



## DAP2 Klausur

Datum: 21. Juli 2006

### Zu beachten:

- Leserlich schreiben!
- Nur dokumentenechte Schreibgeräte verwenden!
- Nur Papier verwenden, das von uns ausgeteilt wurde!
- Es sind keine Hilfsmittel erlaubt!

Die Klausur dauert 90 Minuten. Es gibt 60 Punkte zu erreichen. Zum Bestehen der Klausur sind 30 Punkte notwendig.

Wir als Klausurveranstalter sind organisatorisch nicht dazu in der Lage, vor- bzw. während der Klausur zu überprüfen, ob die Teilnehmer/-innen dazu berechtigt sind, die Klausur mitzuschreiben bzw. ob sie ordnungsgemäß bei der jeweils zuständigen Stelle angemeldet sind. Daher gilt: Durch Ihre **Unterschrift** erkennen Sie an, dass diese Klausur **unter Vorbehalt** stattfindet, d.h. die Teilnahmeberechtigung und Anerkennung der Klausur wird erst nach der Klausur von der jeweils zuständigen Stelle überprüft und ist nicht automatisch durch die Teilnahme an der Klausur gegeben. Darüber hinaus bestätigen Sie durch Ihre Unterschrift, dass Sie **prüfungsfähig** sind.

Matrikelnummer:

Nachname:

Vorname:

Studienrichtung:

Unterschrift:


### Punkte:

*(Nur vom  
Klausurveranstalter  
auszufüllen)*

Th1:		Th2:		Th3:		Gesamt:	
Th4:		Th5:					

**Thema 1.** (insg. 10 Punkte) *O-Notation*

*Aufgabe 1.a.* (5 Punkte)

Bestimme zu den beiden Codefragmenten jeweils die Best- und Worst-Case-Laufzeit in  $\Theta$ -Notation.

Die Variable  $A$  bezeichnet ein Array mit den Indizes  $1 \dots n$  von ganzen Zahlen zwischen 1 und  $n$ . Gib eine Belegung des Arrays an, in dem der Worst- bzw. Best-Case auftritt.

```
r := 0
for i := 2 to (n - 2) do
  for j := n to n2 do
    r := r + j - i
  end for
end for
```

```
for i := 1 to n do
  while A[i] > 3 do
    A[i] := A[i] - 1
  end while
end for
```

Linker Code, Best-Case:  $\Theta(\quad)$

Linker Code, Worst-Case:  $\Theta(\quad)$

Rechter Code, Best-Case:  $\Theta(\quad)$

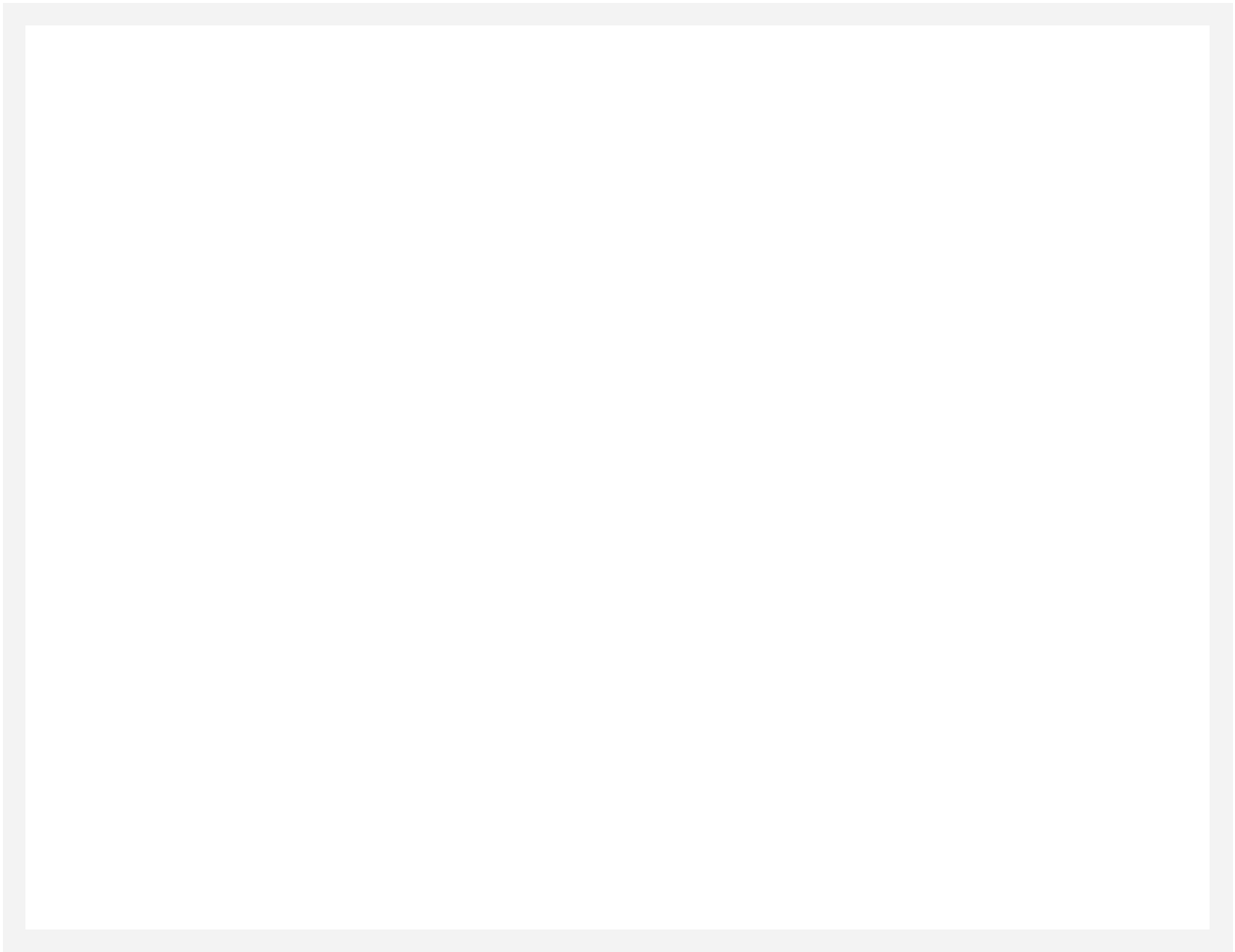
Rechter Code, Worst-Case:  $\Theta(\quad)$

