

Informatik II - Stoffzusammenfassung für mündliche Prüfung bei Prof. Calmet

0 Inhalt

1 Einleitung

2 Prädikatenlogik

- 2.1 Allgemein
- 2.2 Bestandteile der PL
- 2.3 Variablen
- 2.4 Struktur
 - 2.4.1 Universum
 - 2.4.2 Interpretation
- 2.5 Atomare Formeln und Formeln
 - 2.5.1 Atomare Formeln
 - 2.5.2 Formeln
- 2.6 Passend, gültig, erfüllbar etc.
 - 2.5.1 Passend
 - 2.5.2 Wahr
 - 2.5.3 Gültig
 - 2.5.4 Erfüllbar
 - 2.5.5 Äquivalent
 - 2.5.6 Erfüllbarkeitsäquivalent
 - 2.5.7 Kongruent
- 2.7 Unifikation
 - 2.7.1 Allgemeinster Unifikator
- 2.8 Normalformen
 - 2.8.1 Bereinigte Pränexform
 - 2.8.1.2 Gebundene Umbenennung
 - 2.8.2 Skolemform
- 2.9 Resolution

3 O-Kalkül

4 Datentypen

- 4.1 Abstrakte Datentypen
 - 4.1.1 Keller (Stack)
 - 4.1.2 Schlange (Queue)
- 4.2 Hashing
 - 4.2.1 Prinzip
 - 4.2.2 Was tun bei Kollision?
 - 4.2.2.1 Verketteten
 - 4.2.2.2 Sondieren
 - 4.2.2.2.1 Linear
 - 4.2.2.2.2 Quadratisch
 - 4.2.2.3 Kollision 2. Ordnung

5 Graphen

- 5.1 Allgemein
- 5.2 Halde (Heap)
 - 5.2.1 Allgemein
 - 5.2.2 Unterschied zum Binärbaum
- 5.3 Suchen im Graphen
 - 5.3.1 Tiefensuche
 - 5.3.2 Breitensuche
 - 5.3.3 Kruskal
 - 5.3.4 Prim
 - 5.3.5 Dijkstra

6 Algorithmen

- 6.1 Mögliche Eigenschaften
 - 6.1.1 Terminierend
 - 6.1.2 Deterministisch
 - 6.1.3 Determinierend
- 6.2 Teile und Herrsche
- 6.3 Greedy
- 6.4 Dynamische Programmierung
- 6.5 Probabilistische Algorithmen
 - 6.5.1 Monte Carlo Algorithmen
 - 6.5.2 Las Vegas Algorithmen
 - 6.5.3 Macao Algorithmen
- 6.6 Sortieralgorithmen
 - 6.6.1 SelectionSort
 - 6.6.2 InsertionSort
 - 6.6.3 MergeSort
 - 6.6.4 QuickSort
 - 6.6.5 HeapSort
- 6.7 Suchalgorithmen
 - 6.7.1 Knut-Morris-Pratt
 - 6.7.2 Boyer-Moore
- 6.8 Diverse Algorithmen
 - 6.8.1 Euklidischer Algorithmus
 - 6.8.2 Brute Force

7 Objektorientierte Programmierung

- 7.1 Allgemeines
- 7.2 Sinn der OOP
- 7.3 "Die 5 Schlüsselbegriffe"
 - 7.3.1 Klasse
 - 7.3.2 Objekt
 - 7.3.3 Polymorphie
 - 7.3.4 Kapselung
 - 7.3.5 Vererbung
- 7.4 Abstrakte Klasse
- 7.5 UML

8 Fragen

- 8.1 Prädikatenlogik
- 8.2 Algorithmen
- 8.3 Wissensfragen
- 8.4 OOP

9 Impressum

1 Einleitung

Diese Zusammenfassung hat KEINEN Anspruch auf Vollständigkeit. Ebenso wenig will diese Zusammenfassung detaillierte Erklärungen liefern.

