Lehrstuhl für Praktische Informatik III
Prof. Dr. Guido Moerkotte
B6, 29, Raum C0.10
68131 Mannheim
Telefon: (0621) 181–2582
Email: moer@pi3.informatik.uni-mannheim.de

Norman May
Email: norman@pi3.informatik.uni-mannheim.de

Matthias Brantner
Email: msb@pi3.informatik.uni-mannheim.de

UNIVERSITÄT MANNHEIM

Algorithmen und Datenstrukturen
Wintersemester 2004/05

Vordiplomsklausur Lösungsvorschlag
22. März 2005

Name: .....

Vorname: .....

Matrikelnummer: .....

Studienfach: .....

Hinweise:

1. Prüfen Sie Ihr Klausurexemplar auf Vollständigkeit (20 Seiten).
2. Alle 9 Aufgaben sind zu bearbeiten.
3. Es sind keine Hilfsmittel zugelassen.
4. Die Klausur dauert 100 Minuten.
5. Vermerken Sie auf jedem Lösungsblatt Ihren Namen und Ihre Matrikelnummer.
6. Unterschreiben Sie die Klausur auf dem letzten Blatt.

	maximale Anzahl Punkte	erreichte Anzahl Punkte
Aufgabe 1	20	
Aufgabe 2	9	
Aufgabe 3	5	
Aufgabe 4	10	
Aufgabe 5	11	
Aufgabe 6	4	
Aufgabe 7	13	
Aufgabe 8	16	
Aufgabe 9	12	
	100	

Aufgabe 1 — 20 Punkte

Lösen Sie folgende Rekurrenzen. Geben Sie jeweils die Komplexität in der Θ-Notation an. Gehen Sie davon aus, daß $T(c) = \Theta(1)$ für eine kleine Konstante $c > 0$.

Aufgabe 1 a) — 5 Punkte

$T(n) = 4T(n/2) + n^2$

Lösung

Master-Theorem 2. Fall mit $a = 4$, $b = 2$, $f(n) = n^2$. Es gilt $n^{\log_2 4} = n^2 = \Theta(f(n))$. Daraus folgt:

$$T(n) = \Theta(n^2 \lg n)$$

Aufgabe 1 b) — 5 Punkte

$T(n) = 4T(n/3) + \lg n$

Lösung

Master-Theorem 1. Fall mit $a = 4$, $b = 3$, $f(n) = \lg n$. Wähle $\epsilon = 1$. Dann gilt: $f(n) = O(n^{\log_3(4-1)}) = O(n)$. Daraus folgt:

$$T(n) = \Theta(n^{\log_3 4})$$

Aufgabe 1 c) — 10 Punkte

$T(n) = 4T(n/2) + 2^n$

Lösung

Master-Theorem 3. Fall: Sei $a = 4$, $b = 2$, $f(n) = 2^n$ und $\epsilon = 1$. Es gilt $f(n) = \Omega(n^{log_2 4+1}) = \Omega(n^3)$. Test der Bedingung $af(n/b) \leq cf(n)$ für $c < 1$ und $n > 0$:

$$\begin{aligned} 4 \cdot 2^{n/2} &\leq c \cdot 2^n \\ 4 \cdot 2^{\frac{n}{2}-n} &\leq c \\ 4 \cdot 2^{-\frac{n}{2}} &\leq c \end{aligned}$$

gilt für alle $n > 8$ und $c = 1/2$.
Lösung durch Iterationsmethode:

$$
\begin{aligned}
T(n) &= 4 \cdot T(n/2) + 2^n \\
&= 4 \cdot (4 \cdot T(n/4) + 2^{\frac{n}{2}}) + 2^n \\
&= 4^2 \cdot T(n/4) + 2^{\frac{n}{2}+2} + 2^n \\
&= 4^3 \cdot T(n/8) + 2^{\frac{n}{4}+3} + 2^{\frac{n}{2}+2} + 2^n \\
&\cdots \\
&= 4^{\lg n} \cdot T(1) + \sum_{i=1}^{\lg n - 1} 2^{\frac{n}{2^i}+i+1} + 2^n \\
&= 2^{n+1} + \sum_{i=1}^{\lg n - 1} 2^{\frac{n}{2^i}+i+1}
\end{aligned}
$$

obere Schranke:

$$
\begin{aligned}
T(n) &\leq 2^{n+1} + \lg n \cdot 2^{\frac{n}{2^{\lg n}}+\lg n+1} \\
&= 2^{n+1} + \lg n \cdot 4 \cdot n \\
&= O(2^n)
\end{aligned}
$$

untere Schranke:

$$
\begin{aligned}
T(n) &\geq 2^{n+1} \\
&= \Omega(2^n)
\end{aligned}
$$

Insgesamt also: $T(n) = \Theta(2^n)$

Aufgabe 2 — 9 Punkte

Gegeben sei ein Max-Heap `A` mit `n` Elementen.

