

Übung 07: Informationstechnik (IT)

Marc Weber

Institutsleitung

Prof. Dr.-Ing. Dr. h.c. J. Becker

Prof. Dr.-Ing. E. Sax

Prof. Dr. rer. nat. W. Stork

Institut für Technik der Informationsverarbeitung (ITIV)



Besprechung Aufgabenblatt 7

Aufg. 7.01: Verständnisfragen Lsg. (1)

- a) Ein Algorithmus ist eine genau definierte Handlungsvorschrift zur Lösung eines Problems oder einer bestimmten Art von Problemen in endlich vielen Schritten.
- b) Dynamische Finitheit besagt, dass das Verfahren zu jedem Zeitpunkt nur endlich viel Speicherplatz benötigen darf.
- c) Komplexitätstheorie bezeichnet das Verhalten von Algorithmen bezüglich Ressourcenbedarf, wie Rechenzeit und Speicherbedarf.

Aufg. 7.01: Verständnisfragen Lsg. (2)

- d) Berechenbarkeitstheorie besagt, dass:
- i. der Algorithmus bei denselben Voraussetzungen das gleiche Ergebnis liefern muss. → Falsch
 - ii. jeder Schritt des Verfahrens tatsächlich ausführbar sein muss. → Falsch
 - iii. das Verhalten bezüglich der Terminierung (ob der Algorithmus überhaupt jemals erfolgreich beendet werden kann) erfüllt sein soll. → Richtig
- e) Ein Algorithmus kann mit Pseudo Code oder Nassi-Shneiderman Diagrammen oder Ablaufdiagrammen beschrieben werden.
- f) Der Quicksortalgorithmus benötigt, abgesehen von dem für die Rekursion benötigten Platz auf dem Aufruf-Stack, keinen zusätzlichen Speicherplatz. → Richtig

Aufg. 7.01: Verständnisfragen Lsg. (3)

- g) Die Laufzeit des Quicksortalgorithmus ist abhängig von der Zerlegung des Feldes (Aufbau des Feldes - Zufall).
- h) Merge Sort ist ein Sortieralgorithmus, der auf dem Prinzip Teile und Herrsche basiert.
- i) Die Tiefensuche ist geeignet, um Eigenschaften von allen Knoten in einem Graphen auf dem Bildschirm auszugeben. → Richtig
- j) Nennen Sie vier verschiedene Suchalgorithmen.
Lineare Suche, Binäre Suche, Interpolationssuche, Breitensuche, Tiefensuche

Aufg. 7.01: Verständnisfragen Lsg. (4)

- k) Wann heißen zwei Graphen G_1 und G_2 isomorph?
Zwei Graphen G_1 und G_2 heißen isomorph, wenn es eine bijektive Abbildung der Knoten- und Kantenmengen von G_1 auf die entsprechenden Mengen von G_2 gibt, so dass die Inzidenzbeziehungen erhalten bleiben.
- l) Welche zusätzlichen Eigenschaften muss ein Graph erfüllen, damit er ein Baum ist?
Ein Baum ist zyklensfrei und zusammenhängend.
- m) Beim Optimierungsverfahren Random-Interchange wird ein Austausch auch beibehalten, falls dies zu einer Verschlechterung der Kosten führt. → Falsch

Aufg. 7.01: Verständnisfragen Lsg. (5)

- n) Beim Optimierungsverfahren Simulated Annealing wird ein Schritt, welcher zu einer Verbesserung führt immer ausgeführt. → Richtig
- o) Beim Kernighan-Lin-Algorithmus wird eine Vertauschung nicht durchgeführt, wenn Sie zu einer Verschlechterung der Gesamtkosten führt. → Falsch
- p) Beschreiben Sie das Prinzip der Rekombination (Kreuzung) in Bezug auf Evolutionäre Algorithmen:
Geschlechtliche Fortpflanzung: Zwei Individuen vermischen ihre Erbanlagen und "zeugen" zwei Nachkommen, die die Position ihrer Eltern in der Population übernehmen, d.h. Nachkommen ersetzen ihre Vorfahren.

- Aufg. 7.01: Verständnisfragen



Aufg. 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.

