flußprobeleme: 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!

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.

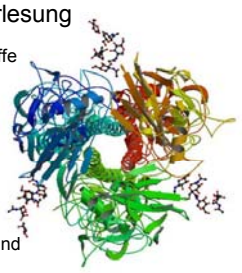
- Beispiel:

```
#gerichtet
Hamburg,Bremen,123
Hamburg,Berlin,289
```

- Beispiel:

```
#ungerichtet
v1,v2,123,456
v3,v4,289,888
```

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

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, Handlungsreisendenprobelm
27.05.	VL11 P03	Petri-Netze: Grundbegriffe, B/E-,S/T- P/E-Systeme

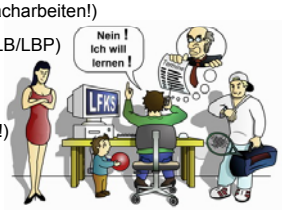
Anforderungen

- 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!)



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

Zum Kauf empfohlen!

- 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*, Oldenbourg , bei Books.Google
- L. Volkmann: *Graphen an allen Ecken und Kanten*, RWTH Aachen , als *.pdf
- K. Reich: *Konstruktivistische Didaktik*, Beltz, bei Books.Google

Aufgabenstellung

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

(a)

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

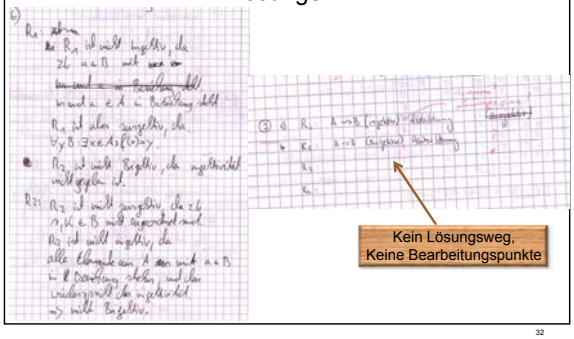
$R_3 := \{(m,a),(a,r),(i,k),(a,a)\}$ $R_4 := \{(m,r),(a,k),(f,a),(f,k)\}$

(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**

Lösungen



Kein Lösungsweg,
Keine Bearbeitungspunkte

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**

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 ?

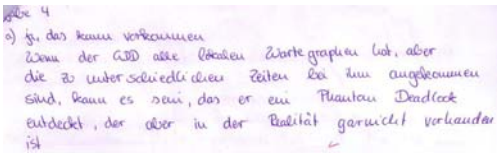
Begründen Sie jeweils Ihre Antwort.

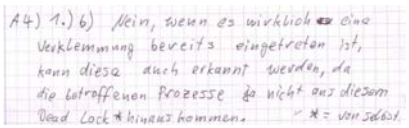
Bei dieser Klausur (90 Minuten Lösungszeit) und vorher unbekannter Aufgabe:
für sehr gute Bewertung *minimal* 3P in **3,8 Minuten Lösungszeit**
für ausreichende Bewertung *minimal* 3P in **7,6 Minuten Lösungszeit**
für sehr gute Bewertung *statistisch* 3P in **5 Minuten Lösungszeit**
für ausreichende Bewertung *statistisch* 3P in **10 Minuten Lösungszeit**

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.

Lösungen





Musterlösung

(a) Ja, es kann passieren. Wegen der Verzögerungen der Nachrichten kann der Wartegraph 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.

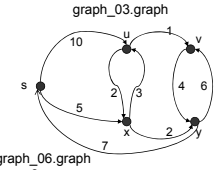
(b) Nein. Eine Verklemmung kann nicht von alleine verschwinden und wird deshalb nach einer bestimmten Zeit entdeckt.

