Fragen des I. GDI-Tests 2014

1 Beschreiben Sie möglichst präzise den Unterschied zwischen analog und digital.

Ein digitales Signal ist entweder 1 oder 0 (on/off bzw. true/false), ein analoges Signal kann hingegen beliebig viele Werte dazwischen annehmen.

2 Warum ist analog genauer als digital? Warum ist digital genauer als analog?

Ein digitales Signal ist präziser als ein analoges, weil es nur zwei annehmbare Werte gibt, die besser verwertet werden können. Mögliche Verzerrungen und Ungenauigkeiten, die bei einem analogen Signal auftreten können, werden automatisch eliminiert. Ein analoges Signal ist genauer als ein digitales, weil es Grauzonen gibt, die durch ein digitales Signal nicht gezeigt werden können; Das analoge Signal weist einen stufenlosen und beliebig feinen Verlauf auf, kann unendlich viele Werte annehmen und so theoretisch unendliche „Genauigkeit" erreichen.

3 Diskutieren Sie die Frage, ob Informatik ohne Strom einen Sinn haben kann.

Natürlich - es ist bloß ein weitverbreitetes Missverständnis, dass Informatik von Strom abhängt; Man kann Informatik mit fast allem treiben, was wohlunterschieden (digital) ist - die Methoden der Informatik sind universell anwendbar.

4 Erklären Sie den Begriff „Sampling".

Sampling bezeichnet das Abtasten eines (meist analogen) Signals, wodurch aus einem zeitkontinuierlichen ein zeitdiskretes Signal gewonnen wird. Mithilfe dieser „Abtast-Punkt-Werte" kann man dann wieder die ursprüngliche Welle errechnen.

5 Definieren Sie, was ein Baum ist.

Ein Baum ist ein ungerichteter, zyklenfreier Graph.

6 Was ist ein Algorithmus, und wozu braucht er einen Korrektheitsbeweis?

Er ist eine eindeutige Folge von Anweisungen endlicher Länge(=Finitheit) zur Lösung eines bestimmten Problems. Zu einem Algorithmus gehören PreCondition, PostCondition, Ablaufbeschreibung und Korrektheitsbeweis.

Der Korrektheitsbeweis wird benötigt, um sicherzustellen, dass der Algorithmus der Problemstellung gerecht wird (=Effektivität) und die Spezifikation vollständig und korrekt umsetzt.

Liefert der Algorithmus zu jeder der PreCondition genügenden Eingabe eine der PostCondition genügende Ausgabe *(=partiell korrekt)*, und terminiert er überdies für alle möglichen, erlaubten Eingaben, ist er *total korrekt*, der Korrektheitsbeweis ist erbracht.

7 Beschreiben Sie möglichst kurz und präzise den ggT-Algorithmus von Euklid.

originaler Euklid:

Man nehme zwei Zahlen a und b, und ziehe von der größeren solange die kleinere Zahl ab, bis eine der beiden Zahlen 0 ist. Die andere ist dann der ggT von a und b.

moderner Euklid:

Man nehme zwei Zahlen und dividiere die größere durch die Kleinere. Dann nehme man den Rest der Division und den Teiler (=kleinere Zahl v. vorhin) und starte wiederum. Schlussendlich wird man zu einer Division mit 0 Rest kommen - der Teiler ist dann der ggT.

8 Welchen Vorteil bietet das Hornerschema? Hat es denn auch Nachteile?

Kurzer Exkurs: Das Hornerschema ist ein Umformungsverfahren für Polynome, um die Berechnung von Funktionswerten zu erleichtern. Hierbei wird das Polynom so umgeschrieben, dass keine Potenzen der Funktionswerte vorkommen und dadurch nur noch addiert/multipliziert werden muss, was die Rechenzeit deutlich verkürzt.

$$\text{Beispiel: } 1 + 2x + 3x2 + 4x3 + 5x4 = 1 + x(2 + x(3 + x(4 + 5x)))$$

Das Hornerschema verkürzt die Rechenzeit bei Rechnen mit Polynomen - man formt sie um, sodass man nur noch addieren und multiplizierten, nicht aber auf die Potenzen achten muss. Allerdings kann diese Umformung die Rechnung, wenn sie lange ist, unübersichtlich machen, und man kann sich leicht verzetteln. Außerdem braucht alleine das Aufstellen der Rechnung schon eine Menge Zeit, also zahlt sich der Vorgang nur aus, wenn es sich um ein großes x handelt (nicht etwa bei einem x=1 etc.)