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

Aufg. 7.02: Arrays und Sortieren Lsg.

```

int main() {
    long matrix[10] = {546, 21, 65, 1, 987, 88, 654, 5, 98, 123};
    int i, j, key;

    for( j = 1; j <= 9; j++ ) {
        key = matrix[j];
        i = j - 1;
        while( i > -1 && matrix[i] > key ) {
            matrix[i+1] = matrix[i];
            i = i - 1;
        }
        matrix[i+1] = key;
    }

    for( i = 0; i <= 9; i++ )
        cout << setw( 4 ) << matrix[i];
    cout << endl;

    return 0;
}
  
```

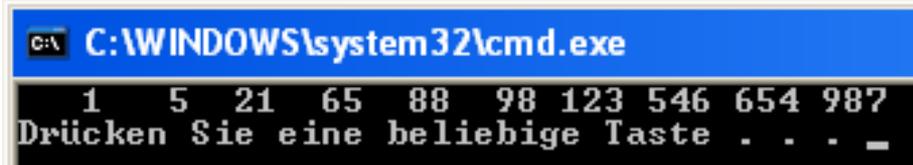
Startet bei 1, da Array bei 0 beginnt

Jedes Element nach dem Ersten

Schiebt die größeren Elemente nach hinten im Array

Fügt das Element an der richtigen Stelle ein

Arrayausgabe



```

C:\WINDOWS\system32\cmd.exe
1  5 21 65 88 98 123 546 654 987
Drücken Sie eine beliebige Taste . . . _
  
```

- Aufg. 7.02: Arrays und Sortieren



Fragen?

Aufg. 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]
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
```

Aufg. 7.03: Laufzeitanalyse von Insertion Sort

Lsg. (1)

```
INSERTIONSORT( A )
```

```
1 for j = 2 to length(A)
2 do key = A[j]
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
```

Hinweis:

$$\sum_{j=1}^n (j) = \frac{n \cdot (n+1)}{2}$$

				Array A			
				1	2	3	4
j	i	key		8	6	4	2
2	1	6		8	8	4	2
2	0	6		6	8	4	2
3	2	4		6	8	8	2
3	1	4		6	6	8	2
3	0	4		4	6	8	2
4	3	2		4	6	8	8
4	2	2		4	6	6	8
4	1	2		4	4	6	8
4	0	2		2	4	6	8

Aufg. 7.03: Laufzeitanalyse von Insertion Sort

Lsg. (2)

- Best-Case Betrachtung ($n =$ Länge des Arrays)
 - Array ist vorsortiert \rightarrow keine Verschiebungen notwendig
 - Schleife von Zeile 1-7 wird $n-1$ mal ausgeführt
 - Und Zeile 1 noch einmal zusätzlich wenn die Schleife abgebrochen wird
 - Schleife in Zeile 4 wird sofort abgebrochen, da immer $A[i] < \text{key}$
 - Zeile 5 & 6 wird nie ausgeführt

				Array A			
				1	2	3	4
j	i	key		2	4	6	8
2	1	4		2	4	6	8
3	2	6		2	4	6	8
4	3	8		2	4	6	8

INSERTIONSORT(A)

```
1 for j = 2 to length(A)
2 do key = A[j]
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
```

Zeile	Kostenkoeffizient	Zeit (bestcase)
1	c_1	n
2	c_2	$n-1$
3	c_3	$n-1$
4	c_4	$n-1$
5	c_5	0
6	c_6	0
7	c_7	$n-1$

Aufg. 7.03: Laufzeitanalyse von Insertion Sort

Lsg. (3)

■ Worst-Case Betrachtung

- Array ist rückwärts sortiert
 - Verschiebungen immer notwendig
- Nur Änderungen in der inneren Schleife
 - Zeile 4 – 6 neu zu betrachten

$$Z_4 = j \sum_{j=2}^n Z_4 = \sum_{j=2}^n j = \left(\sum_{j=1}^n j \right) - 1$$

$$= \frac{n \cdot (n+1)}{2} - 1 = \frac{n^2 + n}{2} - 1$$

$$Z_5 = Z_6 = j - 1 \sum_{j=2}^n Z_{5,6} = \sum_{j=2}^n (j-1) = \sum_{k=1}^{n-1} (k) =$$

$$(k = j - 1) \quad = \frac{(n-1) \cdot n}{2} = \frac{n^2 - n}{2}$$

				Array A			
				1	2	3	4
j	i	key		8	6	4	2
2	1	6		8	8	4	2
2	0	6		6	8	4	2
3	2	4		6	8	8	2
3	1	4		6	6	8	2
3	0	4		4	6	8	2
4	3	2		4	6	8	8
4	2	2		4	6	6	8
4	1	2		4	4	6	8
4	0	2		2	4	6	8

Zeile	Kostenkoeffizient	Zeit (worstcase)
1	c ₁	n
2	c ₂	n-1
3	c ₃	n-1
4	c ₄	(n ² +n)/2 - 1
5	c ₅	(n ² -n)/2
6	c ₆	(n ² -n)/2
7	c ₇	n-1

Aufg. 7.03: Laufzeitanalyse von Insertion Sort

Lsg. (4)