Aufgabe 2 a) — 7 Punkte

Schreiben Sie eine Funktion mit der Signatur `HEAP-INCREASE-KEY(A,i,k)`. Die Funktion soll in einen bestehenden Heap den Schlüssel an der Stelle `i` des Arrays `A` durch den neuen größeren Schlüssel `k` ersetzen. Die Laufzeit soll $O(\lg n)$ betragen.

Lösung

HEAP-INCREASE-KEY(A,i,k)
  if $(k < A[i])$
    then error "new key is smaller"
  $A[i] = k$
  while ($i > 1$ and $A[Parent(i)] < A[i]$)

do
  swap $A[i]$ and $A[Parent(i)]$
  $i = Parent(i)$

Aufgabe 2 b) 2 Punkte

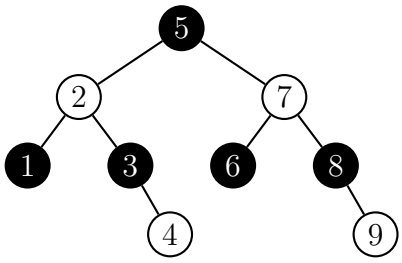
Begründen Sie, warum Ihr Algorithmus die geforderte Laufzeit einhält.

Lösung

Ein Heap mit `n` Elementen hat maximal die Höhe $\lg n$. Der Algorithmus läuft maximal einmal vom Blatt bis zur Wurzel. Dadurch ergibt sich die obere Schranke von $O(\lg n)$.

Aufgabe 3 5 Punkte

Gegeben sei der Rot-Schwarz-Baum aus folgender Abbildung:



Die Knoten mit schwarzem Hintergrund besitzen die Eigenschaft "schwarz", die weißen Knoten die Eigenschaft "rot".

Aufgabe 3 a) 2 Punkte

Geben Sie für den Knoten mit dem Schlüssel `5` den Wert des `Tree-Successors` an.

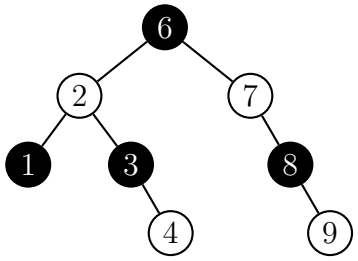
Lösung

6

Aufgabe 3 b) 3 Punkte

Geben Sie den Rot-Schwarz Baum an, der durch Löschen des Knotens mit dem Schlüssel 5 **vor** der Ausführung der Methode `RB-DELETE-FIXUP` entsteht.

Lösung



Aufgabe 4 — 10 Punkte

Konstruieren Sie den deterministischen Musterautomaten für das Muster $P = abaab$ und das Alphabet $\Sigma = a, b, c$.

Lösung

Zustand	a	b	c
0	1	0	0
1	1	2	0
2	3	0	0
3	4	2	0
4	1	5	0
5	3	0	0

Aufgabe 5 — 11 Punkte

Aufgabe 5 a) — 4 Punkte

Es soll eine Zahlenfolge A der Länge n aufsteigend sortiert werden. Kreuzen Sie für die verschiedenen Sortierverfahren jeweils die gültige möglichst engste obere Schranke für den Sortieraufwand im best-case an.

Falls Ihrer Meinung nach keine Angabe möglich ist, kreuzen Sie *keine Angabe möglich* an.

	$O(\lg n)$	$O(n)$	$O(n \lg n)$	$O(n^2)$	keine Angabe möglich
Insertion-Sort	☐	☐	☐	☐	☐
Merge-Sort	☐	☐	☐	☐	☐
Heap-Sort	☐	☐	☐	☐	☐
Quick-Sort	☐	☐	☐	☐	☐
Bucket-Sort	☐	☐	☐	☐	☐
Counting-Sort	☐	☐	☐	☐	☐
Radix-Sort	☐	☐	☐	☐	☐