Bei dieser Klausur (90 Minuten Lösungszeit):

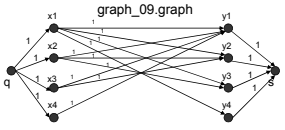
für sehr gute Bewertung	3P in **7,1 Minuten Lösungszeit**
für ausreichende Bewertung	3P in **14,2 Minuten Lösungszeit**

Gerichtete Beispielgraphen

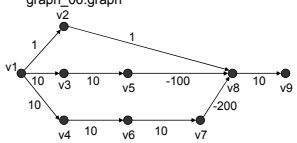
graph_03.graph



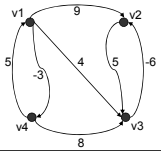
graph_09.graph



graph_06.graph

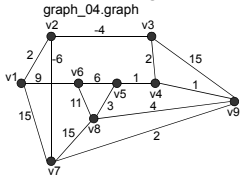


graph_07.graph

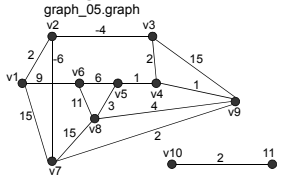


Ungerichtete Beispielgraphen

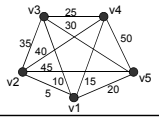
graph_04.graph



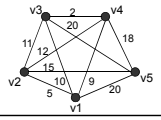
graph_05.graph



graph_10.graph

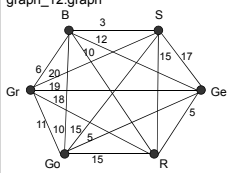


graph_11.graph



Ungerichtete Beispielgraphen

graph_12.graph



aktueller Pfad: [go, ge, go] mit Kosten 10
aktueller Pfad: [go, r, ge, go] mit Kosten 25
aktueller Pfad: [go, b, r, ge, go] mit Kosten 30
aktueller Pfad: [go, s, b, r, ge, go] mit Kosten 38
aktueller Pfad: [go, s, b, gr, r, ge, go] mit Kosten 52
Startecke go mit Kosten 52 und Kreis [go, s, b, gr, r, ge, go]
Startecke gr mit Kosten 45 und Kreis [gr, go, ge, r, s, b, gr]
Startecke b mit Kosten 45 und Kreis [b, gr, go, ge, r, s, b]
Startecke s mit Kosten 45 und Kreis [s, r, ge, go, gr, b, s]
Startecke ge mit Kosten 52 und Kreis [ge, go, s, b, gr, r, ge]
Startecke r mit Kosten 45 und Kreis [r, s, b, gr, go, ge, r]

Kosten: 45(12-mal), z.B. [ge, go, gr, b, s, r, ge]
Kosten: 52(12-mal), z.B. [ge, go, s, b, gr, r, ge]
Kosten: 54(12-mal), z.B. [ge, go, gr, s, b, r, ge]
Kosten: 57(12-mal), z.B. [ge, s, b, gr, go, r, ge]
Kosten: 61(36-mal), z.B. [ge, go, b, gr, s, r, ge]
Kosten: 63(48-mal), z.B. [ge, gr, go, b, s, r, ge]
Kosten: 64(96-mal), z.B. [ge, b, gr, go, s, r, ge]
Kosten: 66(12-mal), z.B. [ge, b, s, gr, go, r, ge]
Kosten: 70(48-mal), z.B. [ge, gr, b, go, s, r, ge]
Kosten: 71(36-mal), z.B. [ge, s, go, b, gr, r, ge]
Kosten: 72(24-mal), z.B. [ge, gr, s, b, go, r, ge]
Kosten: 73(96-mal), z.B. [ge, b, go, gr, s, r, ge]
Kosten: 75(24-mal), z.B. [ge, b, s, r, go, gr, ge]
Kosten: 76(12-mal), z.B. [ge, s, r, go, gr, b, ge]
Kosten: 79(24-mal), z.B. [ge, gr, s, go, b, r, ge]
Kosten: 80(36-mal), z.B. [ge, b, go, s, gr, r, ge]
Kosten: 82(72-mal), z.B. [ge, b, s, go, r, gr, ge]
Kosten: 83(36-mal), z.B. [ge, s, go, gr, r, b, ge]
Kosten: 85(12-mal), z.B. [ge, s, gr, go, r, b, ge]
Kosten: 89(24-mal), z.B. [ge, b, go, s, r, gr, ge]
Kosten: 91(24-mal), z.B. [ge, gr, s, go, r, b, ge]
Kosten: 92(12-mal), z.B. [ge, s, gr, r, go, b, ge]