```
INSERTIONSORT( A )
1 for j = 2 to length(A)
2 do key = A[j]
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
```

Zeile	Kostenkoeffizient	Zeit (worstcase)	Zeit (bestcase)
1	c_1	n	n
2	c_2	$n-1$	$n-1$
3	c_3	$n-1$	$n-1$
4	c_4	$(n^2+n)/2 - 1$	$n-1$
5	c_5	$(n^2-n)/2$	0
6	c_6	$(n^2-n)/2$	0
7	c_7	$n-1$	$n-1$

Worstcase:

$$T(n) = O(n^2)$$

Bestcase:

$$T(n) = O(n)$$

- Aufg. 7.03: Laufzeitanalyse von Insertion Sort



Fragen?

Aufg. 7.04: Tiefensuche (1)

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 )
  
```

Initialisierung

Für alle Knoten u im Graphen G

Falls der Knoten u noch nicht untersucht wurde

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
  
```

Für alle mit dem aktuellen Knoten u verbundenen Knoten (Adjazenz)

Rekursiver Aufruf

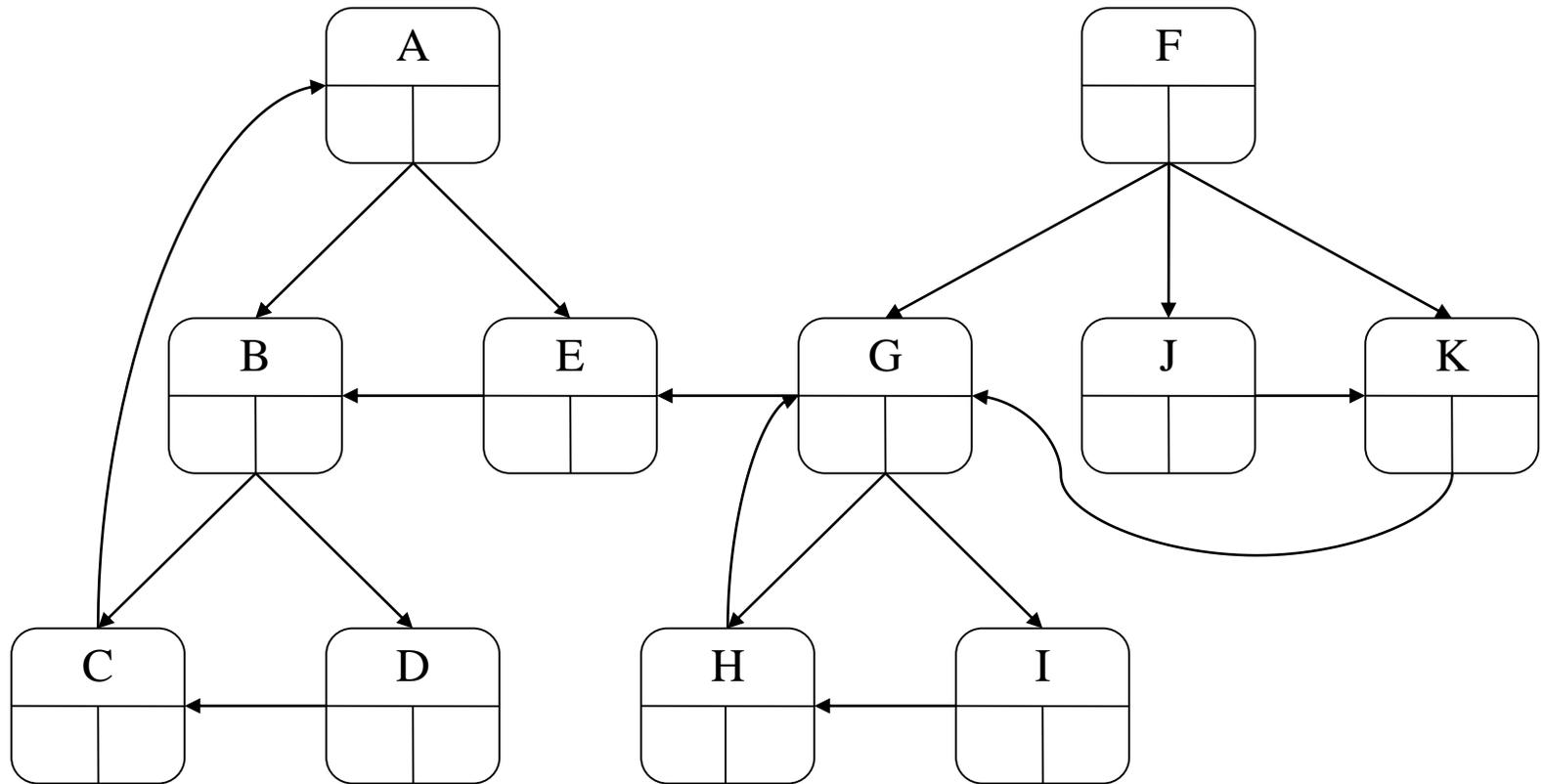
Aufg. 7.04: Tiefensuche (2)