9 Welchen Aufwand hat die Addition zweier Binärzahlen?

Die Addition zweier n-Bit-Zahlen erfordert mithin n Bit-Operationen. Der Aufwand ist daher O(n).

10 Wie sehen negative Binärzahlen aus?

Beim Vorzeichen-Wechsel von Binärzahlen gilt: „alles invertieren und 1 addieren“.
Wenn man nun z.B. die Zahl $0000\,1011_2 = 11_{10}$ hat, invertiert man sie: $1111\,0100_2$
und addiert 1 dazu: $1111\,0101_2$, was -11_{10} ergibt.
Bei negativen Binärzahlen ist das linkste Bit das "Vorzeichen"; 1 ist negativ, 0 ist positiv.

11 Führen Sie die Addition zweier Binärzahlen a,b vor $(16 < a, b < 30)$

$$\begin{array}{l} 00010010_2 = 18_{10} \\ \underline{00010001_2 = 17_{10}} \\ 00100011_2 = 35_{10} \end{array}$$

12 Warum konvergiert die harmonische Reihe, wenn man sie in C programmiert?

Die Glieder der harmonischen Reihe sind Brüche bzw. rationale Zahlen, können also nur als float oder double abgespeichert werden. Beide Datentypen haben allerdings nur eine beschränkte Genauigkeit. Irgendwann werden die Werte so klein, dass der Computer sie bei der Addition abrundet(*=Absorption - die Exponenten der Gesamtsumme und den sehr kleinen Reihengliedern unterscheiden sich so sehr, dass alle signifikanten Bit nach einer Exponentenangleichung "hinter"den 23 Bit der Mantisse der Gesamtsumme dazuaddiert würden, und daher verworfen werden*) und sich die Gesamtsumme nicht mehr weiter erhöht. Daher wächst die Reihe ab einem bestimmten Wert nicht mehr und erweckt den Anschein, sie konvergiere.

13 Erklären Sie den Aufbau einer single-precision float-Zahl an einem guten Beispiel (selber wählen, dezimal und binär anzeigen)

Eine Single-Precision Floating-Point Zahl nach IEEE 754 benötigt 4 Byte = 32 Bit, unterteilt in (vlnr.) 1 Bit Vorzeichen-Bit, 8 Bit Exponent und 23 Bit Mantisse.

Wenn Vorzeichen-Bit = 0, ist die Zahl positiv, wenn VZ-Bit = 1, ist die Zahl negativ.

Der Exponent wird getrennt von Mantisse und VZ-Bit als eigenständige Binärzahl gesehen. Der Zahlenbereich verläuft von -126 bis 127, wobei man das Vorzeichen vermeidet, indem man zum Exponenten einen Bias von 127 dazuaddiert.

Die letzten 23 Bit sind die Mantisse, wobei hier die Binärzahl zuerst auf die Form „1.. . . “ gebracht wird*(=normalisiert)*, und anschließend nur die Zahlen hinter dem Komma abspeichert werden, das „1.“ ist *implizit* gespeichert. *Ausnahme*: Wenn die Mantisse 0 ist, ist die gesamte Zahl 0, daher ändert sich das „1.“ per Konvention zu „0.“.