Best-Case Belegung von  $A$ :  $A[1] = \square, A[2] = \square, A[3] = \square, \dots, A[n] = \square$

Worst-Case Belegung von  $A$ :  $A[1] = \square, A[2] = \square, A[3] = \square, \dots, A[n] = \square$

*Aufgabe 1.b.* (3 Punkte)

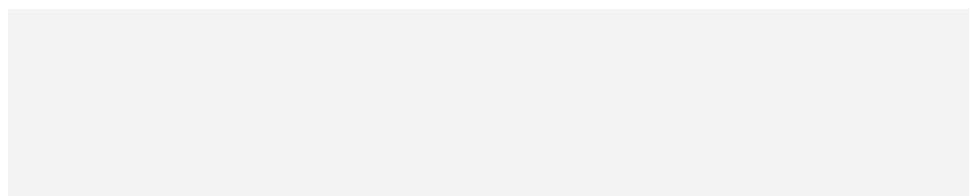
Gib die  $\Theta$ -Notation für die Laufzeitfunktion  $f(n) := 7n^{8/3} + n$  an und beweise sie formal.



*Aufgabe 1.c.* (2 Punkte)

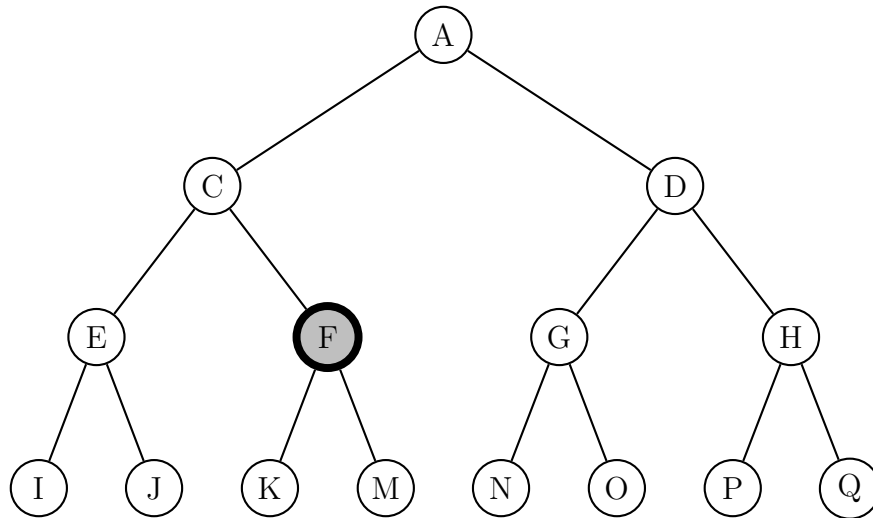
Wie kann man die  $\Theta$ -Notation ausschliesslich mit Hilfe der  $O$ -Notation definieren?