Lösung

	$O(\lg n)$	$O(n)$	$O(n \lg n)$	$O(n^2)$	keine Angabe möglich
Insertion-Sort	☐	X	☐	☐	☐
Merge-Sort	☐	☐	X	☐	☐
Heap-Sort	☐	☐	X	☐	☐
Quick-Sort	☐	☐	X	☐	☐
Bucket-Sort	☐	X	☐	☐	☐
Counting-Sort	☐	☐	☐	☐	X
Radix-Sort	☐	☐	☐	☐	X

Aufgabe 5 b) 5 Punkte

Sei ein Array P der Länge n gegeben. Die einzelnen Elemente des Arrays enthalten Objekte, die Informationen über Angestellte eines Unternehmens enthalten. Zu den Informationen gehören:

- Personalnummer (ganz Zahl)
- Name (Zeichenkette)
- Gehalt (ganze Zahl im Bereich 1000-5000)

Das Array ist aufsteigend nach Personalnummern sortiert. Angenommen es bestehen keinerlei Restriktionen bezüglich Hauptspeicher.

Wählen Sie ein geeignetes möglichst effizientes Sortierverfahren aus, um die Elemente absteigend nach dem Gehalt zu sortieren. Bei gleichem Gehalt soll die Sortierung der Personalnummern erhalten bleiben. Begründen Sie Ihre Wahl.

Lösung

Man verwendet Radix oder Counting Sort.

- Kein in-place Sortieralgorithmus notwendig, da Hauptspeicher keine Rolle spielt.
- Der Sortieralgorithmus muss aber stabil sein, da die Sortierung der Personalnummern erhalten bleiben soll.
- Für Radix-Sort ist der Wertebereich fest vorgegeben (Zahlen mit genau 4 Ziffern).
- Für Counting-Sort ist der Wertebereich auch fest vorgegeben bzw. beschränkt.
- Sortierung in linearer Zeit.

Aufgabe 5 c) 2 Punkte

Sei das Array aus dem vorigen Aufgabenteil gegeben.

Angenommen das Array P passt gerade noch in den Hauptspeicher. Welchen möglichst effizienten Hauptspeicher-Sortieralgorithmus verwendet man, wenn die Elemente absteigend nach dem Gehalt sortiert werden sollen? Die Sortierung der Personalnummern braucht, im Gegensatz zur vorigen Teilaufgabe, nicht mehr erhalten bleiben. Begründen Sie Ihre Wahl.

Lösung

Man verwendet Quick-Sort oder Heap-Sort, da beide in-place sortieren.

Aufgabe 6 4 Punkte

Kreuzen Sie bei den folgenden Fragen die richtige Antwort an.

	Ja	Nein
Für jedes Optimierungsproblem, dessen optimale Lösung mit einem Greedy-Algorithmus gefunden werden kann, kann auch dynamisches Programmieren verwendet werden, um eine optimale Lösung zu finden.		
Wenn dynamisches Programmieren eine optimale Lösung für ein Optimierungsproblem findet, findet auch Memoization eine optimale Lösung für dieses Optimierungsproblem.		
Für jedes Optimierungsproblem, dessen optimale Lösung mit einem Greedy-Algorithmus gefunden werden kann, kann auch mit dem Algorithmus "`GREEDY`" für gewichtete Matroiden eine optimale Lösung gefunden werden.		
Dynamisches Programmieren ist sinnvoll einsetzbar, wenn keine gemeinsamen Teilprobleme existieren.		

Lösung

Lösung:

- Ja.
- Ja.
- Nein.
- Nein.

Aufgabe 7 — 13 Punkte

Wir betrachten folgende Variation des 0-1-Knappsackproblems aus der Vorlesung: Gegeben ist eine ganze Zahl $W > 0$ und eine Menge S von Gegenständen mit $|S| = n$. Der i-te Gegenstand k_i hat ein ganzzahliges positives Gewicht g_i. Finde eine Menge $s \subseteq S$ von Gegenständen, die $(\sum_{k_i \in s} g_i) \leq W$ maximiert. D.h. alle Gegenstände haben denselben Wert pro Gewicht.

Aufgabe 7 a) — 11 Punkte

Geben Sie einen Algorithmus an, der eine optimale Lösung für das Problem findet. Ihre Lösung muss nicht notwendigerweise effizient bezüglich der Laufzeit sein.

Lösung

In beiden Algorithmen ist S ein Array der Größe n, das die Gewichte der Elemente enthält.