Sie dient lediglich dazu, um sich einen Überblick darüber zu verschaffen, was in den Mündlichen Prüfungen bei Prof. Calmet im Jahre 2005 so drangekommen ist.

Diese Zusammenfassung darf frei verarbeitet und verwendet werden.

Wenn jemand einen Fehler entdeckt, egal welcher Natur, dann möge er/sie ihn mir bitte melden (uni@cokra.de). Das Gleiche gilt für Ergänzungen.

Cosmo, im November 2005

2 Prädikatenlogik

2.1 Allgemein

Mit PL lassen sich Aussagen über Objekte machen. Mit Aussagenlogik nicht.

2.2 Bestandteile der PL

Besteht aus Termen. Terme können sein:

- Variablen (z. B. x, y, z)
- Funktionssymbole (z.B. $f(x)$)
- Konstanten (Funktionssymbole der Stelligkeit 0)

Dazu kommen:

- Prädikatensymbole (z. B. $P(x) Q(y)$)
- Quantoren (forall und exists)
- logische Symbole (und oder nicht)

2.3 Variablen

Können gebunden oder frei vorkommen.

Gebunden werden sie durch die Quantoren

2.4 Struktur

Eine Struktur setzt sich zusammen aus Interpretation und Universum.

2.4.1 Universum

Enthält die Grundmenge (zum Beispiel „Menschen“)

2.4.2 Interpretation

Weist Prädikatensymbolen ein Prädikat

Funktionssymbolen ein Symbol

und freien Variablen einen Wert aus dem Universum zu

2.5 Atomare Formeln und Formeln

2.5.1 Atomare Formeln

Sind alleinstehende Prädikatensymbole und Funktionssymbole

2.5.2 Formeln

Durch Junktoren verbundenen atomare Formeln

2.6 Passend, gültig, erfüllbar etc.

2.5.1 Passend

Eine Struktur A heißt zu einer Formel F passend, wenn in der Interpretation jedem Prädikatensymbol ein Prädikat, jedem Funktionssymbol ein Symbol, jeder freien Variablen einen Wert aus U zugeordnet wird

2.5.2 Wahr

Eine passende Struktur A ist wahr, wenn die Formel unter ihr erfüllbar ist: $A(F)=1$

Man spricht dann auch von einem Modell für F

2.5.3 Gültig

Wenn für eine Formel F alle PASSENDEN Strukturen wahr sind, dann ist F gültig

2.5.4 Erfüllbar

Ist eine Formel F für mindestens EINE Struktur wahr, so ist die Formel erfüllbar

2.5.5 Äquivalent

Gibt es eine Struktur, die sowohl für F als auch für G wahr ist, so sind F und G äquivalent

2.5.6 Erfüllbarkeitsäquivalent

Gibt es für F eine Struktur die wahr ist und für G eine Struktur die wahr ist, dann sind F und G erfüllbarkeitsäquivalent. (Beachte den Unterschied zur Äquivalenz: Hier spreche ich nicht nur von EINER Struktur, die für BEIDE Formeln passt)

2.5.7 Kongruent

Haben zwei Formeln F und G keine Modelle, so sind sie kongruent.

2.7 Unifikation

Bei Unifikation versucht man durch Substitution (sprich: ersetzen von Variablen durch Terme (da Variablen auch Terme sind, ist es also erlaubt, Variablen durch Variablen zu ersetzen)) aus einer n-elementigen Literalmenge eine ein-elementige Literalmenge zu machen. Wenn man dies schafft, dann ist die Formel F unifizierbar.