Beispiel: 13.1875_{10}

$13.1875 > 0 \rightarrow$ VZ-Bit $= 0$

Mantissenteil vor Komma: $13_{10} = 1101_2$
Mantissenteil nach Komma: $0.1875_{10} = 0011_2$
$0.1875 \cdot 2 = 0.375$, Übertrag 0
$0.375 \cdot 2 = 0.75$, Übertrag 0
$0.75 \cdot 2 = 1.5$, Übertrag 1 $0.5 \cdot 2 = 1$, Übertrag $1 \rightarrow 0011$
Mantisse: 1101.0011; anschließend Normalisierung: $1.1010011 \cdot 2^3$
Nachkommateil 1010011 ist 7 Bit lang (23-7= 16), es werden 16 Nullen angehängt.
Mantissenteil der Float-Zahl lautet: 1010011*0000000000000000*

Exponent: 2^3; ausgehend vom Bias 127: $3 + 127 = 130_{10} = 10000010_2$

Ergebnis: 13.1875_{10} ist in IEEE 754 single-precision float:
0100 0001 0**101 0011 0000 0000 0000 0000**
Vorzeichen-Bit Exponent **Mantisse**

14 Wie funktioniert der Grey-Code?

Beim Hinaufzählen von Binärzahlen ändern sich öfters mehr als 1 Bit zur gleichen Zeit, z.B. von 3 auf 4: 0011 auf 0100. Der Grey-Code beschreibt ein eigenes Zähl-System, wobei sich immer nur 1 Bit ändert; von 3 auf 4: 0010 auf 0110. Es handelt sich um eine andere Variante des Binärsystems.

15 Geben Sie eine kurze Erklärung der von-Neumann-Architektur.

Ein Computer besteht laut Von-Neumann-Architektur aus folgenden 3 Komponenten: CPU (aufgeteilt in Steuer- und Rechenwerk), Speicherwerk und Ein-/Ausgabewerk. Alle kommunizieren über ein gemeinsames Bus-System.

16 Welche logischen Operationen kennen Sie?

AND, OR, NOT, NAND, XOR, NOR

17 Was ist das Dualitätsprinzip für Boole'sche Ausdrücke?

Vertauscht man UND mit ODER und negiert man alle Variablen, so erhält man einen neuen Ausdruck Y, welcher dual zu X ist. Dass Y dual zu X ist bedeutet, dass Y das genaue Gegenteil von X, also ein invertiertes X, ist.

18 Wie drückt man AND, OR und NOT aus wenn man nur NAND hat?

a AND b, *mit* **(a NAND b) NAND (a NAND b)** $\rightarrow$ *mit* 3 NANDs *realisiert*
$a \wedge b = \neg(\neg(a \wedge b) \wedge \neg(a \wedge b))$

NOT a, *mit* **a NAND a** *realisiert*
$\neg a = \neg(a \wedge a)$

a OR b, *mit* **(a NAND a) NAND (b NAND b)** $\rightarrow$ *mit* 3 NANDs *realisiert*
$a \vee b = \neg(\neg(a \wedge a) \wedge \neg(b \wedge b))$

19 Was sind Mikrobefehle, und wozu werden sie gebraucht?

Ein Mikrobefehl ist eine elementare Anweisung zur Steuerung eines Prozessors; bei einem Prozessor mit CISC-Architektur sind Mikrobefehle z.B. Prozessor-interne Steuercodes im Mikroprogrammsteuerwerk; sie steuern und verschalten die verschiedenen Arbeitseinheiten. Mehrere Microbefehle zusammengefasst ergeben den Microcode, der - ähnlich dem Assembly - sehr maschinennah ist. Er kann zum Steuern des Rechenwerks in einem Prozessor angesehen werden, wobei er sehr eng mit dem BIOS zusammenhängt.

evtl. zur Ergänzung:

Microcode ist Binärcode, der in einem festen Speicher im Prozessor abgelegt wird. Er wird vom Steuerwerk benutzt, um die Ausführungen von Befehlscodes zu realisieren. Erhält der Prozessor bspweise den Befehl ADD, so wird der entsprechende Microcode ausgeführt, der Register und ALUs steuert. Dies ist eine Alternative zu fest verdrahteten Befehlsimplementierungen. Auf modernen Prozessoren kommt häufig eine Mischung aus beiden Varianten zum Einsatz.