1. Algorithmus:

```
ALGO1(S, K)
  weight = 0
  result = ∅
  for each  s ⊆ S
    weight_n = Σ_{k_i ∈ s} S[i]
    if weight_n ≤ K and  weight_n > weight then
      result = s
      weight = weight_n
      if weight_n = K then
        return result
  return result
```

2. Algorithmus:

Idee: Finde für alle Gewichte $w \leq W$ einen Menge $s \subseteq S$, die das Gewicht w vollständig ausschöpft.

Wenn das Problem für $(S \setminus k_i, w)$ lösbar ist, dann ist es auch für (S, w) lösbar — wenn k_i nicht benutzt wird. Ansonsten, wenn das Problem für $(S \setminus k_i, w - S[n])$ lösbar ist, ist es unter Verwendung des Gegenstands k_i lösbar. Mit Hilfe einer Tabellenstruktur werden die dabei auftretenden gemeinsamen Teilprobleme nur einmal gelöst.

```
ALGO2(S, W)
  P[0,0].exist = true
  for w := 1 to W do
    P[0,w].exist := false
  for  i := 1 to n do
    for  w := 0 to W do
      P[i,w].exist := false // initialize
      if  P[i-1, w].exist then
        P[i,w].exist := true
        P[i,w].belong := false
      else if  w − S[i] ≥ 0 then
        if  P[i − 1, w − S[i]].exist then
          P[i,w].exist := true
          P[i,w].belong := false
  for w := W down to 0 do
    if P[n,w].exist = true then
      return w
```

Die Ausgabe des Algorithmus ist ein 2-dimensionales Array, in dem $P[i,w].exist = true$ bedeutet, daß es eine Lösung für das Problem gibt, in dem die ersten i Gegenstände benutzt werden und das Gewicht w ist. $P[i,w].belong = true$, wenn der i-te Gegenstand zur Lösung gehört. Der Algorithmus liefert das größte gefundene Gewicht, das kleiner als W ist.

Aufgabe 7 b) 2 Punkte

Leiten Sie aus Ihrem Algorithmus dessen Laufzeit ab, und geben Sie die Laufzeit Ihres Algorithmus mit Hilfe der O-Notation an.

Lösung

Laufzeit:

1. Algorithmus: $O(n2^n)$, da es 2^n mögliche Teilmengen von S gibt und die Berechnung des Gewichts einer Menge s in $O(n)$ ausgeführt werden kann.

2. Algorithmus: $O(nW)$, die äußere der 2 verschachtelten Schleifen wird W-mal ausgeführt. In jeder dieser Schleifen wird die innere Schleife maximal n-mal ausgeführt. Jede Ausführung der inneren Schleife erfordert konstanten Aufwand.

Aufgabe 8 — 16 Punkte

Gegeben sei der gerichtete Graph (G) aus Abbildung 1.

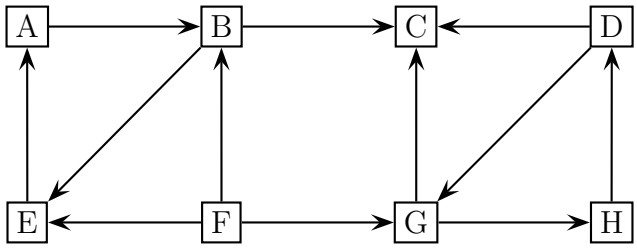


Abbildung 1: Graph G

Aufgabe 8 a) — 8 Punkte

Führen Sie eine Tiefensuche auf dem gegebenen Graphen (G) aus und geben Sie die dazugehörigen Start- und Endzeiten für jeden Knoten an. Tragen Sie die Start- und Endzeiten in die unten stehende Tabelle ein.

Wenn mehrere Knoten besucht werden können, besuchen Sie diese in alphabetischer Reihenfolge.

Knoten	Startzeit	Endzeit
A		
B		
C		
D		
E		
F		
G		
H		

Lösung

Knoten	Startzeit	Endzeit
A	1	8
B	2	7
C	3	4
D	9	14
E	5	6
F	15	16
G	10	13
H	11	12

Aufgabe 8 b) 8 Punkte

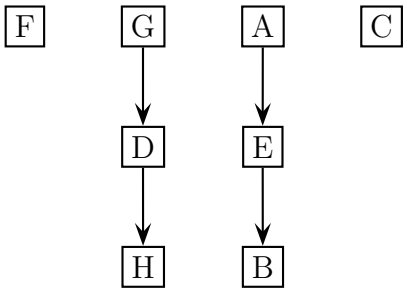
Führen Sie den Algorithmus zum Finden von *Starken Zusammenhangskomponenten* auf dem Graphen (G) aus Abbildung 1 aus. Geben Sie den entstehenden *Komponentengraph* an. Geben Sie weiterhin einen nachvollziehbaren Lösungsweg (einzelne Lösungsschritte) an.