$$f(n) = \Theta(g(n)) \iff$$



**Thema 2.** (insg. 6 Punkte) *ADT*

Betrachte den folgenden MinHeap und die Operationen `DECREASEPRIORITY` und `INCREASEPRIORITY`. Bei beiden Funktionen wird der Schlüsselwert eines Eintrages geändert (gesenkt bzw. erhöht). Das entsprechende Element soll danach mit den aus der Vorlesung bekannten Sifting-Verfahren innerhalb des Baums so verschoben werden, dass die Heap-Eigenschaft wieder erfüllt ist.



*Aufgabe 2.a.* (3 Punkte)

Gehe vom in der Abbildung dargestellten Heap aus. Gib die Vergleiche an, die der Algorithmus durchführt, wenn wir das Element „F“ auf „B“ ändern (`DECREASEPRIORITY`).

*Aufgabe 2.b.* (3 Punkte)

Gehe vom in der Abbildung dargestellten Heap aus. Gib die Vergleiche an, die der Algorithmus durchführt, wenn wir das Element „F“ auf „L“ ändern (`INCREASEPRIORITY`).

**Thema 3.** (insg. 8 Punkte) *Sortieren*

*Aufgabe 3.a.* (4 Punkte)

Vergleiche das Quick-Sort Verfahren mit dem Merge-Sort Verfahren. Gib jeweils zwei Vorteile an, die das Verfahren gegenüber dem anderen hat.

Vorteil von Quick-Sort	Vorteil von Merge-Sort

*Aufgabe 3.b.* (4 Punkte)

Wir möchten  $n$  Integerzahlen aus dem Intervall  $[0, 2^a - 1]$  sortieren. Dazu benutzen wir Radix-Sort, das seinerseits Counting-Sort benutzt. Wir interessieren uns für den Einfluss des Basis-Parameters, und wählen deshalb einmal 2 als Basis für Radix-Sort, und einmal  $2^a$ . Gib jeweils die resultierende Laufzeit des Algorithmus in  $\Theta$ -Notation als Funktion von  $n$  und  $a$  an.

Laufzeit von Radix-Sort (inkl. Counting-Sort) mit Basis 2:  $\Theta(\quad)$

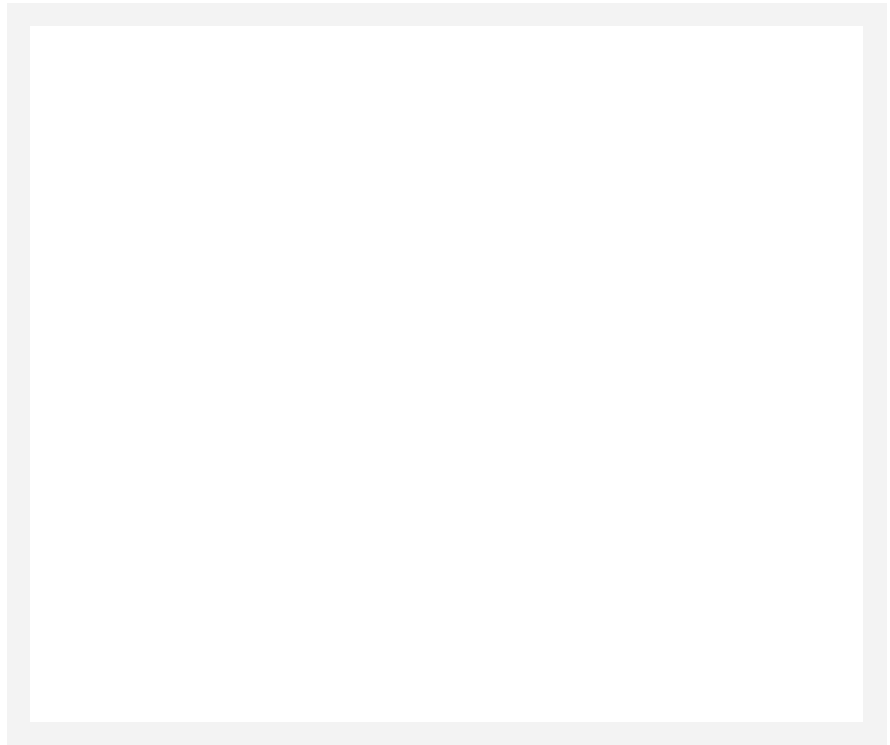
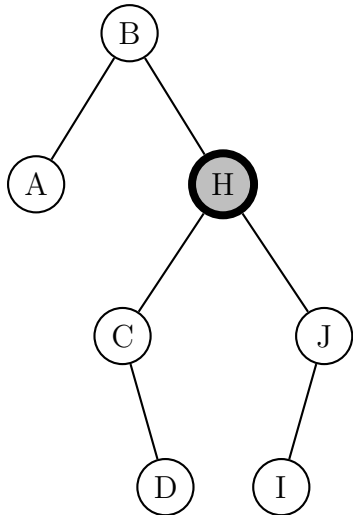
Laufzeit von Radix-Sort (inkl. Counting-Sort) mit Basis  $2^a$ :  $\Theta(\quad)$