Microcode kann (meist temporär; über BIOS) durch eine aktuellere Version überschrieben werden.

20 Beschreiben Sie den Unterschied zwischen Latenz und Durchsatz. Was ist größer, was kleiner? Kann beides gleich sein?

Latenz ist die Zeit (*in Taktschlägen*), die der Prozessor benötigt, um einen Befehl abzuarbeiten, oder anders: Die Latenzzeit wird benötigt, bis das Ergebnis eines Befehls anderen Berechnungen zur Verfügung stehen kann. Durchsatz ist die Zeit, die benötigt wird, bis der gleiche Befehl nochmals in die Pipeline eingelesen werden kann, oder anders: Durchsatz ist die Zeit, in der eine *execution unit* zur Berechnung eines Befehls belegt ist, bevor sie einen weiteren Befehl berechnen kann. Die Latenz ist meistens geringer als der Durchsatz, z.B. bei der Addition *FADD* (3 vs. 1), **kann aber auch gleich sein**, z.B. beim Befehl, den Betrag einer Zahl zu bilden *FABS* (1 vs. 1), oder beim Wurzelziehen mit *FSQRT* (58 vs. 58). (Werte für Nehalem-Prozessoren mit CPUID 06_0DH)

21 Sind interpretierte oder kompilierte Programme schneller?

Kompilierte Programme sind schneller.

22 Erklären Sie den Unterschied zwischen statischem und dynamischen Linken.

Beim Statischen Linken werden alle benötigten Bibliotheken in das Programm gepackt, um von da aus darauf zugreifen zu können. Beim Dynamischen Linken werden die Bibliotheken nicht inkludiert, sondern beim Ausführen des Programms erst aus anderen Dateien geladen, wodurch man im Falle eines Veränderung äußerhalb"des Programms, z.B. nach Verbesserung einer Funktion einer gemeinsam genutzten Library, nicht das gesamte Programm neu kompilieren muss. Zudem können sich mittels dynamischen Linkens mehrere Programme die gleichen Libraries teilen, was Platz sparen kann.

23 Der Compiler meldet einen Linkfehler. Was kann die Ursache sein, was können Sie tun?

Eine mögliche Ursache sind Mehrfachdeklarationen (und damit verbunden -definitionen) gleichlautender Variablen oder Funktionen, die jedoch in verschiedenen Projektdateien/Libraries auftreten, und dem Kompiler daher nicht unmittelbar auffallen. Beim Linken aber weiß der Linker nicht, welche der „doppelten“ Funktionen er nun benutzen soll, es kommt zum Abbruch.

Lösung: Namenskonflikte beheben.

24 Welche zwei ganz unterschiedliche Bedeutungen hat das Wort "Heap"

Ein Heap kann eine zumeist auf Bäumen basierende, abstrakte Datenstruktur oder einen dynamischen Speicherbereich, aus dem zur Laufzeit eines Programms zusammenhängende Speicherabschnitte angefordert und in beliebiger Reihenfolge wieder freigegeben werden können, bezeichnen.

25 Was ist eine rekursive Funktion? Geben Sie ein sinnvolles Beispiel!

Eine rekursive Funktion ist eine sich wiederholt selbst aufrufende Funktion, z.B. die Fibonacci-Folge, wobei die „nächste“ Zahl immer auf die vorherigen 2 Zahlen in der Reihe zurückgreift, sie miteinander addiert und den Wert der Summe annimmt.

26 Kann man strukturiertes und objektorientiertes Programmieren kombinieren?

Ja. Objektorientiertes Programmieren baut auf dem strukturierten Programmieren auf und erweitert dieses.

C++, eine objektorientierte Programmiersprache, ist aus C entstanden, was wiederum eine strukturierte Programmiersprache ist.