Beispiel:

$$\begin{aligned} &\{P(x), P(f(y))\} \\ &[x/f(y)] \\ \Rightarrow &\{P(f(y))\} \end{aligned}$$

2.7.1 Allgemeinsten Unifikator

Das Beispiel aus 2.7 ist der allgemeinste Unifikator. Man könnte in dem Beispiel auch wie folgt substituieren:

$$\begin{aligned} &\{P(x), P(f(y))\} \\ &[x/f(a)] \\ \Rightarrow &\{P(f(a)), P(f(y))\} \\ &[y/a] \\ \Rightarrow &\{P(f(a))\} \end{aligned}$$

Dies ist nicht der allgemeinste Unifikator, da mehr als nötig unifiziert wurde und man somit nicht mehr zurück kann.

2.8 Normalformen

2.8.1 Bereinigte Pränexform

Bereinigen: Keine Variable darf sowohl gebunden als auch ungebunden vorkommen

Pränex: Alle Quantoren stehen ganz vorne.

2.8.1.2 Gebundene Umbenennung

Benennt man beim bereinigen die gebundene Variablen um, so ist die BPF äquivalent zur Ausgangsformel

2.8.2 Skolemform

BPF liegt vor, sonst erst in BPF überführen, dann Existenzquantoren eliminieren.

2.9 Resolution

Mit Resolution weist man die Unerfüllbarkeit einer Formel nach. Schafft man dies nicht, so sagt dies NICHTS über die Formel aus!

Um Resolution anwenden zu können, muss die Formel in Konjunktiver Normalform (KNF) vorliegen.

Die UNDs trennen die Klauseln, die ODERs trennen die Prädikate in der Klausel durch “,“

Beispiel: $F = P(x) \text{ ODER } \neg Q(y) \text{ UND } \neg P(x) \text{ ODER } Q(y)$

Dies ergibt folgende Klausel: 1 $\{P(x), \neg Q(y)\}$ 2 $\{\neg P(x), Q(y)\}$

Klausel 1 verbunden mit Klausel 2 ergibt $\{P(x), \neg P(x), \neg Q(y), Q(y)\}$ also die leere Klausel $\{\}$

Somit ist gezeigt, dass F unerfüllbar ist und $\neg F$ ist eine Tautologie (“immer wahr“)

3 O-Kalkül

Keine Zusammenfassung vorhanden. Am besten im Cormen nachlesen.

Dies wurde bisher gefragt:

- obere untere Schranke aufzeichnen+erklären

- Warum f element $O(g)$? Grafik aufgezeichnet ($c \cdot g(n)$, $f(n)$, n_0)

- Warum ist $3n$ in $O(n)$? Weil man ja c zb 4 wählen kann
- O-Kalkül (Obereschränke erklären... $n0$, c und bild gezeichnet)
- Teta Kalkuell, Beispiel

4 Datentypen

4.1 Abstrakte Datentypen

4.1.1 Keller (Stack)

First In Last Out

Methoden:

push	einfügen eines Elementes
pop	entfernen des obersten Elementes
top	liefert oberstes Element
length	liefert die Länge des Stacks

4.1.2 Schlange (Queue)

First In First Out

Methoden:

insert	einfügen eines Elementes
tail	entfernen des letzten Elementes
head	liefert letztes Element
length	liefert Länge der Schlange

4.2 Hashing

4.2.1 Prinzip

Um schneller auf die Daten zugreifen zu können wird ihnen jeweils ein Index zugeordnet. Damit man den Speicher gering halten kann, begrenzt man den Speicherplatz durch eine Hashfunktion:

Bsp: $h(x) = x \bmod 7$

Nun habe ich 8 Plätze (0-7)

4.2.2 Was tun bei Kollision?

4.2.2.1 Verketteten

An jedem Index ist eine List, so können dort mehrer Elemente abgelegt werden

4.2.2.2 Sondieren

Element wird verschoben

4.2.2.2.1 Linear

+ j

4.2.2.2.2 Quadratisch

+ j^2

4.2.2.3 Kollision 2. Ordnung

Soll eine Element an eine Stelle, wo bereits ein KOLLIDIERTES Element ist, so handelt es sich um eine Kollision zweiter Ordnung.