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

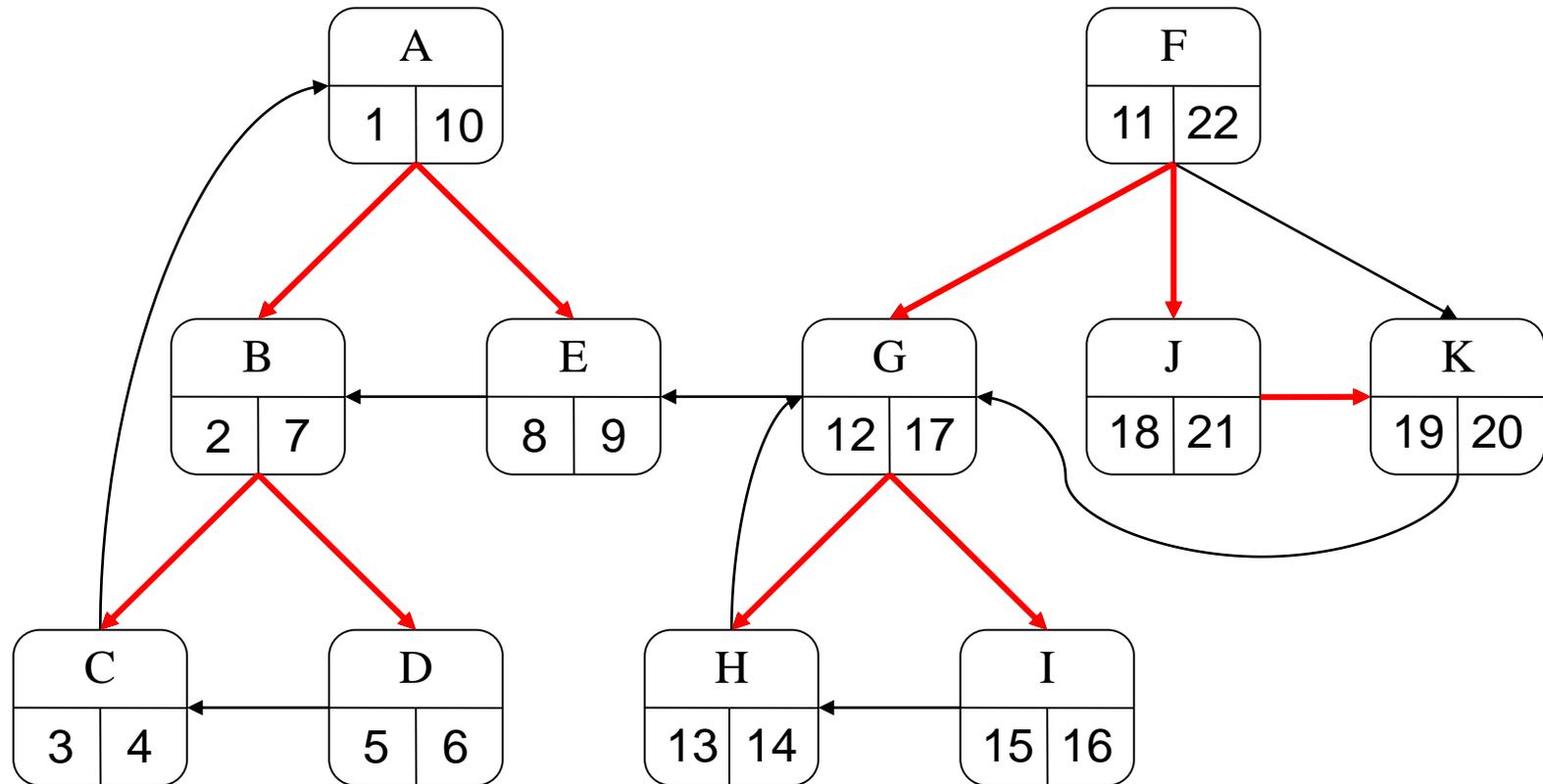
Aufg. 7.04: Tiefensuche Lsg. (1)

- Umsetzung in einen Graphen



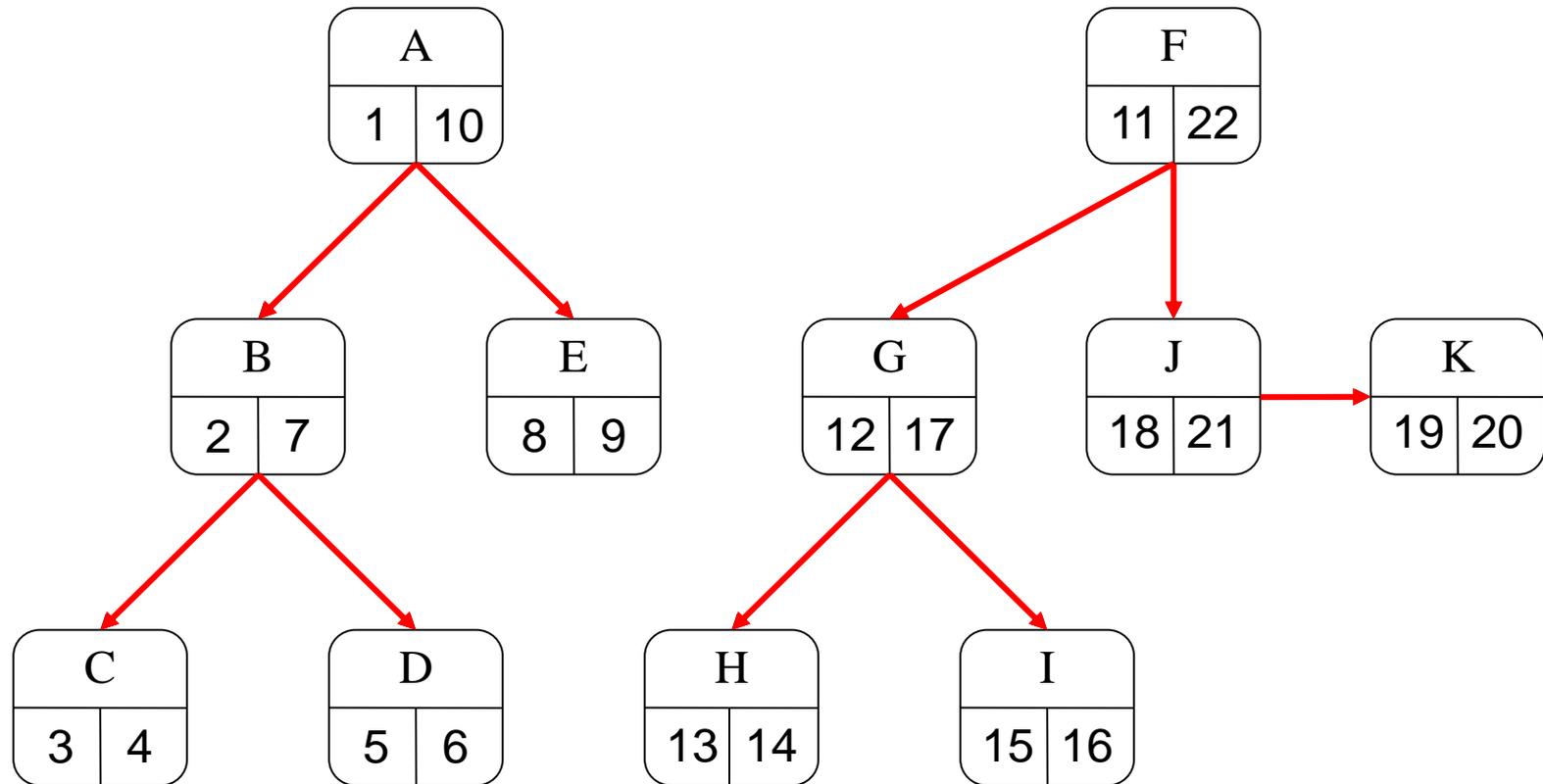
Aufg. 7.04: Tiefensuche Lsg. (2)

■ Anwendung der Tiefensuche



Aufg. 7.04: Tiefensuche Lsg. (3)

■ Tiefensuchwald



- Aufg. 7.04: Tiefensuche



Aufg. 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.
```

