Zu Übung 06, Besprechung: Do., 14.07.2016, 14^{00} – Neue Chemie

Aufgabe 7.01: Verständnisfragen

a) Ein Algorithmus ist eine genau definierte Handlungsvorschrift zur Lösung eines Problems oder einer bestimmten Art von Problemen in _____ vielen Schritten.

b) Dynamische Finitheit besagt, dass das Verfahren zu jedem Zeitpunkt nur endlich viel _____ benötigen darf.

c) Komplexitätstheorie bezeichnet das Verhalten von Algorithmen bezüglich Ressourcenbedarf, wie _____ und _____.

d) Berechenbarkeitstheorie besagt, dass:

   i. der Algorithmus bei denselben Voraussetzungen das gleiche Ergebnis liefern muss.

   ii. jeder Schritt des Verfahrens tatsächlich ausführbar sein muss.

   iii. das Verhalten bezüglich der Terminierung (ob der Algorithmus überhaupt jemals erfolgreich beendet werden kann) erfüllt sein soll.

e) Ein Algorithmus kann mit _____ oder _____ beschrieben werden.

f) Der Quicksortalgorithmus benötigt, abgesehen von dem für die Rekursion benötigten Platz auf dem Aufruf-Stack, keinen zusätzlichen Speicherplatz. Richtig / Falsch

g) Die Laufzeit des Quicksortalgorithmus ist abhängig von

_____

h) Merge Sort ist ein Sortieralgorithmus, der auf dem Prinzip _____ basiert.

i) Die Tiefensuche ist geeignet, um Eigenschaften von allen Knoten in einem Graphen auf dem Bildschirm auszugeben. Richtig/Falsch

j) Nennen Sie vier verschiedene Suchalgorithmen.

k) Wann heißen zwei Graphen G1 und G2 isomorph?

l) Welche zusätzlichen Eigenschaften muss ein Graph erfüllen, damit er ein Baum ist?

m) Beim Optimierungsverfahren Random-Interchange wird ein Austausch auch beibehalten, falls dies zu einer Verschlechterung der Kosten führt. Richtig / Falsch

n) Beim Optimierungsverfahren Simulated Annealing wird ein Schritt, welcher zu einer Verbesserung führt immer ausgeführt.
Richtig / Falsch

o) Beim Kernighan-Lin-Algorithmus wird eine Vertauschung nicht durchgeführt, wenn Sie zu einer Verschlechterung der Gesamtkosten führt. Richtig / Falsch

p) Beschreiben Sie das Prinzip der Rekombination (Kreuzung) in Bezug auf Evolutionäre Algorithmen:

_____

_____

Aufgabe 7.02: Arrays und Sortieren

Schreiben Sie ein Programm, welches mit Hilfe eines InsertionSort-Algorithmus ein 1-dimensionales Array in aufsteigender Reihenfolge sortiert. Testen Sie anschließend Ihr Programm mit geeigneten Testarrays.

Der folgende Pseudocode beschreibt den InsertionSort-Algorithmus,

Hinweis: Beachten Sie, dass bei C++ ein Array an der Stelle 0 beginnt, dies ist nicht im Pseudocode berücksichtigt.

```
1 for j = 2 to length[A]
2 do key = A[j]
  //Füge A[j] in die sortierte Folge A[1 .. j – 1] ein
3    i = j – 1
4    while( i > 0 and A[i] > key )
5    do A[i + 1] = A[i]
6       i = i – 1
7    A[i + 1] = key
```