5 Graphen

5.1 Allgemein

Graphen können gerichtet oder ungerichtet sein.

Sie lassen sich durch eine Adjazensliste oder eine Adjazensmatrix darstellen (und im Rechner speichern)

5.2 Halde (Heap)

5.2.1 Allgemein

Heap ist ein Baum, in dem das größte (maxHeap) oder das kleinste (minHeap) Element in der Wurzel steht.

Die Knoten sind von oben, von links nach rechts durchnummeriert.

Daher lässt sich ein Heap sehr gut als Liste darstellen.

Man benutzt Heaps, wenn man schnell auf das Element höchster Priorität zugreifen möchte.

5.2.2 Unterschied zum Binärbaum

In einem Binärbaum stehen links vom Knoten nur kleinere, rechts vom Knoten nur größere Elemente. Dies kann im Heap nicht so sein, da das kleinste oder das größte Element in der Wurzel steht.

5.3 Suchen im Graphen

5.3.1 Tiefensuche

Suche zunächst im Pfad des linken Kindes des Startknotens
Lässt sich mit STACK realisieren.

5.3.2 Breitensuche

Suche in beiden Kindern des Startknotens
Lässt sich mit QUEUE realisieren

5.3.3 Kruskal

Sucht den minimal spannenden Baum in einem ungerichteten Graphen
dazu fügt er immer die günstigste Kante hinzu, sofern diese keinen Zyklus erzeugt

5.3.4 Prim

Wie Kruskal, nur dass Prim beim Knoten ganz links anfängt und immer die günstigste Kante zu einem Knoten hinzufügt, der noch nicht im Baum ist.

5.3.5 Dijkstra

Sucht den günstigsten Weg vom Startknoten zum gesuchten Knoten und bildet dabei den minimal spannenden Baum in einem GERICHTETEN Graphen.

6 Algorithmen

6.1 Mögliche Eigenschaften

6.1.1 Terminierend

Der Algorithmus endet nach einer endlichen Anzahl an Schritten

6.1.2 Deterministisch

Der nachfolgende Schritt ist immer eindeutig festgelegt

6.1.3 Determinierend

Das Ergebnis des Algorithmus ist immer eindeutig

6.2 Teile und Herrsche

Aufteilen von Problem in Teilprobleme. Dann Teilprobleme lösen

Beispiel: MergeSort

6.3 Greedy

In jedem Schritt die beste Möglichkeit wählen

Beispiele: Kruskal, Prim, Dijkstra

6.4 Dynamische Programmierung

Wie Teile und Herrsche werden Teilprobleme gelöst, diese dann aber in Tabelle gespeichert und bei Wiederauftreten des gleichen Teilproblems, wird dieses nicht noch einmal gelöst, sondern die Lösung aus der Tabelle genommen.

Beispiele: Trellis, Viterbi

6.5 Probabilistische Algorithmen

Sie sind Zufallsgesteuert und lassen sich in folgende Arten unterteilen:

6.5.1 Monte Carlo Algorithmen

Sie terminieren immer, liefern allerdings nicht zwingend ein richtiges Ergebnis

Beispiel: Miller-Rabin (Primzahltest, der eine Zahl n immer wieder durch eine Zahl a zu teilen versucht, bis er Erfolg hat und dann sagt: "Ist keine Primzahl". Wenn ihm dies nach endlich vielen Schritten nicht gelingt, dann sagte er, dass n eine Primzahl ist, was aber nicht bewiesen ist.

6.5.2 Las Vegas Algorithmen

Sie terminieren nicht zwingend, aber wenn sie terminieren, dann liefern sie auf jeden Fall ein korrektes Ergebnis.