27 Wie lange dauert das Finden eines Elementes in einem sortierten Array der Länge n?

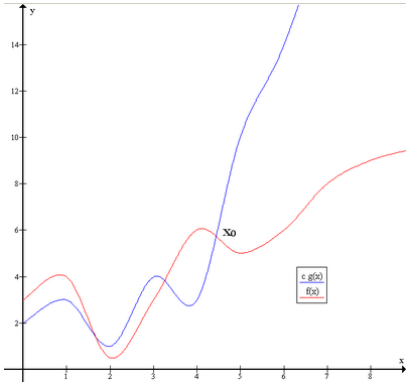
Mithilfe der linearen Suche dauert es schlimmstenfalls $O(n)$. Da das Array sortiert ist, kann man darüber hinaus noch die binäre Suche verwenden, die maximal $O(log\, n)$ dauert.

28 Was bringt es, Binärbäume zu balancieren?

Balancierte Binärbäume sind „ausgewogen“, alle Blätter weisen (in etwa) die gleiche Tiefe auf, es gibt n Knoten (Höhe $log_2(n+1)$?) und die Suche im Baum beträgt max. $log_2(n+1)$.

29 Ist $f(n) = n^2 \in O(n^3)$?

Ja. Es gilt: $n^2 \in O(n^3)$, da $\exists\, c > 0 \quad \exists\, x_0 \quad \forall\, x > x_0 : \quad x^2 < c \cdot x^3$



30 Was bedeutet „Matrizenmultiplikation ist $O(n^3)$“ und warum ist das so?

Das bedeutet, dass der Algorithmus zur Berechnung des Produkts zweier Matrizen eine Laufzeit von $O(n^3$ aufweist, wobei n die Anzahl der Spalten- oder Zeilenelemente einer quadratischen Matrix beschreibt. (Die Matrizenmultiplikation ist nur für quadratische Matrizen kubisch!)

Der Algorithmus verwendet drei Schleifen. Zwei werden benötigt, um in der Ergebnismatrix C alle Zeilen und Spalten durchzugehen, die innerste, dritte Schleife multipliziert dabei alle Elemente der Zeile i von Matrix A mit der Spalte i von Matrix B und summiert sie auf. Ein Algorithmus mit drei Schleifen besitzt per definitionem eine Komplexität von $O(n^3)$.

innere Schleife: $c_{ik} = \sum_{j=1}^{m} a_{ij} \cdot b_{jk}$

31 Kann man in linearer Zeit sortieren?

Ja. Z.B. mittels RadixSort, CountingSort, BucketSort, etc.

Die Ausgangslage muss aber gewissen Bedingungen genügen. Völlig zufällig angeordnete Daten, über die keine Informationen bekannt sind ,lassen sich nicht linear sortieren.

32 Was ist der entscheidende Unterschied zwischen Internet und analogem Telefonnetz?

Das Internet ist verbindungs- bzw. paketorientiert; die Pakete werden einzeln durch die Leitung geschickt, die nach diesem Vorgang wieder freiïst und wieder für ein neues Paket verwendet werden kann. Beim analogen Telefonnetz ist die Leitung für den Dauer des gesamten Anrufs belegt; während dieses Vorgangs bleibt eine dauerhafte Verbindung bestehen, weil keine einzelnen Pakete in diesem Sinn übertragen werden.

33 Beschreiben Sie das OSI-Schichtenmodell

Das OSI-Modell soll Kommunikation über unterschiedliche technische Systeme hinweg ermöglichen und die Weiterentwicklung begünstigen. Das Modell besteht aus 7 aufeinander folgenden Schichten mit jeweils eigenen Aufgaben; Layer 1-4 gehören zum Transportsystem und Layer 5-7 sind anwendungsorientierte Schichten. Wenn nun ein Datenpaket von einem Sender kommt, durchläuft es alle Schichten (7 bis 1, nicht umgekehrt!), wobei jede Schicht dem Paket Protokoll-Informationen zufügt. Der 1. Layer wandelt das Datenpaket inklusive aller Protokoll-Informationen dann schließlich in technisch übertragbare Daten um und schickt es über das Übertragungsmedium (Funk/Kabel) zum Empfänger.