Aufgabe 7.03: Laufzeitanalyse von Insertion Sort

In dieser Übung soll die Laufzeit vom "Sortieren durch Einfügen" bzw. Insertion Sort jeweils für den worst- und bestcase bestimmt werden. Hierzu soll anhand des nachstehenden Pseudo-Codes die Gesamtlaufzeit *T(n)* in beiden Fällen ermittelt und jeweils eine obere asymptotische Schranke bestimmt werden. Dabei nimmt man den Wert *n* für die Länge des zu sortierenden Arrays *A*. Noch zu beachten ist, dass c_i die Kosten der Codezeile *i* darstellt.

```
INSERTIONSORT( A )
1 for j = 2 to length[A]
2 do key = A[j]
  //Füge A[j] in die sortierte Folge A[1 .. j - 1] ein
3    i = j - 1
4    while( i > 0 and A[i] > key )
5    do A[i + 1] = A[i]
6       i = i - 1
7    A[i + 1] = key
```

Aufgabe 7.04: Tiefensuche

Der folgende Pseudocode beschreibt die Tiefensuche:

```
DepthFirstSearch( G )
  for alle Knoten u in G
  do farbe[u] = weiss
     vater[u] = NIL
  zeit = 0
  for alle Knoten u in G
  do if farbe[u] == weiss
     then DFS-VISIT( u )

DFS-VISIT( u )
  farbe[u] = grau; zeit = zeit + 1
  startTime[u] = zeit
  for alle Knoten v aus Adj[u]
  do if farbe[v] == weiss
     then vater[v] = u
          DFS-VISIT( v )
  farbe[u] = schwarz; zeit = zeit + 1
  endTime[u] = zeit
```

Gegeben ist die folgende Adjazenzmatrix. Wenden Sie darauf den Algorithmus der Tiefensuche an und zeichnen Sie den daraus resultierenden Suchbaum (Startknoten A). Markieren Sie dabei (mit einer fortlaufenden Zahl) im jeweiligen Knoten den Zeitpunkt des Entdeckens (**`startTime`** im Algorithmus) des Knotens und den Zeitpunkt, wann der Knoten vollständig bearbeitet worden ist (**`endTime`** im Algorithmus). Knoten mit einem kleineren Buchstaben (A ist kleiner als B) werden dabei zuerst gefunden.

	A	B	C	D	E	F	G	H	I	J	K
A	0	1	0	0	1	0	0	0	0	0	0
B	0	0	1	1	0	0	0	0	0	0	0
C	1	0	0	0	0	0	0	0	0	0	0
D	0	0	1	0	0	0	0	0	0	0	0
E	0	1	0	0	0	0	0	0	0	0	0
F	0	0	0	0	0	0	1	0	0	1	1
G	0	0	0	0	1	0	0	1	1	0	0
H	0	0	0	0	0	0	1	0	0	0	0
I	0	0	0	0	0	0	0	1	0	0	0
J	0	0	0	0	0	0	0	0	0	0	1
K	0	0	0	0	0	0	1	0	0	0	0

Zusatzaufgabe 7.05: Hardwarenahe Programmierung

Schreiben Sie eine C++ Codesequenz, die in einer Endlosschleife den 32 Bit Wert aus dem Analog-Digital-Wandler (ADC) des Mikrocontrollers Atmel AVR32 ausliest.

Es soll nur gelesen werden, sobald ein neuer Wert im ADC zur Verfügung steht. Ist dies der Fall, wird das DATAREADY (DRDY) Bit im Status-Register (SR) des ADC gesetzt (DRDY = Bit 16 im SR). Der neu gewandelte Wert ist sodann im Last-Converted-Data-Register (LCDR) abgelegt und kann ausgelesen werden. Das DRDY Bit wird nach dem Lesen des LCDR Registers automatisch gelöscht.

Hinweis: Verwenden Sie folgende vordefinierten Variablen zur Ansteuerung des ADC Geräts und dessen Register:

```
volatile uint32_t* adc_basis = 0xFFFF3C00;
uint32_t sr_offset = 0x0000001C;
uint32_t lcdr_offset = 0x00000020;
uint32_t sr_value, lcdr_value, data_ready;
```

`uint32_t` ist ein Datentyp, der 32 Bit **`unsigned int`** Werte darstellt.