Beispiel:

6.5.3 Macao Algorithmen

Sie terminieren immer und liefern ein richtiges Ergebnis. Dieses kann aber mitunter sehr, sehr lange dauern. Daher kann man aus einem Macao schnell einen Las Vegas Algorithmus machen, in dem man ihn einfach nach gewisser Zeit abbricht. Dann hat er entweder ein richtiges Ergebnis, oder er hat kein Ergebnis.

Beispiel: RandomQuickSort (Pivotelement wird zufällig bestimmt)

6.6 Sortieralgorithmen

6.6.1 SelectionSort

Sortieren durch Auswählen.

Liste wird nach dem kleinsten Element durchsucht und dieses wird dann mit dem ersten Element vertauscht. Danach wird das kleinste Element der Restliste gesucht und mit dem Element an zweiter Stelle vertauscht, und so weiter.

Der Aufwand ist dabei logischerweise recht hoch. Da jedes Element mit jedem verglichen wird, ist es $n \cdot n$ also n^2 . Sowohl im Best als auch im Worst Case

6.6.2 InsertionSort

Sortieren durch Einfügen

Jedes Element mit seinem Vorgänger verglichen. Ist es kleiner, wird es mit ihm vertauscht. So "wandert" jedes Element durch die Liste zu seinem Platz.

Im Best Case hat man hier eine sortierte Liste vorliegen, dann wird nichts verschoben und jedes Element wird einmal verglichen. Also $O(n)$

Im WorstCase ist die Liste falschrum sortiert, dann muss jedes Element verschoben werden. Der Aufwand ist dann n^2

6.6.3 MergeSort

Sortieren durch Mischen

Die Liste wird geteilt und die beiden entstandenen Listen werden ebenfalls wieder geteilt und so weiter, so lange, bis nur noch ein Elementige Teillisten vorliegen. Diese werden nun Zusammengefügt und dabei sortiert, indem immer die ersten Elemente (also die kleinsten) der Teillisten miteinander verglichen werden.

Da die Listen immer weiter aufgesplittet werden, ergibt sich quasi ein Baum. Somit hat MergeSort im Best und im WorstCase $O(n \log n)$

6.6.4 QuickSort

Zuerst wird ein Pivot Element gewählt (beim probabilistischen RandomQuickSort geschieht dies zufällig)

Idealer Weise ist das Pivot Element wertmäßig in der Mitte der Liste.

Nun werden alle Elemente die kleiner sind als das Pivot Element in einer Liste und alle Elemente die größer sind in einer anderen Liste gespeichert. Danach kommt das Pivot Element zwischen beide Listen und der Algorithmus macht rekursiv mit den beiden neuen Teillisten weiter.

Im Worst Case ist das Pivot Element immer das kleinste oder das größte Element der Liste. So wird immer nur ein Element von der Liste abgespalten. Jedesmal müssen n Vergleiche gemacht werden, für n Elemente. Also $O(n^2)$

Der Best Case ist, wie oben beschrieben, wenn man immer den Median wählt. So erstellt sich durch das aufsplitten der Listen ein Baum $\Rightarrow O(n \log n)$

6.6.5 HeapSort

Man hat einen Heap. Dieser lässt sich gut als Liste darstellen.

Man vergleicht nun das Element an der Stelle i mit seinen beiden Nachfolgern, die an den Stellen $2i$ und $2i+1$ sitzen. Das kleinste dieser drei Elemente kommt an Stelle i . Dies macht man mit jedem Element. Das ganze nennt sich MinHeapify. Danach ist die Heapeigenschaft hergestellt: Jeder Knoten ist größer als seine beiden Nachfolger. Auf diese Art weis man, dass das kleinste Element in der Wurzel ist. Dieses wird nun entfernt und in einer Liste gespeichert. Danach wendet man wieder MinHeapify an um die Heap eigenschaften wieder herzustellen. Und so weiter. Im Best Case hat man bereits eine sortierte Heapliste, so dass das man nur nach und nach die Wurzeln in die Liste speichern muss und fertig. Aufwand: $O(n)$
Im Worst Case hat man eine falschrumsortierte Heapliste. So mit hat man dann den Aufwand $O(n \log n)$

6.7 Suchalgorithmen

6.7.1 Knut-Morris-Pratt

Sucht in einem Text/String nach einem Wort/Teilwort.