Aufg. 7.05: Hardwarenahe Programmierung

Lsg. (1)

Adresse	Gerät	Name
0xFFFE0000	USB	USB 2.0 Interface - USB
0xFFFE1000	HMATRIX	HSB Matrix - HMATRIX
0xFFFE1400	HFLASHC	Flash Controller - HFLASHC
0xFFFF0000	PDCA	Peripheral DMA Controller - PDCA
0xFFFF0800	INTC	Interrupt controller - INTC
0xFFFF0C00	PM	Power Manager - PM
0xFFFF0D00	RTC	Real Time Counter - RTC
0xFFFF0D30	WDT	Watchdog Timer - WDT
0xFFFF0D80	EIM	External Interrupt Controller - EIM
0xFFFF1000	GPIO	General Purpose Input/Output Controller - GPIO
0xFFFF1400	USART0	Universal Synchronous/Asynchronous Receiver/Transmitter - USART0
0xFFFF1800	USART1	Universal Synchronous/Asynchronous Receiver/Transmitter - USART1
0xFFFF1C00	USART2	Universal Synchronous/Asynchronous Receiver/Transmitter - USART2
0xFFFF2400	SPIO	Serial Peripheral Interface - SPIO
0xFFFF2C00	TWI	Two-wire Interface - TWI
0xFFFF3000	PWM	Pulse Width Modulation Controller - PWM
0xFFFF3400	SSC	Synchronous Serial Controller - SSC
0xFFFF3800	TC	Timer/Counter - TC
0xFFFF3C00	ADC	Analog to Digital Converter - ADC
0xFFFF4000	ABDAC	Audio Bitstream DAC - ABDAC

■ Analog-Digital-Converter

- Basisadresse: 0xFFFF3C00
- Größe (in Bytes): **0x400** = 0xFFFF4000 – 0xFFFF3C00 = 1024

Aufg. 7.05: Hardwarenahe Programmierung

Lsg. (2)

Offset	Register	Name	Access	Reset State
0x00	Control-Register	CR	Write-only	0x00000000
0x04	Mode-Register	MR	Read/Write	0x00000000
0x10	Channel-Enable-Register	CHER	Write-only	0x00000000
0x14	Channel-Disable-Register	CHDR	Write-only	0x00000000
0x18	Channel-Status-Register	CHSR	Read-only	0x00000000
0x1C	Status-Register	SR	Read-only	0x000C0000
0x20	Last-Converted-Data-Register	LCDR	Read-only	0x00000000
...
0xFC	Version-Register	VERSION	Read-only	Device-specific

■ Status-Register (SR)

- Enthält Status-Informationen über die Puffer des Wandlers
- Bit 16 zeigt an, ob neue Daten vom Wandler erzeugt wurden

■ Last-Converted-Data-Register (LCDR)

- Speichert den Wert der letzten AD-Wandlung
- wird nach jeder Konvertierung überschrieben

■ Direkter Zugriff auf Register mit Zeigern

- $\text{ADC.Registeradresse} = \text{ADC.Basisadresse} + \text{ADC.Offset}$

Aufg. 7.05: Hardwarenahe Programmierung

Lsg. (3)

25.7.6 Status Register

Name: SR

Access Type: Read-only

Offset: 0x1C

Reset Value: 0x000C0000

31	30	29	28	27	26	25	24
-	-	-	-	-	-	-	-
23	22	21	20	19	18	17	16
-	-	-	-	RXBUFF	ENDRX	GOVRE	DRDY
15	14	13	12	11	10	9	8
OVRE7	OVRE6	OVRE5	OVRE4	OVRE3	OVRE2	OVRE1	OVRE0
7	6	5	4	3	2	1	0
EOC7	EOC6	EOC5	EOC4	EOC3	EOC2	EOC1	EOC0

■ DRDY: Data Ready

- Bit 16 im Register → 17. Stelle von rechts
- Dieses Bit wird gesetzt, wenn ein neuer Wert im Register LCDR verfügbar ist; es wird gelöscht, wenn LCDR gelesen wird

Aufg. 7.05: Hardwarenahe Programmierung

Lsg. (4)

Offset	Register	Name	Access	Reset State
0x00	Control-Register	CR	Write-only	0x00000000
0x04	Mode-Register	MR	Read/Write	0x00000000
0x10	Channel-Enable-Register	CHER	Write-only	0x00000000
0x14	Channel-Disable-Register	CHDR	Write-only	0x00000000
0x18	Channel-Status-Register	CHSR	Read-only	0x00000000
0x1C	Status-Register	SR	Read-only	0x000C0000
0x20	Last-Converted-Data-Register	LCDR	Read-only	0x00000000
...
0xFC	Version-Register	VERSION	Read-only	Device-specific

■ ADC

- Basisadresse: 0xFFFF3C00
- Status-Register-Adresse: Basisadresse + Offset (0x1C)
- LDCR-Adresse: Basisadresse + Offset (0x20)
- Neuer Wert in LCDR: Bit 16 in SR gesetzt

Aufg. 7.05: Hardwarenahe Programmierung

Lsg. (5)

- LCDR auslesen, wenn neuer Wert verfügbar:

```
int main(){
    volatile uint32_t* adc_basis = 0xFFFF3C00; // Zeiger auf den ADC
    uint32_t sr_offset = 0x0000001C; // Offset zum Status-Register des ADC
    uint32_t lcdr_offset = 0x00000020; // Offset zum Daten-Register des ADC
    uint32_t sr_value, lcdr_value, data_ready;

    while(1) { // typisch für "embedded": Endlosschleife
        sr_value = *( adc_basis + sr_offset ); // * = Adresse auflösen
        data_ready = sr_value & 0x00010000; // Bit 16 ausmaskieren

        // Wenn data_ready nicht Null ist, steht ein neuer Wert in LCDR!
        if ( data_ready ) {
            lcdr_value = *( adc_basis + lcdr_offset ); // * = Adresse auflösen
            // Tue etwas mit lcdr_value
            // ...
        }
        // ...
    }
}
```

volatile hindert den Compiler daran, die Variable für den weiteren Programmverlauf wegzuoptimieren. Dies ist nötig, da sich der Register-Wert, der über die Adresse aufgelöst wird, unabh. vom Programm ändern kann. In diesem konkreten Fall ändert der ADC sein Status-Register zur Laufzeit unabh. vom Programm, sobald z.B. eine Wandlung abgeschlossen ist.

- Aufg. 7.05: Hardwarenahe Programmierung



Fragen?

(Probe-)Klausur

- Probeklausur und Fragestunde
 - Donnerstag, 21.07.2016, 14:00 bis 15:30 Uhr, Neue Chemie
 - Ggf. weitere Fragen bitte per E-Mail bis spätestens 23.08.2016!
 - Ich bin in vom 27.08.2016 bis 04.09.2016 nicht erreichbar!

- Klausur
 - Dienstag, 06.09.2016, 11:00 bis 13:00 Uhr
 - Hilfsmittel: Ein Blatt eigene Notizen (2-seitig, handschriftlich)
 - Anmeldebeginn: 01.08.2016
 - **Anmeldeende: 30.08.2016**
 - **Abmeldeende: 04.09.2016**
 - Hörsaalverteilung per Aushang und auf der Lernplattform nach Abmeldeende
 - **Viel Erfolg!**

Vielen Dank für Ihre Aufmerksamkeit



Marc Weber
Karlsruher Institut für Technologie (KIT) – ITIV
marc.weber3@kit.edu