Lösung

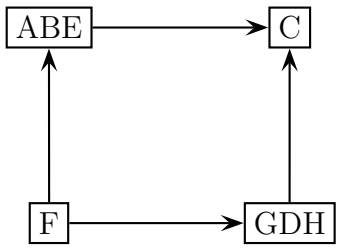
Start- und Endzeiten von der Durchführung von $DFS(G^T)$.

Knoten	Startzeit	Endzeit
A	9	14
B	11	12
C	15	16
D	3	8
E	10	13
F	1	2
G	5	6
H	4	7

Tiefensuchewald:



Komponentengraph:



Aufgabe 9 12 Punkte

Sei ein gerichteter, gewichteter, zusammenhängender Graph $G = (E, V)$ gegeben.

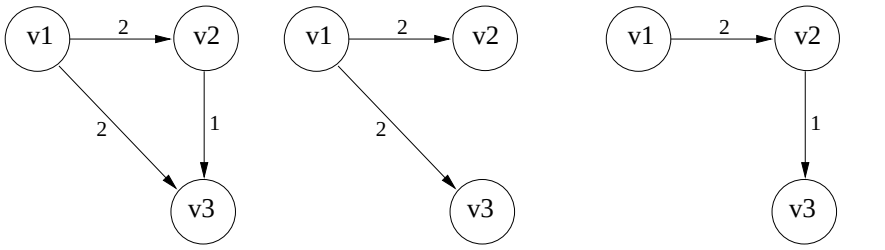
Beweisen Sie für folgende Aussagen die Korrektheit, oder geben Sie ein Gegenbeispiel. Gehen Sie in jeder Teilaufgabe davon aus, daß in dem Graph alle Knoten vom Startknoten der Breiten- und Tiefensuche aus erreichbar sind.

Aufgabe 9 a) 4 Punkte

Der Shortest-Path-Tree, der durch den Single-Source-Shortest-Path-Algorithmus von Dijkstra berechnet wird, ist ein minimaler Spannbaum.

Lösung

Falsch: Gegenbeispiel ...



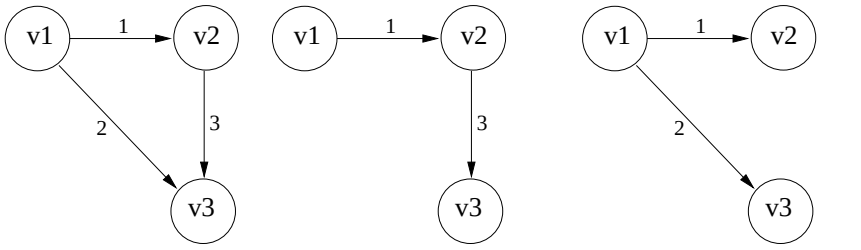
Beispielgraph Shortest-Path-Tree Minimal-Spanning-Tree

Aufgabe 9 b) 4 Punkte

Wenn bei der Tiefensuche Nachbarknoten nach aufsteigendem Kantengewicht der verbindenden Kanten besucht werden, ist der entstehende Tiefensuchbaum ein minimaler Spannbaum.

Lösung

Nein, Gegenbeispiel:



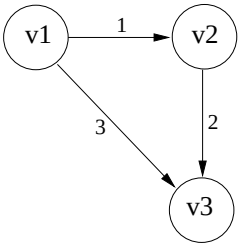
Beispielgraph Tiefensuchbaum Minimal-Spanning-Tree

Aufgabe 9 c) 4 Punkte

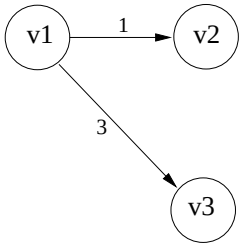
Wenn bei der Breitensuche Nachbarknoten nach aufsteigenden Kantengewicht der verbindenden Kanten besucht werden, ist der entstehende Breitensuchbaum ein minimaler Spannbaum.

Lösung

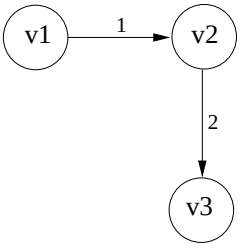
Nein, Gegenbeispiel:



Beispielgraph



Breitensuchbaum



Minimal-Spanning-Tree