6.7.2 Boyer-Moore

Wie KMP, allerdings wird hier geprüft, ob der Buchstabe überhaupt im Wort vorkommt und wenn nicht, wird dieses direkt um die gesamte Wortlänge verschoben.

6.8 Diverse Algorithmen

6.8.1 Euklidischer Algorithmus

Algorithmus zur Berechnung des ggT

Interessant dabei ist der Rekursive Aufruf:

$ggT(a,b) = ggT(b, a \bmod b)$

6.8.2 Brute Force

zu Deutsch: Brutaler Zwang

Algorithmus geht jede Möglichkeit durch (zum Beispiel bei Passwörtern)

Dies ist logischerweise sehr uneffektiv

7 Objektorientierte Programmierung

7.1 Allgemeines

Bei der OOP wird in Klassen programmiert, auf die aus dem Hauptprogramm/Hauptklasse (main) zugegriffen wird

7.2 Sinn der OOP

Dies hat den Vorteil, dass der Code einfach verändert werden kann.

Zudem kann er wieder verwendet werden

7.3 "Die 5 Schlüsselbegriffe"

7.3.1 Klasse

Die Klasse ist quasi eine Schablone für die Objekte. In ihnen werden die Methoden und Variablen deklariert

7.3.2 Objekt

Sind die Instanzen der Klassen.

7.3.3 Polymorphie

Zu Deutsch: Vielgestaltigkeit.

Sie besagt, dass eine Variable/eine Methode in verschiedenen Objekten für verschiedenen Zwecke verwendet werden kann.

Wenn zum Beispiel die Unterklasse die Methode "hallo(int x)" erbt, dann kann man sie in der Unterklasse so überschreiben, dass man sie danach mit einem Char füttern kann. (also "hallo(char x)")

7.3.4 Kapselung

oder auch: Geheimnisprinzip.

Die Variablen in den Objekten werden als Private deklariert. Von außen kann man nicht auf sie zugreifen. Nur Methoden aus derselben Klasse können darauf zugreifen.

Von außen greift man also über get und set Methoden auf die privaten Variablen zu

7.3.5 Vererbung

Eine Oberklasse kann ihre Eigenschaften (Methoden und Variablen) an die Unterklasse vererben.

Beispiel:

Man hat Anfang der 90er eine Adresskartei programmiert. Nun will man diese erweitern, damit man auch eMail Adressen darin speichern kann.

Man hat also eine Klasse "Adressen"

Nun erstellt man eine Klasse "Adressen2005". Diese lässt man alles aus "Adressen" erben und ergänzt das ganze um "eMail"

7.4 Abstrakte Klasse

Eine abstrakte Klasse kann keine Objekte erzeugen. Sie kann lediglich Methoden und Variablen in sich haben, die sie an andere Klassen vererbt.

Beispiel:

Man hat eine Klasse "Fahrzeuge". In ihr ist definiert, was ein Fahrzeug alles haben muss. Also Räder, Motor, etc.

Nun hat man eine Klasse "Auto". Sie erbt die Fahrzeugeigenschaften und füllt sie mit Werten. Also 4 Räder, ein Motor, etc. Daraus lassen sich dann Objekte instanzieren wie zum Beispiel "Auto1"

7.5 UML

Steht für Unified Modelling Language und ist von der OMG (Object Management Group)

Mit UML lassen sich Abläufe eines Programms gut darstellen.

8 Fragen

Folgende Fragen sollte jeder beantworten können, der das oben Beschriebene verstanden hat.

Die Fragen wurden so, oder so ähnlich in den mündlichen Informatik II Prüfungen im Jahr 2005 gefragt.