**Thema 4.** (insg. 16 Punkte) *Suchen*

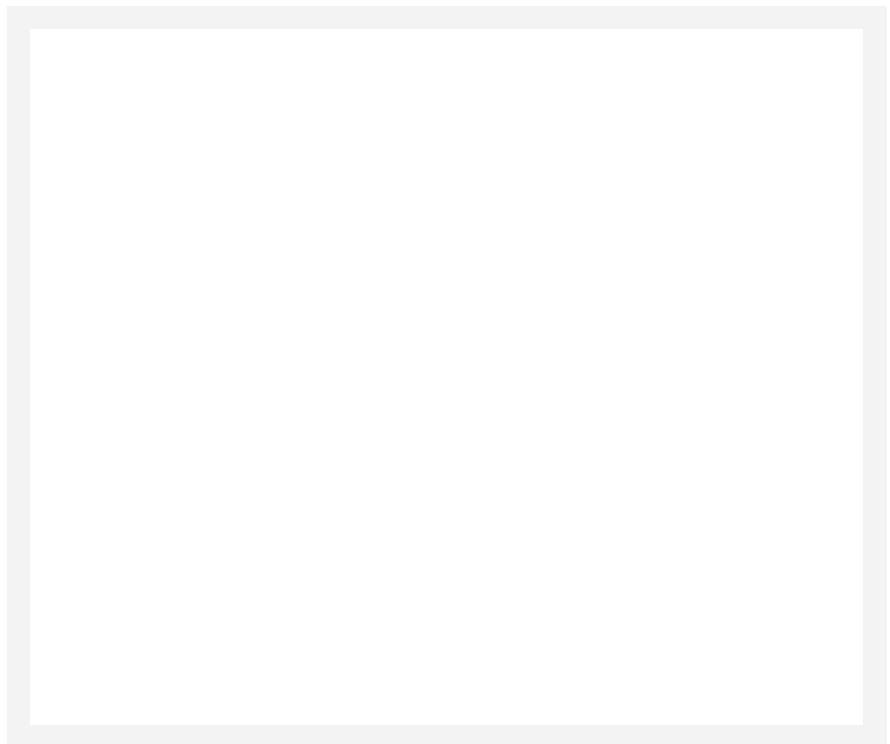
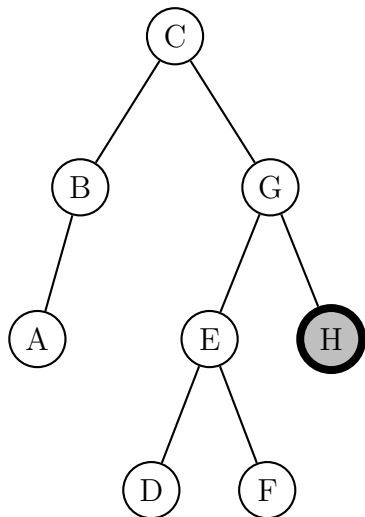
*Aufgabe 4.a.* (9 Punkte)

Lösche jeweils den Schlüssel „**H**“ aus den folgenden Bäumen (*Natürlicher Suchbaum*, *AVL-Baum* und *B-Baum der Ordnung 3*), und gib die resultierenden Bäume an.