34 Geben Sie ein Beispiel für Programmcode, der syntaktisch korrekt ist, nicht aber semantisch.

Man fragt den User über printf nach den Zahlen von 1 bis 10, man fragt mit scanf die Zahlen ab, lässt aber nur das Abspeichern dreier Variablen zu; sprich, der User kann nur 3 und nicht 10 Zahlen eingeben. (Absolut sinnlos, aber das Programm kompiliert und läuft, weil der Compiler ja nicht versteht, was man in printf geschrieben hat) ist das ein gültiges Beispiel??? Einfaches Bsp.

```
# include <stdio.h>
# include <stdlib.h>

int main()
{
    printf("Geben Sie zwei zu addierende Zahlen ein:\n");
    int a,b;
    printf("Zahl 1: ");
    scanf("%d",&a);
    printf("Zahl 2: ");
    scanf("%d",&b);
    printf("Ergebnis: %d\n",a*b); //man beachte die Multipl.
    return 0;
}
```

35 Zeichnen Sie einen endlichen Automaten, der nur gültige Variablennamen (Java) akzeptiert.

WORK IN PROGRESS!

zu ergänzen

36 Warum kann ein endlicher Automat keine Klammerausdrücke erkennen?

Er kann sie nicht erkennen, weil ein endlicher Automatik Schritt-für-Schritt vorgeht, sodass er selbst nicht erkennen kann, wann ein „Klammer-Zu“ auf das passende „Klammer-Auf“ trifft.

37 Warum gibt es nur abzählbar unendlich viele Computerprogramme?

Die Wortmenge über einem Alphabet ist eine abzählbar unendliche Menge - nicht jedes der entstehenden Wörter ist tatsächlich semantisch korrekt, und nicht jedes Programm, das mithilfe des Alphabets einer Programmiersprache zusammengeschrieben wurde, ist tatsächlich semantisch korrekt, weshalb sich der Schluss ergibt, dass es nur abzählbar unendlich viele Computerprogramme gibt. (Programme sind Texte endlicher Länge über einem endlichen Alphabet.)

38 Beschreiben Sie das 3-COLOR-Problem

Gefragt ist beim 3-COLOR-Problem, ob die Knoten eines einfachen Graphen so mit drei Farben einfärbbar sind, dass zueinander benachbarte Knoten unterschiedliche Farben haben. (Das Problem ist NP-vollständig.)

39 Wie funktioniert eine Turingmaschine?

Die Turingmaschine bildet Zeichenketten, die anfangs auf einem Band stehen, auf Zeichenketten, die nach "Bearbeitung"durch die Maschine auf dem Band stehen, ab. Genauer gesagt modifiziert die Maschine die Eingabe auf dem Band nach dem festgelegten Programm. Die Startposition der Turingmaschine ist am Anfang des Eingabeworts (1. Eingabezeichen). Mit jedem Schritt liest der Lese-Schreib-Kopf das aktuelle Zeichen, überschreibt dieses mit einem anderen (oder dem gleichen) Zeichen und bewegt sich dann ein Feld nach links oder rechts oder bleibt stehen. Welches Zeichen geschrieben wird und welche Bewegung gemacht wird, hängt von dem an der aktuellen Position vorgefundenen Zeichen sowie dem Zustand ab, in dem sich die Turingmaschine gerade befindet.

40 Erkennt ein endlicher Automat, eine Turingmaschine oder ein Kellerautomat die Sprache $a^n b^n c^n$?

Nur die Turingmaschine erkennt sie.

Zusatz:
$L1 = a^n b^m | n, m > 0$ mit endlichem Automaten lösbar
$L2 = a^n b^n | n > 0$ mit Kellerautomat lösbar
$L3 = a^n b^n c^n | n > 0$ nur mit Turingmaschine lösbar