8.1 Prädikatenlogik

1. Was ist Prädikatenlogik?
2. Erklären sie die Syntax der Prädikatenlogik
3. Was ist Unifikation?
4. Was ist der allgemeinste Unifikator?
5. Was ist ein Prädikat?
6. Was ist ein Term?
7. Was ist eine Substitution?
8. Was bedeutet $P(x)$ in der Prädikatenlogik?
9. Was ist "passend" in der Prädikatenlogik?
10. Was bedeutet $f(x)$?
11. Schreiben Sie eine Prädikatenlogische Formel auf
12. Wie können Variablen auftauchen?
13. Was ist die Disjunktive Normalform?
14. Was ist die Skolemnormalform?

8.2 Algorithmen

1. Erklären:
 - * Mergesort (Teile und Herrsche)
 - * Kruskal (Graphenalgorithmus)
 - * Brute-Force
 - * Primzahltest (Miller-Rabin)
 - * Algorithmus für ggT -> Euklid
 - * Knuth-Morris-Pratt (Zeichensuche)
 - * Heapsort
 - * Beispiel für Macao, Monte-Carlo, Las Vegas
2. Machen sie eine Aufwandanalyse bei InsertionSort
3. Erklären sie das O-Kalkül und machen sie ein Beispiel

8.3 Wissensfragen

1. Was sind Unterschiede von Baum und Graphen?
2. Was ist ein Zyklus?
3. Was sind Graphenalgorithmen?
4. Welche Arten von Probabilistischen Algorithmen gibt es?
5. Was bedeutet deterministisch?
6. Was ist ein Greedy-Algorithmus?
7. Was ist eine Datenstruktur?
8. Was ist ein Heap?
9. Wie speichert man einen Heap?
10. Wie ist die Vater-Sohn-Anordnung in einem Heap?
11. Überführen sie eine Heap-Liste in einen Heap-Baum
12. Was ist der Unterschied zwischen einem Heap-Baum und einem Binärbaum?
13. Welche Graphenalgorithmen gibt es?
14. Wie kann man einen Graphen speichern?
15. Wie funktioniert Tiefensuche?
16. Erklären sie einen Suchbaum
17. Wie fügt man eine Zahl hinzu?
18. Was sind ADT (Abstrakte Datentypen)? Machen sie ein Beispiel
19. Wie geht Hashen? Machen sie ein Beispiel
20. Welche Kollisionen kann es beim Hashen geben?
21. Was ist Teile-und-Herrsche? Geben sie ein Beispiel
22. Was ist dynamisches Programmieren?
23. Erklären sie die Rekurrenz für Fibonaccizahlen
24. Was ist ein Keller? Welche Operationen gibt es und was machen sie? Beispiel für "pop"

8.4 OOP

1. Was ist OOP?
2. Was bedeutet Polymorphie in der OOP?
3. Was ist ein Attribut?
4. Was bedeutet Vererbung?
5. Gehört ein Objekt immer einer Klasse an?
6. Kennt ein Objekt seine Klasse und auch umgekehrt?
7. Nennen sie die 5 Schlüsselwörter in der OOP
8. Was ist Kapselung? Machen sie ein Beispiel
9. Für was steht UML und was ist die Bedeutung? Machen sie ein Beispiel

9 Impressum

Diese Zusammenfassung wurde von mir, Cosmo, Student der Informationswirtschaft seit WS04/05, im Rahmen meiner Vorbereitung auf die Prüfung geschrieben.

Für hilfreiche Tipps und Unterstützung gebührt besonderer Dank folgenden Personen:

Daniel, Ina, Alex, Hakan, Tanja, Ben, Michi sowie vielen Mitgliedern von www.unika04.de speziell *Stielschen* fürs Zusammenschreiben der Fragen.

Sollte ich jemanden vergessen haben, kriegt er/sie ein Bier und ich werde es hier ändern ;-)