Ungerichtete Beispielgraphen

graph_10.graph

aktueller Pfad: [v1,v2,v1] mit Kosten 10
aktueller Pfad: [v1,v3,v2,v1] mit Kosten 50
aktueller Pfad: [v1,v4,v3,v2,v1] mit Kosten 80
aktueller Pfad: [v1,v4,v3,v5,v2,v1] mit Kosten 120
Startecke v1 mit Kreis [v1,v4,v3,v5,v2,v1] und Kosten 120
Startecke v2 mit Kreis [v2,v4,v3,v5,v1,v2] und Kosten 120
Startecke v3 mit Kreis [v3,v4,v2,v1,v5,v3] und Kosten 120
Startecke v4 mit Kreis [v4,v3,v5,v2,v1,v4] und Kosten 120
Startecke v5 mit Kreis [v5,v3,v4,v2,v1,v5] und Kosten 120

Kosten: 120 (20-mal), z.B. [v3,v4,v2,v1,v5,v3]
Kosten: 135 (40-mal), z.B. [v3,v1,v2,v4,v5,v3]
Kosten: 140 (40-mal), z.B. [v3,v2,v4,v1,v5,v3]
Kosten: 155 (20-mal), z.B. [v3,v2,v4,v5,v1,v3]

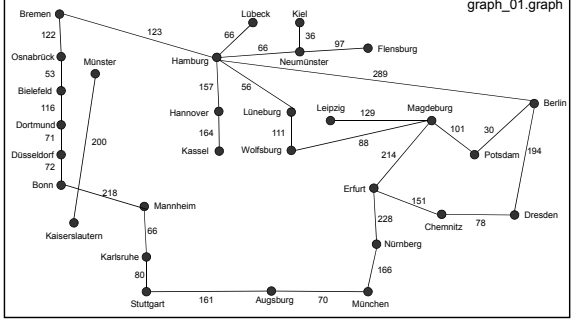
graph_11.graph

aktueller Pfad: [v1,v2,v1] mit Kosten 10
aktueller Pfad: [v1,v4,v2,v1] mit Kosten 26
aktueller Pfad: [v1,v4,v3,v2,v1] mit Kosten 27
aktueller Pfad: [v1,v4,v3,v5,v2,v1] mit Kosten 51
Startecke v1 mit Kreis [v1,v4,v3,v5,v2,v1] und Kosten 51
Startecke v2 mit Kreis [v2,v5,v3,v4,v1,v2] und Kosten 51
Startecke v3 mit Kreis [v3,v5,v2,v1,v4,v3] und Kosten 51
Startecke v4 mit Kreis [v4,v1,v2,v5,v3,v4] und Kosten 51
Startecke v5 mit Kreis [v5,v4,v3,v1,v2,v5] und Kosten 50

Kosten: 50 (10-mal), z.B. [v3,v4,v5,v2,v1,v3]
Kosten: 51 (10-mal), z.B. [v3,v5,v2,v1,v4,v3]
Kosten: 56 (10-mal), z.B. [v3,v4,v5,v1,v2,v3]
Kosten: 57 (10-mal), z.B. [v3,v4,v1,v5,v2,v3]
Kosten: 59 (20-mal), z.B. [v3,v4,v2,v1,v5,v3]
Kosten: 63 (20-mal), z.B. [v3,v2,v1,v4,v5,v3]
Kosten: 65 (10-mal), z.B. [v3,v5,v4,v2,v1,v3]
Kosten: 66 (10-mal), z.B. [v3,v5,v2,v4,v1,v3]
Kosten: 71 (10-mal), z.B. [v3,v2,v4,v5,v1,v3]
Kosten: 72 (10-mal), z.B. [v3,v5,v1,v4,v2,v3]

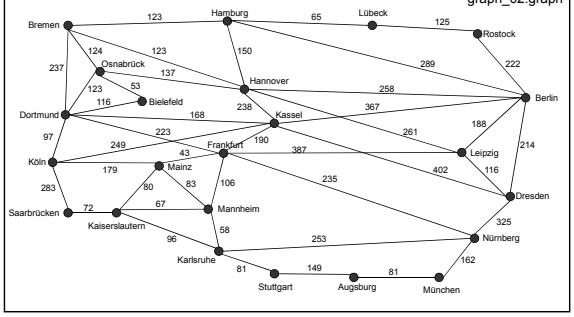
Ungerichtete Beispielgraphen

graph_01.graph



Ungerichtete Beispielgraphen

graph_02.graph



Gerichtete Beispielgraphen

graph_08.graph