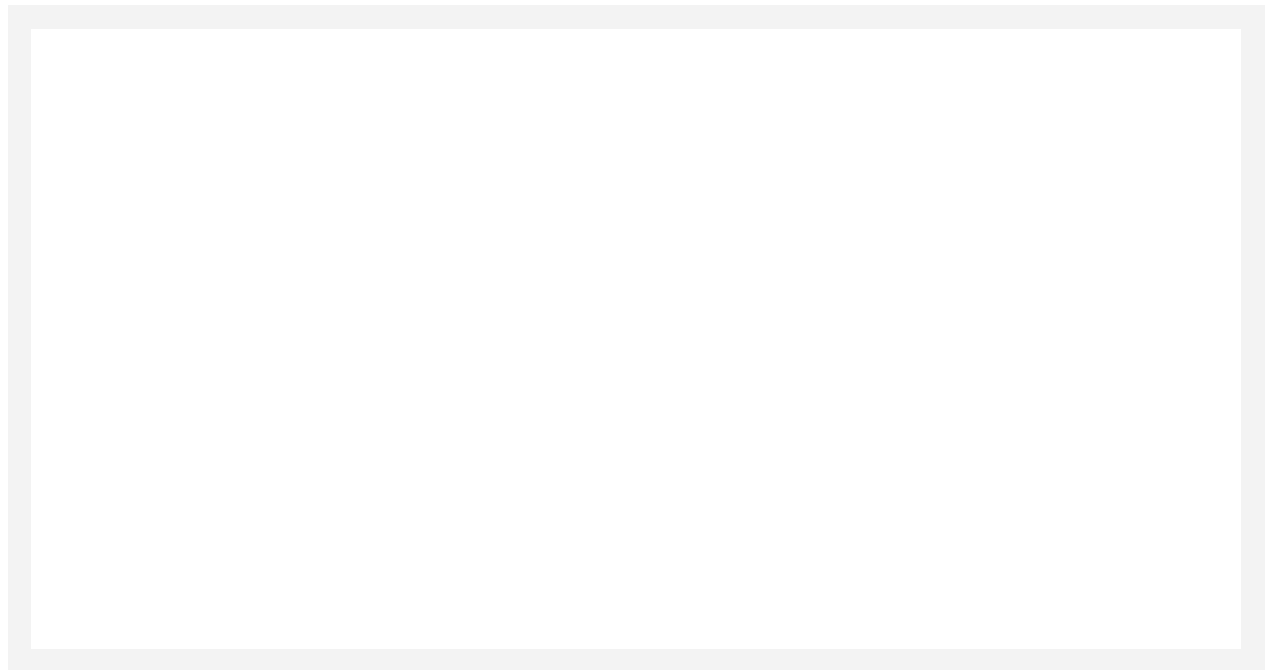
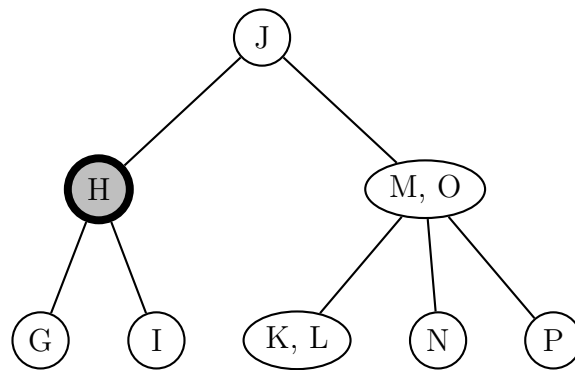
*Natürlicher Suchbaum.*



*AVL-Baum.*



B-Baum der Ordnung 3.



Aufgabe 4.b. (3 Punkte)

Was sind die Worst-Case Laufzeiten (in  $O$ -Notation) für die Löschooperationen auf diesen Datenstrukturen?

Löschen aus natürlichem Suchbaum:  $\Theta(\quad)$

Löschen aus AVL-Baum:  $\Theta(\quad)$

Löschen aus B-Baum der Ordnung  $m$ :  $\Theta(\quad)$

*Aufgabe 4.c. (4 Punkte) Skipliste*

Wie ist die erwartete Laufzeit beim Löschen aus einer Skipliste?

$\Theta(\quad)$

Wie hoch ist das „ $+\infty$ “Element in einer Skipliste?

Wieviel Prozent der Elemente einer Skipliste haben im Durchschnitt einen Eintrag auf der zweituntersten Ebene?

%

Wieviel Prozent der Elemente einer Skipliste haben im Durchschnitt einen Eintrag auf der zweituntersten, aber nicht auf der drittuntersten Ebene?

