

Übung 07: Informationstechnik (IT)

Daniel Grimm

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) Ein Algorithmus kann mit Pseudo Code oder Nassi-Shneiderman Diagrammen oder Ablaufdiagrammen beschrieben werden.
- c) Dynamische Finitheit besagt, dass das Verfahren zu jedem Zeitpunkt nur endlich viel Speicherplatz benötigen darf.
- d) Komplexitätstheorie bezeichnet das Verhalten von Algorithmen bezüglich Ressourcenbedarf, wie Rechenzeit und Speicherbedarf.

Aufg. 7.01: Verständnisfragen Lsg. (2)

- e) 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
- f) Merge Sort ist ein Sortieralgorithmus, der auf dem Prinzip Teile und Herrsche basiert.
- g) 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)

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

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.

- 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;
}
```

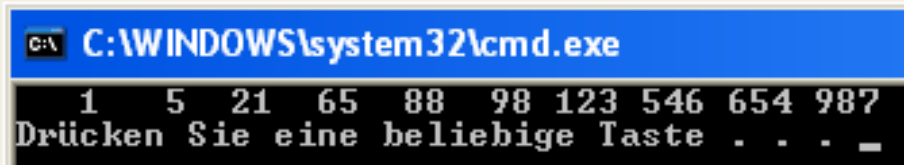
Startet bei 1, da Array bei 0 beginnt

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. Ermitteln Sie anhand des nachstehenden Pseudo-Codes die Gesamtlaufzeit $T(n)$ in beiden Fällen und bestimmen Sie jeweils eine obere asymptotische Schranke! Nehmen Sie dabei an, dass das zu sortierende Array A die Länge n hat. 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

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

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

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

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

				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



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

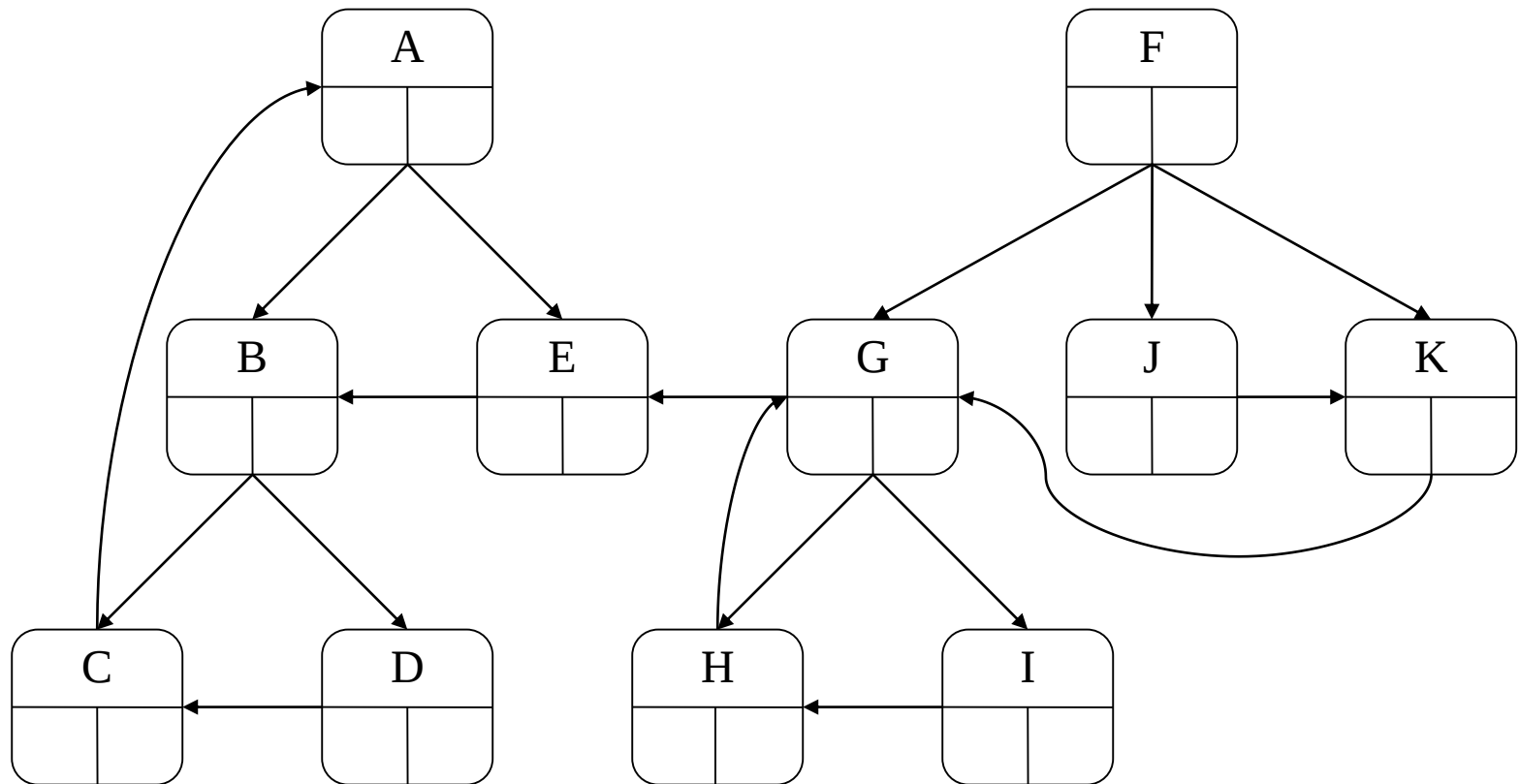
Aufg. 7.04: Tiefensuche (2)

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

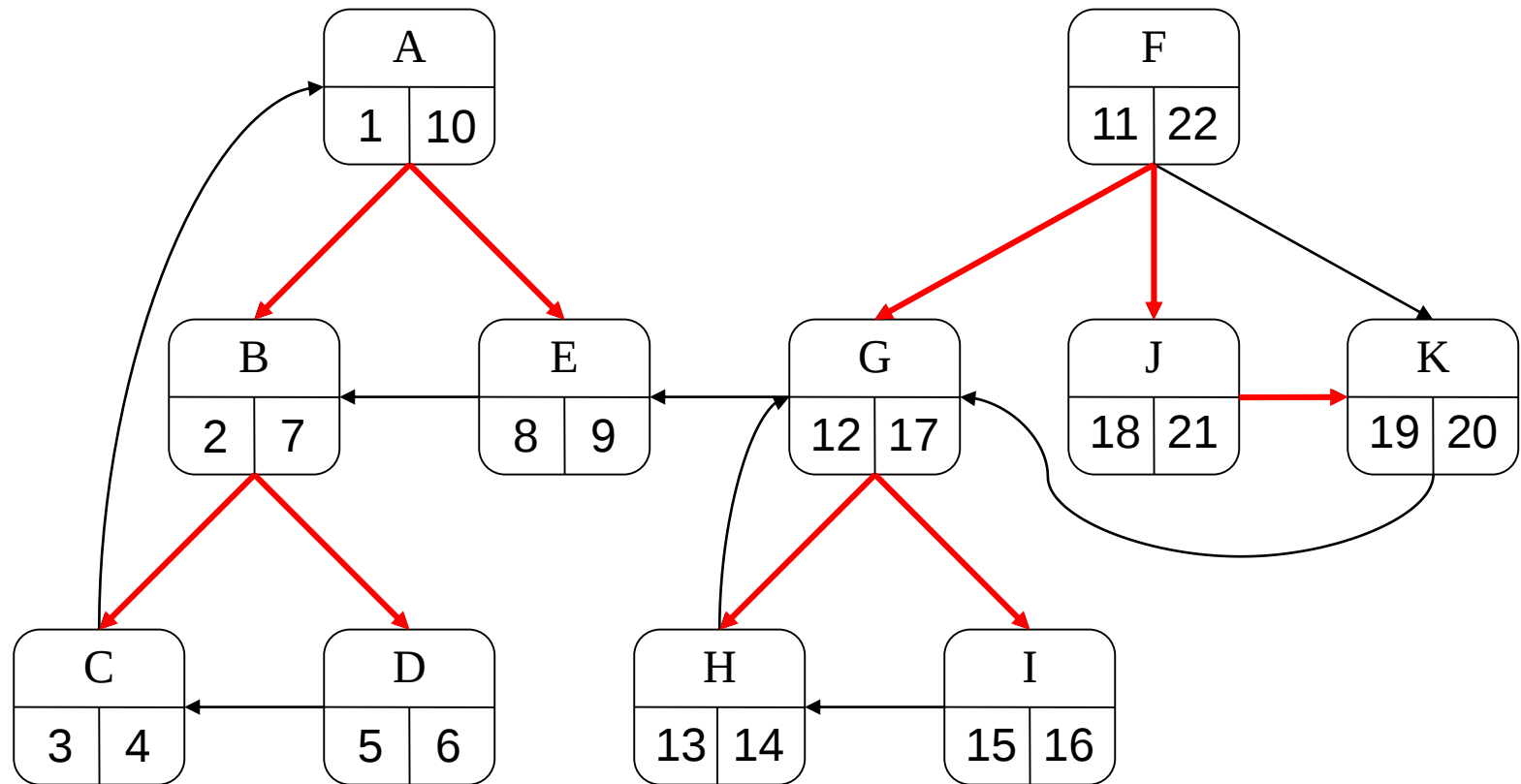
Aufg. 7.04: Tiefensuche Lsg. (1)

■ Umsetzung in einen Graphen



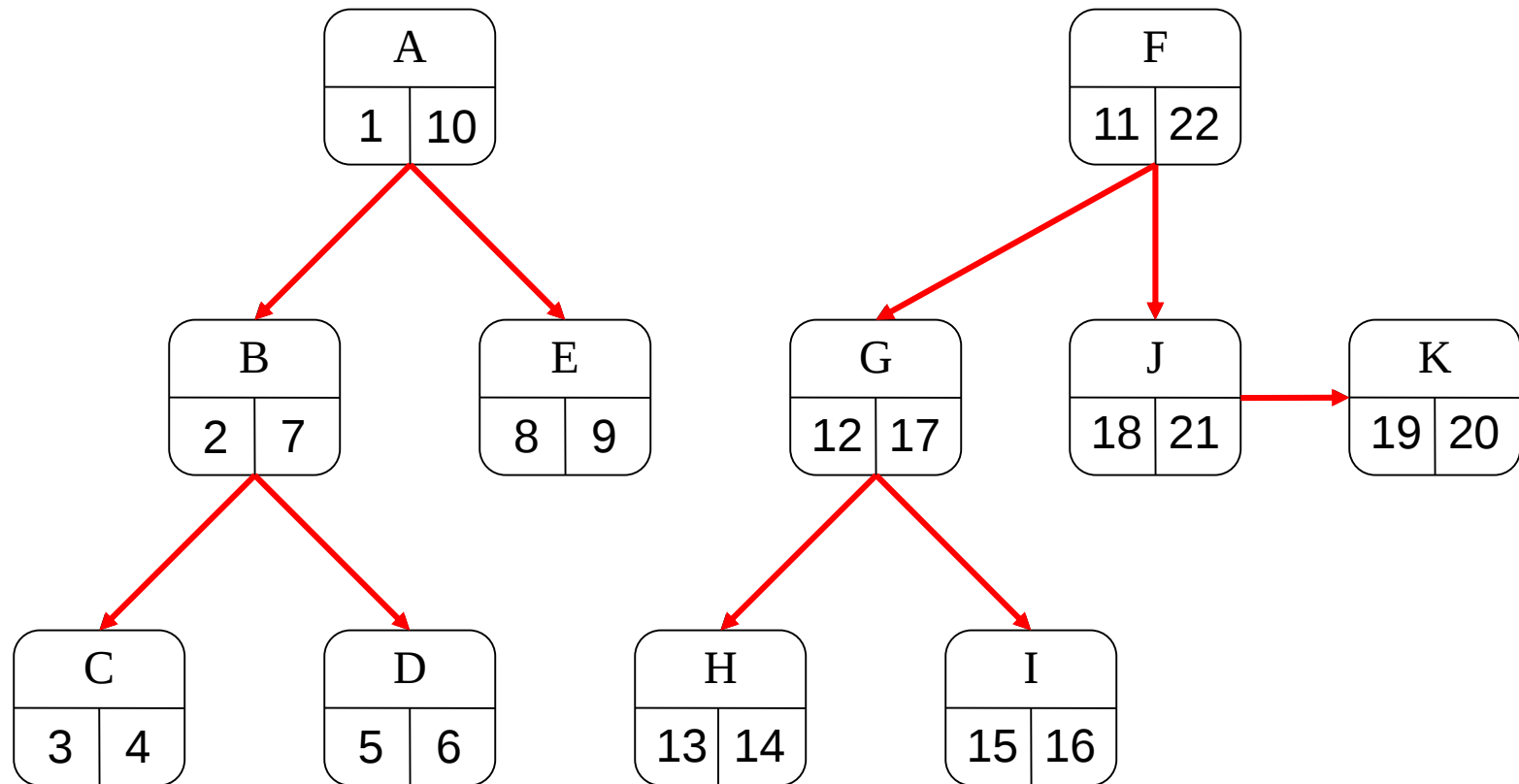
Aufg. 7.04: Tiefensuche Lsg. (2)

■ Anwendung der Tiefensuche



Aufg. 7.04: Tiefensuche Lsg. (3)

■ Tiefensuchwald



- Aufg. 7.04: Tiefensuche



Vielen Dank für Ihre Aufmerksamkeit



Daniel Grimm
Karlsruher Institut für Technologie (KIT) – ITIV
daniel.grimm@kit.edu