%

**Thema 5.** (insg. 20 Punkte) *Graphen*

*Aufgabe 5.a.* (4 Punkte) *Definitionen*

Susanne und Angelika finden eine Truhe mit einem Schatz  $S$ . Dieser Schatz besteht aus mehreren Ringen und Halsketten. An jeder Halskette sind zwei Ringe aufgefädelt, jeder Ring kann aber an mehreren Halsketten aufgefädelt sein. Für jedes Paar von Ringen gilt, dass sie jeweils nur durch maximal eine Halskette verbunden sind.

Wenn man einen Ring aus der Truhe entfernen würde, würde man dadurch den gesamten Schatz mitziehen.

Susanne und Angelika erkennen sofort, dass es sich bei dem Schatz  $S$  eigentlich um einen Graphen handelt. Was wissen wir über diesen Graphen?

Die Ringe entsprechen den  des Graphen.

Die Halsketten entsprechen den  des Graphen.

Ist der Graph gerichtet?

Wenn der Schatz  $r$  Ringe enthält, wieviele Halsketten enthält er?

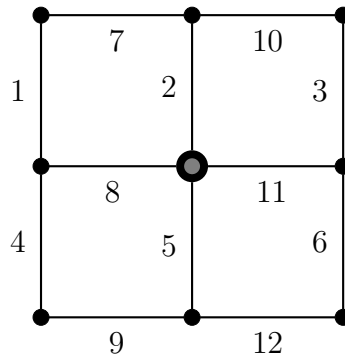
Mindestens:

Maximal:

Als Susanne und Angelika den Schatz mitnehmen wollen, zerfallen einige Halsketten zu Staub, jeder Ring bleibt aber an mindestens einer Kette aufgefädelt. Die übrigen Halsketten teilen Susanne und Angelika so auf, dass sie jeweils gleich viele erhalten, und durch die Aufteilung keinen Ring gemeinsam haben. Angelika jubelt: „Hurra, ich habe genau einen Kreis!“. Susanne antwortet freudig: „Und ich habe genau einen Weg!“. Wer von den beiden besitzt mehr Ringe?

Aufgabe 5.b. (8 Punkte) MST

Wir möchten den Minimalen Spannbaum auf folgendem gewichteten Graphen berechnen:



Gib sowohl für den Algorithmus von Kruskal, als auch für den Algorithmus von Prim die Baumkanten in der Reihenfolge an, in der sie aufgenommen werden. Da die Kantengewichte eindeutig sind, benutze die Gewichte gleichzeitig als Kantenbezeichner. Für den Algorithmus von Prim starte am markierten Knoten in der Mitte.

Reihenfolge bei Kruskal:

Reihenfolge bei Prim:

Aufgabe 5.c. (8 Punkte) DFS

Wenn wir die Knoten die wir bei einem DFS-Durchlauf besuchen so von 1 bis  $n$  nummerieren, dass jeder Knoten wenn er markiert wird die kleinste noch nicht vergebene Nummer erhält, so spricht man von einer *DFS-Nummerierung*.

Der folgende DFS-Algorithmus SPASSMITFARBEN soll alle Baumkanten die von einem Knoten mit gerader DFS-Nummer ausgehen *grün*, die anderen Baumkanten *blau* einfärben. Die Rückwärtskanten sollen *rot* werden, ausser einer der adjazenten Knoten ist mit einer durch 5 teilbaren Zahl nummeriert (in diesem Fall soll die Kante *gelb* werden). Wir nehmen an, dass der gegebene Graph zusammenhängend ist und keine Mehrfachkanten und Schleifen enthält.

Vervollständige den nachfolgenden Code. Es steht dabei die Funktion SETCOLOR(*edge*, *color*) zur Verfügung, mit der man eine gegebene Kante einfärben kann.

**Eingabe:** ungerichteter, zusammenhängender Graph  $G = (V, E)$

**Resultat:** DFS-Nummerierung und richtig eingefärbte Kanten

**var** **bool** *marked*[ $V$ ]

**var** **int** *dfsnum*[ $V$ ] *▷ hierin sollen die DFS-Nummern abgespeichert werden*

**procedure** SPASSMITFARBEN(Node  $v$ )

**for all**  $v \in V$  **do**

*marked*[ $v$ ] := **false**;

**end for**

$v$  := beliebiger Knoten aus  $V$

  DFS-VISIT( $v$ )

**end procedure**

**procedure** DFS-VISIT(Node  $v$ )

*marked*[ $v$ ] := **true**

**for all**  $w \in N(v)$  **do**

**if not** *marked*[ $w$ ] **then**

**else**

**end if**

**end for**

**end procedure**

Raum für Notizen:

Raum für Notizen:

Raum für Notizen: