Lehrstuhl für Informatik VII
Rheinisch-Westfälische
Technische Hochschule Aachen
Prof. Dr. W. Thomas
Dipl.-Inform. C. Hirsch

SS 99

13. April 1999

1. Übung

Vorlesung Datenstrukturen und Algorithmen

Abgabe: am Montag, 19.4.1999 (in den Übungen)

Aufgabe 1: 3 Punkte

Begründen Sie anhand der Definition von $O(f(n))$ folgende Aussagen:

- Die konstante Funktion 17 ist $O(1)$
- $n(n+1)/2$ ist $O(n^2)$
- $\max(n^3, 10n^2)$ ist $O(n^3)$

Aufgabe 2: 1+2+1 Punkte

Betrachten Sie folgende drei (in Pascal notierten) Prozeduren und geben Sie mit der O-Notation jeweils die Laufzeit als Funktion von n an. Begründen Sie Ihre Behauptung.

```
procedure matmult ( n : integer );
  var
    i, j, k : integer;
  begin
    for i := 1 to n do
      for j := 1 to n do begin
        C[i, j] := 0;
        for k := 1 to n do
          C[i, j] := C[i, j] + A[i, k] * B[k, j]
      end
  end
```

```
procedure mystery ( n : integer );
  var
    i, j, k : integer;
  begin
    for i := 1 to n - 1 do
      for j := i + 1 to n do
        for k := 1 to j do
          (eine Anweisung mit Zeitaufwand O(1))
  end
```

```
procedure veryodd ( n : integer );
  var
    i, j, x, y : integer;
  begin
    for i := 1 to n do
      if odd(i) then begin
        for j := i to n do
          x := x + 1;
      else
        for j := 1 to i do
          y := y + 1
      end
  end
```

Lehrstuhl für Informatik VII
Rheinisch-Westfälische
Technische Hochschule Aachen
Prof. Dr. W. Thomas
Dipl.-Inform. C. Hirsch

SS 99

15. April 1999

2. Übung

Vorlesung Datenstrukturen und Algorithmen

`Abgabe: am Freitag, 23.4.1999`

Aufgabe 3: binäre Suche

3 Punkte

Sei $A[1..n]$ ein aufsteigend sortiertes Feld von ganzen Zahlen,
d.h. für alle $i, j \in \{1, \ldots, n\}$ gilt: wenn $i < j$ so $A[i] \leq A[j]$.

a) Geben Sie in Form von Pseudo-Code einen Algorithmus an, der für eine gegebene Zahl k entscheidet, ob k in $A[1..n]$ vorkommt.

   Benutzen Sie dabei das Verfahren der binären Suche.

b) Begründen Sie, daß dieser Algorithmus eine Komplexität von $\mathcal{O}(\log n)$ besitzt.

Der Aufgabenteil c) braucht nur von Studierenden im ersten Semester bearbeitet zu werden.

c) Formulieren Sie den Algorithmus aus a) als Modula-3 oder C Programm.

Aufgabe 4: Komplexität von Funktionen

4 Punkte

Sortieren Sie die folgenden Funktionen anhand der in der Vorlesung vorgestellten Kriterien nach ihrer Komplexität, so daß $f(n)$ in der Liste vor $g(n)$ kommt, wenn $\mathcal{O}(f(n)) \subseteq \mathcal{O}(g(n))$. Wann gilt $\mathcal{O}(f(n)) \subsetneq \mathcal{O}(g(n))$? Begründen Sie ihre Antwort.

$$n^2 \qquad n! \qquad 2^{(4+\log n)} \qquad n^n \qquad n \qquad n \cdot 2^n \qquad \left(\frac{6}{5}\right)^n \qquad 2^n \qquad (\log n)^{\log n}$$

Aufgabe 5: Rechnen mit Komplexitäten

4 Punkte

Beweisen oder Widerlegen Sie:

	übliche Schreibweise	formal exakte Schreibweise
a)	$\mathcal{O}(f(n) + g(n)) = f(n) + \mathcal{O}(g(n))$	$\mathcal{O}(f + g) \subseteq \{f + \tilde{g} \mid \tilde{g} \in \mathcal{O}(g)\}$
b)	$f(n) + \mathcal{O}(g(n)) = \mathcal{O}(f(n) + g(n))$	$\{f + \tilde{g} \mid \tilde{g} \in \mathcal{O}(g)\} \subseteq \mathcal{O}(f + g)$
c)	$f(n) + 45n^2 = \mathcal{O}(f(n))$	$f(n) + 45n^2 \in \mathcal{O}(f(n))$
d)	$f(n) \cdot \mathcal{O}(g(n)) = \mathcal{O}(f(n) \cdot g(n))$	$\{f \cdot \tilde{g} \mid \tilde{g} \in \mathcal{O}(g)$

Lehrstuhl für Informatik VII
Rheinisch-Westfälische Technische Hochschule Aachen
http://www-i7.informatik.rwth-aachen.de/

Prof. Dr. W. Thomas
Dipl.-Inform. C. Hirsch

SS99
22. April 1999

3. Übung

Vorlesung Datenstrukturen und Algorithmen

Abgabe: am Freitag, 30.4.1999

Aufgabe 4: Dynamisches Programmieren — 3 Punkte

Wenden Sie das aus der Vorlesung bekannte Verfahren manuell an, um die optimale Reihenfolge für die Auswertung einer Kette von Matrix-Multiplikationen mit Dimensionsvektor $\langle 5, 10, 3, 12, 5, 50, 6\rangle$ zu ermitteln.

Aufgabe 5: Inversionen — 4 Punkte

Sei $A[1..n]$ ein Array, dessen Inhalt aus n *verschiedenen* Zahlen besteht. Sind i, j zwei Indizes mit $1 \leq i < j \leq n$ und gilt $A[i] > A[j]$, so nennt man das Paar (i, j) eine *Inversion* in A.

a) Nennen Sie die fünf Inversionen des Arrays $[2, 3, 8, 6, 1]$.

b) Welches Array dessen Einträge die Menge $\{1, 2, \ldots, n\}$ bilden hat die meisten Inversionen, und wieviele sind es?

c) Geben sie einen Algorithmus an, der mit einer worst-case Zeitkomplexität von $\mathcal{O}(n \log n)$ die Anzahl der Inversionen in einem Array dessen Einträge die Menge $\{1, 2, \ldots, n\}$ bilden bestimmt.

   Hinweis: Modifizieren Sie Mergesort.

Aufgabe 6: Sortierverfahren — 6 Punkte

Implementieren Sie "Selectionsort" und "Mergesort" jeweils als C oder Modula-3 Programm. Dabei soll das zu sortierende Array zunächst statisch initialisiert werden, später sollen die Daten aus einer Datei eingelesen werden. Bei den zu sortierenden Werten handelt es sich um positive ganze Zahlen im Wertebereich $0 \ldots 1048575$.

Bauen Sie einen Zähler für die Anzahl der Vergleichsoperationen im Verlauf der Sortierung ein.

Der Source-Code ist zusammen mit den Laufzeitprotokollen an Ihren HiWi zu mailen. Die entsprechende E-Mail Adresse erhalten Sie in der Übungsstunde.

Für den ersten Testlauf lassen Sie folgendes Array sortieren und protokollieren Sie die Ausgabe:

$$10, 39, 19, 5, 20, 11, 72, 9, 65, 90, 55.$$

Für die weiteren Läufe erhalten Sie die Beispieldateien auf den WWW-Seiten des Lehrstuhls. Führen Sie mit den gegebenen Dateien `zahlen.100`, `zahlen.1000` und `zahlen.10000` Testläufe

durch und kommentieren Sie das Laufzeitverhalten der beiden Algorithmen anhand des eingebauten Zählers. Welche anderen Maße für das Laufzeitverhalten sind sinnvoll?

Zur Kontrolle geben Sie jeweils die mittleren 10 Zahlen des sortierten Arrays aus.

Hinweise: *Die Dateien mit den zu sortierenden Zahlen enthalten jeweils in einer Zeile eine Zahl. Die erste Zeile enthält die Anzahl der folgenden Zeilen und soll* nicht *mit sortiert zu werden.*

In C benutzen Sie die Funktionen `fopen()` *und* `fclose()` *zum Öffnen bzw. Schließen der Datei,* `fread()`, `fscanf()` *oder* `fgetc()` *zum Einlesen der Daten und ggf.* `atoi()` *zum Konvertieren von Strings in Integers. Die benötigten Header-Files sind* `stdio.h` *und* `stdlib.h`*.*

Definieren Sie sich mit `int *array;` *einen Zeiger, um die Basisadresse des Arrays aufzunehmen. Nach einlesen der ersten Zeile in eine* `int` *Variable n können Sie mit*
`array = (int *)malloc(sizeof(int) * n);`
den Speicher für das Array anlegen. In einer Schleife können dann die anderen Zahlen eingelesen werden:
`for (i = 0; i < n; i++) fscanf(f, "%d\n", array + i);`
wobei `f` *als* `FILE *f;` *definiert und durch* `fopen()` *auf einen gültigen File-Deskriptor verweist.*

In Modula-3 benutzen Sie die aus der Vorlesung Informatik 1 bekannten Mechanismen zum Lesen einer Datei.

Lehrstuhl für Informatik VII
Rheinisch-Westfälische Technische Hochschule Aachen
http://www-i7.informatik.rwth-aachen.de/

Prof. Dr. W. Thomas
Dipl.-Inform. C. Hirsch

SS99
29. April 1999

4. Übung

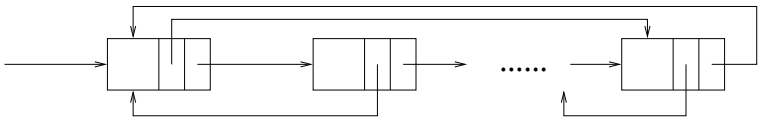
Vorlesung Datenstrukturen und Algorithmen

Abgabe: am Freitag, 7.5.1999

Aufgabe 9: Zyklisch verkettete Liste

6 Punkte

Bei einer zyklischen Liste zeigt ein Pointer im letzten Listenelement auf das erste Listenelement. Die Liste sei darüberhinaus doppelt verkettet, jeder Knoten enthält also Zeiger auf seinen Vorgänger und seinen Nachfolger. Die Liste selbst wird durch einen Zeiger auf das erste Element repräsentiert.



Die Listenknoten sind hier Records des Typs `Node`:

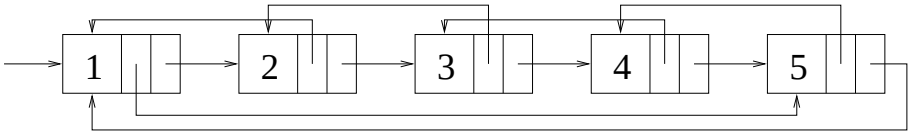
```
TYPE
  NodePointer = REF Node;
  Node = RECORD
    Info: CARDINAL;
    RightNode: NodePointer; (* points in clock direction *)
    LeftNode:  NodePointer; (* points in counter-clock direction *)
  END;
```

Implementieren Sie die folgenden Funktionen in C bzw. Modula-3:

a) Neues Element nach dem aktuellen Element einfügen (das neue Element ist dann das aktuelle):
`PROCEDURE Insert (VAR cList: NodePointer; info: CARDINAL);`

b) Zyklische Liste einen Schritt im Uhrzeigersinn drehen:
`PROCEDURE Clockwise (VAR cList: NodePointer);`

Testen Sie Ihr Programm anhand der in der folgenden Abbildung skizzierten Liste (die Variable `X` vom Typ `NodePointer` zeige auf das aktuelle Element).



Geben Sie den Listenzustand nach folgenden Aufrufen an:

a) `Insert(X, 3);`

b) `Clockwise(X);`

Aufgabe 10: Vertauschen von Listenelementen 2 Punkte

Geben Sie eine Prozedur an, die bei gegebenem Zeiger auf das erste Element einer einfach verketteten Liste sowie gegebener Zahl p das p-te und $(p+1)$-te Element der Liste vertauscht.

Aufgabe 11: Spezialfall Suchen 3 Punkte

Sei `A[1..n]` ein Feld ganzer Zahlen mit `A[i] < A[j]` für $i < j$. Geben Sie mit entsprechender Begründung in Pseudocode ein Verfahren nach dem "Divide and Conquer" Ansatz an, welches in Zeit $\mathcal{O}(\log n)$ testet, ob ein $i \in \{1, \ldots n\}$ mit `A[i] = i` existiert.

Aufgabe 12: Dynamisches Programmieren 3 Punkte

Ergänzen Sie den Algorithmus der Vorlesung zur optimalen Multiplikation von Matrizenketten $M_1, \ldots, M_n$ um eine weitere Ausgabe $k_{(i,j)}$ mit jedem Wert m_{ij}, so daß die Klammerung

$$(M_i \cdot \cdots \cdot M_{k_{(i,j)}}) \cdot (M_{k_{(i,j)}+1} \cdot \cdots \cdot M_j)$$

auf den optimalen Wert m_{ij} führt.

Ergänzen Sie entsprechend die Tabelle zum Beispiel aus Übung 3 Aufgabe 4 und geben Sie die optimale Klammerung an.

Lehrstuhl für Informatik VII
Rheinisch-Westfälische Technische Hochschule Aachen
http://www-i7.informatik.rwth-aachen.de/

Prof. Dr. W. Thomas
Dipl.-Inform. C. Hirsch

SS99
5. Mai 1999

5. Übung

Vorlesung Datenstrukturen und Algorithmen

Abgabe: am Freitag, 14.5.1999

Aufgabe 13: Bäume

4 Punkte

Wir betrachten den arithmetischen Term

$$t = (a - \sqrt{t}) + (\sqrt{a * b} + c)$$

mit Konstanten a, b, c, einstelligem Funktionssymbol $\sqrt{}$ und zweistelligen Funktionssymbolen $+, -, *$.

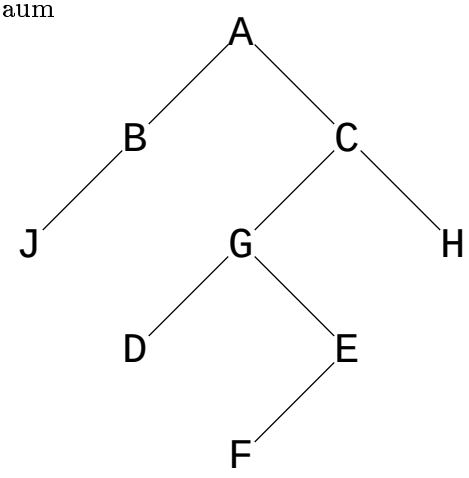
Stellen Sie t dar

a) als Baum

b) in polnischer Notation (preorder)

c) in umgekehrt polnischer Notation (postorder)

Weiter betrachten wir den Baum



Geben Sie die Nummerierung gemäß der Durchlaufstrategien Preorder, Inorder, Postorder an (d.h. ordnen Sie jeweils dem Knotenwert die entsprechende Ordnungszahl zu).

Aufgabe 14: Bubblesort 5 Punkte

Das Verfahren Bubblesort sortiert ein Feld $A[1..n]$ so, daß nach dem i-ten Durchlauf die i größten Einträge an der richtigen Stelle sind. Im i-ten Durchlauf wird also Stelle $n+1-i$ richtig besetzt. Dazu werden jeweils für $j = 1, \ldots, n-i$ die Elemente $A[j]$ und $A[j+1]$ verglichen und im Falle $A[j] > A[j+1]$ vertauscht. Es gibt also eine Schleife über i und eine über j.

a) Führen Sie das Verfahren am Beispiel

3 5 1 7 10 4

manuell mit Notation der sich nach Vertauschungsschritten ergebenden Zwischenergebnisse durch.

b) Schreiben Sie eine entsprechende Prozedur in Modula-3 oder C.

c) Schätzen Sie mit Begründung die Laufzeit anhand der Zahl der nötigen Vergleichsoperationen ab, d.h. finden Sie eine geeignete Schranke der Form $\mathcal{O}(f(n))$.

Können Sie auch $\Theta(f(n))$ garantieren?

Wie sieht es aus, wenn man die Zahl der Vertauschungen betrachtet?

Aufgabe 15: Heap 5 Punkte

Ein Feld $A[1..n]$ von ganzen Zahlen hat die *Heap-Eigenschaft*, falls gilt:

$$A[i] \geq A[2 \cdot i] \text{ und } A[i] \geq A[2 \cdot i + 1].$$

Implementieren Sie in Modula-3 oder C eine Prozedur `CreateHeap (VAR field: ARRAY OF INTEGER)` (C Prototyp: `void CreateHeap(int *field, int size);`), die das Feld `field` so umsortiert, daß `field` die Heap-Eigenschaft besitzt. Testen Sie die Prozedur am Beispiel des Feldes

$$[2, 12, 5, 10, 8, 7, 9, 6, 15, 3, 11, 13, 4, 17, 14]$$

und fertigen Sie ein Laufzeitprotokoll an.

Lehrstuhl für Informatik VII
Rheinisch-Westfälische Technische Hochschule Aachen
http://www-i7.informatik.rwth-aachen.de/

Prof. Dr. W. Thomas
Dipl.-Inform. C. Hirsch

SS99
12. Mai 1999

6. Übung

Vorlesung Datenstrukturen und Algorithmen

Abgabe: am Freitag, 21.5.1999

Aufgabe 16: Quicksort 4 Punkte

Untersuchen Sie Quicksort gemäß Vorlesung für die Eingabe

- von n gleichen Elementen
- der Folge 1010101

Geben Sie die Anzahl der Vergleiche und typische Zwischenergebnisse an.

Aufgabe 17: Stabilität 4 Punkte

Ein Sortieralgorithmus heißt *stabil*, wenn die relative Ordnung der Elemente mit gleichem Schlüssel durch einen Sortiervorgang nicht verändert wird. D.h.: Wird durch das Verfahren bei Sortieren eines Arrays $A[1..n]$ die Permutation $i_1, \ldots, i_n$ der Positionen $1, \ldots, n$ erzeugt, so muß für $A[k] = A[l]$ mit $k < l$ stets $i_k < i_l$ gelten.

Untersuchen Sie (mit Begründung bzw. Gegenbeispiel) Bubblesort, Shellsort (vgl. Aufgabe 19) und Quicksort auf Stabilität.

Aufgabe 18: Mindestkomplexität 3 Punkte

Zeigen Sie, daß jeder Sortieralgorithmus, der bei gegebenem Feld der Größe n nur dadurch Feldelemente bewegt, daß er benachbarte Elemente vertauscht, zumindest einen (worst-case) Zeitaufwand von $\Omega(n^2)$ hat.

— Aufgabe 19 umseitig —

Aufgabe 19: Shellsort 3 Punkte

Ein von Bubblesort abgeleitetes Sortierverfahren ist Shellsort. Zur Verbesserung des Laufzeitverhaltens werden Elemente auch über größere Entfernungen vertauscht. Eine entsprechende Prozedur lautet:

```
PROCEDURE ShellSort (Var a: ARRAY[1..n] OF ItemType)
VAR i, j, incr: CARDINAL;
BEGIN
  incr := n DIV 2; (* initiale Schrittweite *)
  WHILE (incr > 0) DO
    FOR i := incr + 1 TO n DO
      j := i - incr;
      WHILE (j > 0) DO
        IF (a[j] > a[j + incr]) THEN
          vertausche(a[j], a[j + incr]);
          j := j - incr;
        ELSE
          j := 0;
        END; (* IF *)
      END; (* WHILE *)
    END; (* FOR *)
    incr := incr DIV 2;
  END; (* WHILE *)
END Shellsort;
```

Untersuchen Sie manuell den Lauf von Shellsort auf der Eingabe $[1, 7, 3, 2, 0, 5, 0, 8]$. Geben Sie die Zwischenergebnisse nach allen Vertauschungen an.

Lehrstuhl für Informatik VII
Rheinisch-Westfälische Technische Hochschule Aachen
http://www-i7.informatik.rwth-aachen.de/

Prof. Dr. W. Thomas
Dipl.-Inform. C. Hirsch

SS99
20. Mai 1999

7. Übung

Vorlesung Datenstrukturen und Algorithmen

Abgabe: am Freitag, 4.6.1999

Aufgabe 20: InsertionSort

4 Punkte

InsertionSort benutzt folgenden Ansatz für ein Eingabefeld $A[1..n]$. Nach dem i-ten Schritt ist das Feld $A[1] \dots A[i+1]$ sortiert (jedoch *nicht* notwendig schon mit den $i+1$ kleinsten Elementen). Im $(i+1) - ten$ Schritt wird das Element $A[i+2]$ an die richtige Position in die Folge der Elemente $A[1] \dots A[i+1]$ gebracht.

```
PROCEDURE insertionsort;
  VAR i, j, v : INTEGER;
  BEGIN
  FOR i := 2 TO n DO
    BEGIN
    v := a[i]; j := i;
    WHILE a[j-1] > v DO
      BEGIN a[j] := a[j-1]; j := j - 1; END;
    a[j] := v;
    END
  END;
```

Geben Sie für die angegebenen Zahlenfolgen die exakte Anzahl der Bewegungen für die Sortierverfahren *InsertionSort* bzw. *SelectionSort* in Abhängigkeit von N an. Begründen Sie Ihre Antworten.

a) $\{$N, 1, 2, 3, 4, ... , $N-1\}$

b) $\{2, 3, 4, 5, \dots, N, 1\}$

c) $\{1, \frac{N}{2}+1, 2, \frac{N}{2}+2, \dots, \frac{N}{2}, N\}$ (für gerade N)

Aufgabe 21: 3-Wert-Sortieren

4 Punkte

In einem Array seien Records mit den Schlüsselwerten $\{$*rot, grün, blau*$\}$ enthalten. Es soll eine Folge dieser Records hergestellt werden, in der am Anfang alle roten, dann alle grünen und zuletzt alle blauen Schlüssel vorkommen. Als Operationen seien nur das Feststellen eines Schlüsselwertes und das Vertauschen zweier Records erlaubt. Geben Sie einen Algorithmus an,

der das Array *in situ*, d.h. ohne zusätzlichen Speicherplatz[1], mit Laufzeitkomplexität $O(n)$ sortiert.

Zeigen Sie, daß Ihr Algorithmus in $O(n)$ Schritten terminiert und das Array dann tatsächlich sortiert ist.

Aufgabe 22: Paarsuche 4 Punkte

Gegeben sei ein Array $A[1..n]$ von natürlichen Zahlen und eine natürliche Zahl x. Gesucht wird ein Paar (a, b) von Elementen aus A, für das gilt: $x = a + b$.

Geben Sie einen Algorithmus an (Pseudocode), der diese Aufgabe in $O(n)$ erfüllt. Gehen Sie dabei von einem bereits sortierten Array aus. Der Algorithmus soll ein geeignetes Paar (a, b) zurückliefern oder aber eine Fehlermeldung ausgeben.

Aufgabe 23: Lexikographisches Sortieren 5 Punkte

Wir betrachten die Aufgabe, eine Folge von Zahlenfolgen variabler Länge, aber mit Elementen aus einem festen Reservoir, $\{0, \ldots, m-1\}$, lexikographisch zu sortieren.

Handelt es sich um Zahleneinträge aus $\{0, \ldots, 25\}$ (entsprechend den Buchstaben A, ... ,Z), so entspricht die lexikographische Anordnung der Anordnung von Worten im Lexikon: AFFE ¡ GIRAFFE ¡ ZOO.

Formal gilt $(a_1, \ldots, a_k) < (b_1, \ldots, b_l)$ gdw. **eine** der beiden Bedingungen

1. für ein $i \leq \min(k, l)$ gilt: $a_j = b_j$ für $1 \leq j < i$ und $a_i < b_i$
2. $(b_1, \ldots, b_l)$ ist eine echte Verlängerung von $(a_1, \ldots, a_k)$.

erfüllt ist.

Entwerfen Sie durch Modifikation von RadixSort einen Algorithmus (erst in Pseudocode, dann als Implementierung in Modula-3 oder C), der eine Folge von Zahlenfolgen lexikographisch ordnet.

Hierbei bestimmt man für Folgen $A_1 = (a_1^1, \ldots, a_{n_1}^1), \ldots, A_r = (a_1^r, \ldots, a_{n_r}^r)$ zunächst die maximale auftretende Länge $l_{max} = \max_{i=1\ldots r} n_i$ und sortiert so, daß nach i Schritten die Folgen mit Länge $\geq l_{max} - i + 1$ nach ihren Einträgen ab Stelle $l_{max} - i + 1$ richtig sortiert sind.

Der Zeitaufwand sollte $\mathcal{O}(n_1 + \cdots + n_r + m)$ betragen.

Für die Implementierung werden die Zahlenfolgen in einem Array (der Größe r) von Arrays (der Größe l_{max}), also in einem zwei-dimensionalem Array, repräsentiert. Ein zusätzliches Array (der Größe r) gibt zu jeder Zahlenfolge die Länge an.

[1] Für das Vertauschen zweier Records steht Speicher der Größe eines Records zur Verfügung.

Lehrstuhl für Informatik VII
Rheinisch-Westfälische Technische Hochschule Aachen
http://www-i7.informatik.rwth-aachen.de/

Prof. Dr. W. Thomas
Dipl.-Inform. C. Hirsch

SS99
4. Juni 1999

8. Übung

Vorlesung Datenstrukturen und Algorithmen

Abgabe: am Freitag, 11.6.1999

Aufgabe 24: Hashing – Reihenfolge

4 Punkte

Gegeben sei eine Hashtabelle der Größe 7 mit der Belegung

0	1	2	3	4	5	6
1	164	8	21	73	22	89

sowie die Hashfunktion $h(k) = (\text{Quersumme von } k) \bmod 7$. Als Kollisionsstrategie wurde quadratisches Sondieren gewählt.

a) Geben Sie alle Reihenfolgen an, in denen die Schlüssel in die anfangs leere Hashtabelle eingefügt worden sein können.

b) Gibt es eine bessere Reihenfolge die Schlüssel einzufügen, die zu einer geringeren Anzahl bei einer erfolgreichen Suche durchschnittlich zu inspizierender Hashtabellenplätze führt (wobei die Suche nach jedem Schlüssel gleichwahrscheinlich ist)?

Aufgabe 25: Hashing – Kollisionsstrategien

4 Punkte

Gegeben seien für $m = 11$ die Hashfunktionen

$$h(k) = k \bmod m \qquad \text{sowie} \quad h'(k) = k \bmod 6.$$

a) Fügen Sie die Schlüsselfolge

$$16, 44, 21, 5, 19, 22, 8, 33, 27, 30$$

gemäß der Hashfunktion $h(k)$ in eine geschlossene Hashtabelle ein. Stellen Sie das Ergebnis graphisch dar. Verwenden Sie als Kollisionsstrategie

1) Lineares Sondieren:

$$h_i(k) = (h(k) + c \cdot i) \bmod m \quad \text{mit } c = 1;$$

2) Quadratisches Sondieren:

$$h_i(k) = (h(k) + i^2) \bmod m;$$

3) Doppel-Hashing:

$$h_i(k) = (h(k) + h'(k) \cdot i) \bmod m.$$

b) Geben Sie für den Aufgabenteil a.1) und $m = 14$ möglichst allgemein alle c an, die sich als unbrauchbar erweisen. Begründen Sie ihre Lösung.

Aufgabe 26: Hashing – Implementierung 5 Punkte

Implementieren Sie in Modula-3 oder C Wörterbuch-Operationen `Insert` (Hinzufügen eines Elements), `Search` (Test, ob ein gegebenes Schlüssel als Wert in der Hashtabelle vorkommt), `Delete` (Entfernen eines Schlüssels) und `Initialize` (leere Hashtabelle zur Verfügung stellen). Die Grundmenge der einzutragenden Elemente sei die Menge der positiven ganzen Zahlen. Verwenden Sie zur Speicherung eine geschlossene Hashtabelle sowie die Hashfunktion und Kollisionsstrategie aus Aufgabe 25 Teil a.3). Prüfen Sie Ihre Implementierung mit der in Aufgabe 25 Teil a) gegebene Zahlenfolge und fertigen Sie ein geeignetes Laufzeitprotokoll an. Dokumentieren und erläutern Sie die Korrektheit Ihrer `Delete`-Operation indem Sie als nächstes die "44" löschen und dann die "22" suchen. Was gibt es dabei zu beachten?

Aufgabe 27: Suchbäume 3 Punkte

In einem Suchbaum seien die Werte von 1 bis 1000 eingetragen. Welche der folgenden Wertefogen können *nicht* im Rahmen einer Search-Operation auftreten? Geben Sie die Rechts-Links-Verläufe der Pfade an.

(a) 2, 252, 401, 398, 330, 344, 397, 363

(b) 925, 202, 911, 240, 912, 245, 363

(c) 2, 399, 387, 219, 266, 382, 381, 278, 363

Lehrstuhl für Informatik VII
Rheinisch-Westfälische Technische Hochschule Aachen
http://www-i7.informatik.rwth-aachen.de/

Prof. Dr. W. Thomas
Dipl.-Inform. C. Hirsch

SS99
11. Juni 1999

9. Übung

Vorlesung Datenstrukturen und Algorithmen

Abgabe: am Freitag, 18.6.1999

Aufgabe 28: Suchbaum Eigenschaften — 3 Punkte

Sei T ein Suchbaum für die Menge S und $a \in S$ sei Wert eines Blattes von T. Betrachten Sie folgende Argumentation: Bei der Suche nach a wird S in drei Mengen X, Y, Z zerlegt: Y enthalte die Elemente von S die sich auf dem Pfad nach a befinden, X bzw. Z diejenigen, die sich links bzw. rechts des Pfades befinden. Gilt für alle $x \in X, y \in Y, z \in Z$, daß $x < y < z$? Geben Sie ein möglichst kleines Gegenbeispiel an.

Aufgabe 29: Optimaler Suchbaum — 4 Punkte

Erstellen Sie (per Hand) die Tabelle, die sich bei Berechnung des optimalen Suchbaumes für $S = \{1, 2, 3, 4, 5, 6\}$ mit $p_1 = 0.1$, $p_2 = 0.2$, $p_3 = 0.12$, $p_4 = 0.05$, $p_5 = 0.28$, $p_6 = 0.25$ ($q_0 = \cdots = q_6 = 0$) ergibt. Ermitteln Sie hieraus den optimalen Suchbaum.

Aufgabe 30: Bäume zum Sortieren — 4 Punkte

Man kann eine Menge S aus n Elementen (eines geordneten Universums) dadurch sortieren, daß man zunächst einen Suchbaum für S aufbaut und dann einen Inorder-Baumdurchlauf durchführt. Bestimmen Sie die Laufzeit im besten und im schlechtesten Fall.

Aufgabe 31: Implementierung 2-3-Baum — 5 Punkte

In dieser Aufgabe sollen Funktionen zum Auffinden und Speichern von Werten, hier positive natürliche Zahlen, in 2-3-Bäumen implementiert werden. Schreiben Sie in Modula-3 oder C Funktionen `Search` zum Auffinden eines gegebenen Schlüsselwertes und `Insert` zum Einfügen eines neuen Elementes.

Der Einfachheit halber soll nur ein gemeinsamer Datentyp sowohl für innere Knoten als auch für Blätter benutzt werden. Dieser Record soll Platz für vier Zeiger `Succ1`, `Succ2`, `Succ3`, `Vater` (initialisiert mit `NIL` bzw. `NULL`) sowie für zwei ganze Zahlen `Wert1`, `Wert2` beinhalten. Für innere Knoten werden in `Wert1` und ggf. `Wert2` die Trennwerte abgelegt; die Zeiger enthalten die Adressen der Nachfolgerknoten. Wird nur ein Trennwert benutzt, so wird `Wert2` auf 0 gesetzt, um dies anzuzeigen. Blätter werden durch Initialisieren von `Wert1` mit 0 markiert, und `Wert2` enthält dann den zu speichernden Wert; die Zeiger sind in diesem Fall unbenutzt.

Berechnen Sie hiermit die Trennwerte für die inneren Knoten des 2-3-Baumes, der sich nach Einfügen von 5, 8, 2, 12, 19, 16, 7 in den leeren Baum ergibt.

Lehrstuhl für Informatik VII
Rheinisch-Westfälische Technische Hochschule Aachen
http://www-i7.informatik.rwth-aachen.de/

Prof. Dr. W. Thomas
Dipl.-Inform. C. Hirsch

SS99
17. Juni 1999

10. Übung

Vorlesung Datenstrukturen und Algorithmen

Abgabe: am Freitag, 25.6.1999

Aufgabe 32: B-Baum 4 Punkte

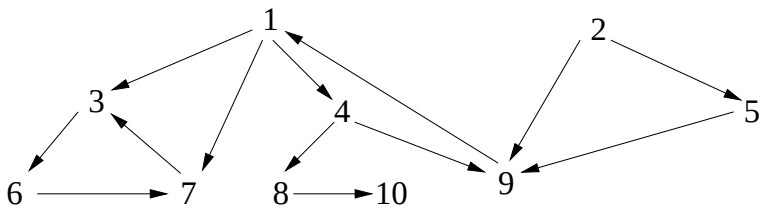
Konstruieren Sie den B-Baum, der sich aus dem leeren Baum ergibt, wenn man (mit der natürlichen Ordnung) nacheinander folgende Elemente einfügt:

$$F, S, Q, K, C, L, H, T, V, W, M, R, N, P, A, B, X, Y, D, Z, E.$$

Wie in der Vorlesung sei $t = 3$, d.h. B-Baum-Knoten enthalten zwischen zwei und fünf Elemente. Geben sie jeweils nur die vor Knotenaufspaltungen erreichten Konfigurationen an.

Aufgabe 33: Darstellung von Graphen 2 Punkte

Sei $G_0 = (\{1, \ldots, 10\}, E)$ der folgende Graph:



Geben Sie die Adjazenzmatrix und die Adjazenzliste an. (Für die Adjazenzmatrix genügen die 1-Einträge.)

Aufgabe 34: Tiefensuche 4 Punkte

Geben Sie für den Graphen G_0 aus Aufg. 33 den Verlauf der Tiefensuche beginnend mit Knoten 1 (wobei Nachfolgerknoten in aufsteigender Nummerierung besucht werden) an, indem Sie die d- und f-Werte an die Knoten schreiben. Geben Sie außerdem die Klassifizierung der Kanten (gemäß Vorlesung) an.

Aufgabe 35: Breitensuche 2 Punkte

Führen Sie für den Graphen G_0 aus Aufg. 33 die Breitensuche durch; notieren Sie die Reihenfolge der Knotenbesuche und heben Sie die entstehenden Baumkanten hervor.

Aufgabe 36: Eigenschaften der Tiefensuche 3 Punkte

Zeigen Sie durch ein Gegenbeispiel, daß folgende Behauptung *falsch* ist:

Gibt es im gerichteten Graphen G einen Pfad von u nach v und ergibt sich in einer Tiefensuche $d[u] < d[v]$, so ist v im entsprechenden Tiefensuchbaum ein Nachkomme von u.

Vorname:	Name:	Matr.-Nr.:

Aufgabe 7: Felder 6 Punkte

Ein Feld $A[1..n]$ aus natürlichen Zahlen im Bereich $[1..k]$ sei gegeben. Der gesuchte Algorithmus soll nach einer geeigneten Vorverarbeitung die Frage "wieviele der n Zahlen liegen im geschlossenen Intervall $[a, b]$" in Zeit $\mathcal{O}(1)$ beantworten.

Beispiel: Wir betrachten folgendes Feld $A[1..10]$ ueber dem Bereich $[1..8]$:

$$(\ 7,\ 2,\ 3,\ 3,\ 5,\ 4,\ 4,\ 7,\ 9,\ 4\ )$$

Fuer $a = 3$ und $b = 6$ ergeben sich 6 Zahlen im Intervall $[a, b]$.

Entwickeln Sie (mit Begründung) einen $\mathcal{O}(n + k)$-Algorithmus in Pseudocode für die Vorverarbeitung (d.h. für den Aufbau geeigneter Hilfsinformation), und geben Sie an, wie man damit die genannten Fragen in Zeit $\mathcal{O}(1)$ beantworten kann. Die Erinnerung an das Linearzeit-Sortieren ist nützlich.

Vorname:	**Name:**	**Matr.-Nr.:**

Aufgabe 8: Hashing 2 + 2 + 2 Punkte

Gegeben seien für $m = 11$ die Hashfunktionen

$$h(k) = k \bmod m \qquad \text{sowie} \quad h'(k) = 1 + (k \bmod 6).$$

Fügen Sie die Schlüsselfolge

$$21, 49, 26, 10, 24, 27, 13, 38, 32, 35$$

gemäß der Hashfunktion $h(k)$ in eine geschlossene Hashtabelle ein. Geben Sie die sich ergebende Tabelle an. Verwenden Sie als Kollisionsstrategie

a) Lineares Sondieren:

$$h_i(k) = (h(k) + c \cdot i) \bmod m \quad \text{mit } c = 1;$$

b) Quadratisches Sondieren:

$$h_i(k) = (h(k) + i^2) \bmod m;$$

c) Doppel-Hashing:

$$h_i(k) = (h(k) + h'(k) \cdot i) \bmod m.$$

Vorname:	Name:	Matr.-Nr.:

Aufgabe 9: Baum Operationen 2 + 3 + 3 Punkte

a) Konstruieren Sie den binären Suchbaum, der sich aus dem leeren Baum ergibt, wenn man nacheinander folgende Elemente einfügt:

$$K, Q, G, M, D, N, O, T, L, B, R, X, P, C, J.$$

Als nächstes entfernen Sie (in dieser Reihenfolge) die Elemente Q, N, G und geben Sie die jeweils erreichte Konfiguration an.

b) Konstruieren Sie den 2-3-Baum, der sich aus dem leeren Baum ergibt, wenn man nacheinander folgende Elemente einfügt:

$$O, E, A, B, P, T, D, C, Q.$$

Geben Sie jeweils die vor Knotenaufspaltungen und Umhängeoperationen erreichten Konfigurationen an.

c) Konstruieren Sie den B-Baum, der sich aus dem leeren Baum ergibt, wenn man nacheinander folgende Elemente einfügt:

$$N, Z, Q, F, M, P, Y, H, W, C, X, G, J, I, S, U, A, V, B, L, O, E, D, R, T, K.$$

Wie in der Vorlesung sei der Parameter für die Knotengröße hier $t = 3$. Geben Sie jeweils nur die vor Knotenaufspaltungen erreichten Konfigurationen an.

Vorname:	Name:	Matr.-Nr.:

Aufgabe 10: Binärer Suchbaum (Programmieren) 2 + 7 Punkte

a) Entwerfen Sie in Modula-3 oder C einen Record bzw. Struct für einen binären Suchbaum, in dem ganze Zahlen abgespeichert sind.

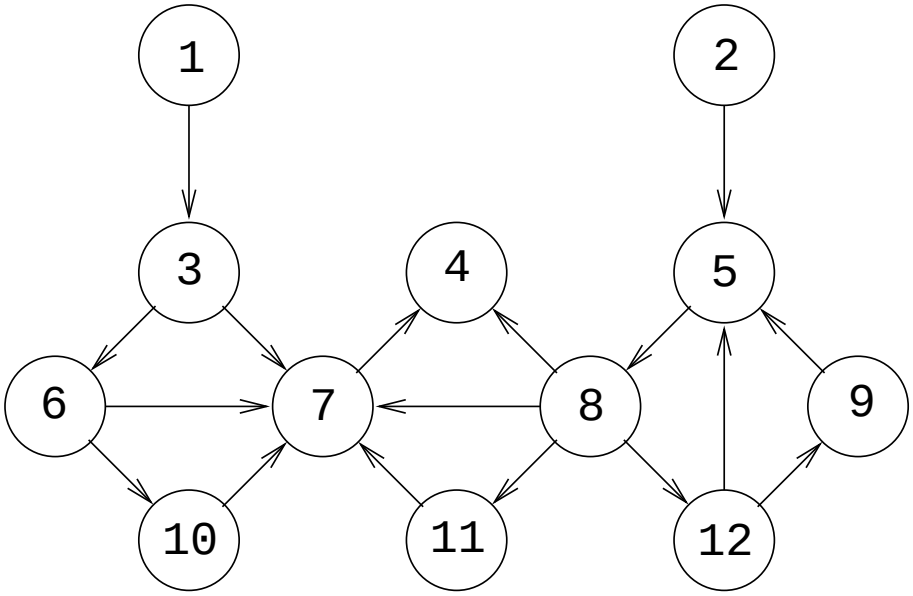
b) Schreiben Sie mit Rückgriff auf diese Datenstruktur zwei kommentierte Prozeduren zum Einfügen eines neuen Elements sowie zum Entfernen eines bestehenden Elements.

Hinweis: Der Baum sei dabei, wie üblich, durch einen Zeiger auf die Wurzel gegeben; der leere Baum durch einen NIL- bzw. NULL-Zeiger repräsentiert. Das Einfügen und Entfernen soll derart erfolgen, daß die Eigenschaften eines binären Suchbaumes erhalten bleiben.

Vorname:	Name:	Matr.-Nr.:

Aufgabe 11: Graph Algorithmen 1 + 1 + 3 + 3 + 3 Punkte

Gegeben sei der Folgende Graph G:



a) Stellen Sie die Adjazenzmatrix von G auf (wobei die 1-Einträge genügen).

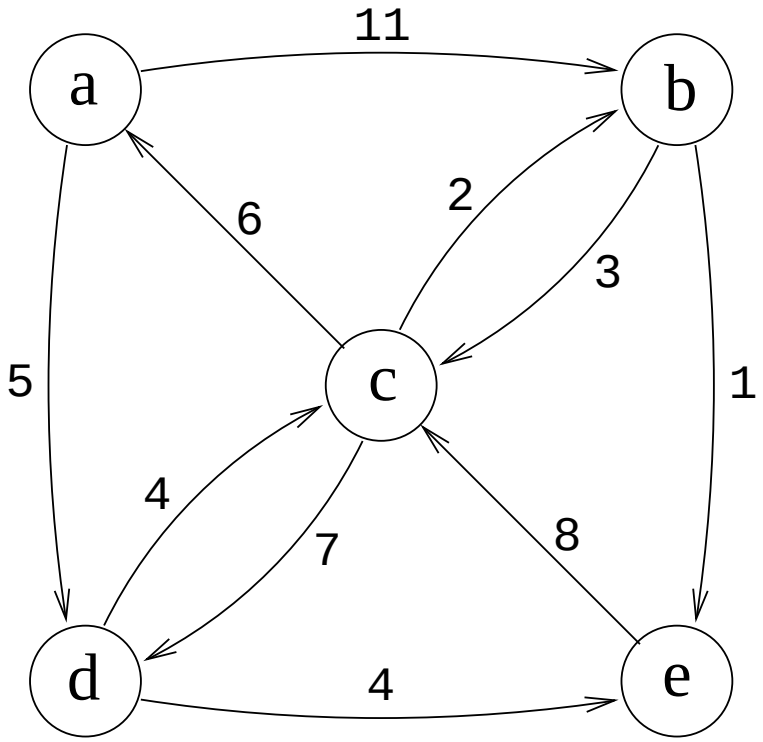
b) Stellen Sie G mit Hilfe von Adjazenzlisten dar.

c) Führen Sie in G eine Tiefensuche durch und geben Sie die dabei entstehenden `d`- und `f`-Werte der Knoten sowie die Klassifizierung der Kanten an. In den `for`-Schleifen der Tiefensuche sei dabei die Reihenfolge der Knoten nach Nummerierung angenommen.

Was ist die asymptotische Laufzeit der Tiefensuche (wie üblich in den Größen $|V|$ und $|E|$ mit Knotenmenge V und Kantenmenge E)?

Vorname:	Name:	Matr.-Nr.:

Für Aufgabenteil d) sei der folgende kantenbewertete Graph H gegeben:

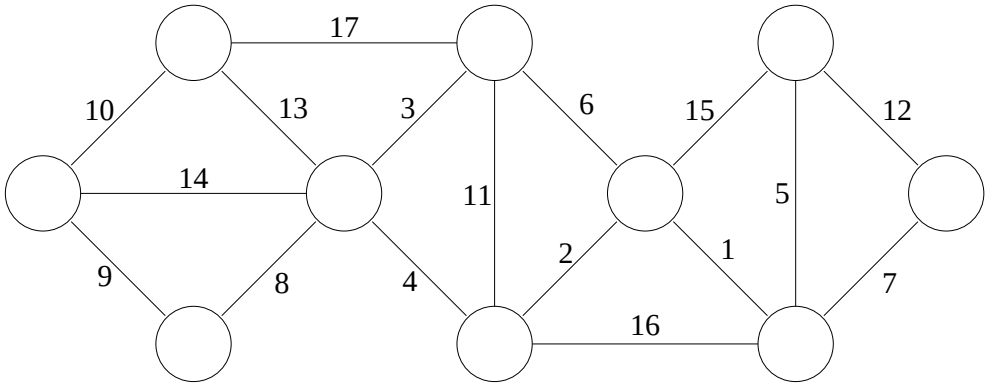


d) Führen sie den Dijkstra-Algorithmus beginnend bei Knoten `a` durch und notieren Sie dabei die Zwischenergebnisse.

Was ist die asymptotische Laufzeit des Dijkstra-Algorithmus (wieder in $|V|$ und $|E|$)?

Vorname:	**Name:**	**Matr.-Nr.:**

Für Aufgabenteil e) sei der folgende kantenbewertete ungerichtete Graph K gegeben:



e) Formulieren Sie in Pseudocode den Kruskal-Algorithmus zur Bestimmung eines minimalen spannenden Baumes.

Illustrieren Sie den Kruskal-Algorithmus anhand des gegebenen Beispiels durch Nummerierung der Kanten gemäß Verlauf des Algorithmus.

Vorname:	Name:	Matr.-Nr.:

Aufgabe 12: Dynamisches Programmieren 3 + 6 + 1 Punkte

Wir betrachten ein endliches Regelsystem zur Ersetzung von Dezimalziffern. Eine Regel hat die Form $z \to z'z''$ mit Dezimalziffern z, z', z''. Eine Ziffernfolge $z_1 \dots z_n$ heisst erzeugbar aus z, wenn man durch endlichmalige Anwendung der Regeln von z zu $z_1 \dots z_n$ gelangt (Kurzschreibweise: $z \to^* z_1 \dots z_n$).

Beispiel: Mit den Regeln $1 \to 11$, $1 \to 12$ und $2 \to 73$
sind nacheinander z.B. folgende Ziffernfolgen aus 1 erzeugbar:
1 (mit nullmaliger Anwendung der Regeln), 11, 111, 1211, 17311

Sie sollen einen Algorithmus entwickeln, der zu gegebener Ziffernfolge $z_1 \dots z_n$ diejenigen z bestimmt, aus denen man $z_1 \dots z_n$ erzeugen kann.

Hinweis: Bestimmen Sie nach der Methode des dynamischen Programmierens die Mengen

$$M_{ij} = \{z \mid z \to^* z_i \dots z_j\} \qquad (\text{ für } 1 \le i \le j \le n)$$

in geeigneter Reihenfolge. Beachten Sie dabei: Es gilt $z \to^* z_i \dots z_j$ für $i < j$ gdw. es gibt eine (zuerst angewendete) Regel $z \to z'z''$ und ein geeignetes k, so dass $z' \to^* z_i \dots z_k$ und $z'' \to^* z_{k+1} \dots z_j$ (mit $i \le k < j$).

a) Entwerfen Sie ein geeignetes Tabellenschema zur Berechnung der M_{ij} und füllen Sie es für das genannte Beispiel 17311 (und die genannten Regeln) zur Hälfte aus.

b) Formulieren Sie (in Pseudocode) einen Algorithmus, der bei Vorgabe von $z_1 \dots z_n$ die Menge M_{1n} liefert. Der Algorithmus sollte sich auf ein festes System R von Regeln beziehen. Sie dürfen indizierte Variablen für endliche Ziffernmengen und Operationen wie die Vereinigung (mit Zeitaufwand $\mathcal{O}(1)$) benutzen.

c) Führen Sie eine asymptotische Laufzeitabschätzung (in Abhängigkeit der Länge n der gegebenen Ziffernfolge; unabhängig von der Größe des Regelsystems) durch.

Diplom-Vorprüfung Informatik I 23.8.1999

Musterlösung zum Teil 2: Datenstrukturen

Aufgabe 7: Felder

6 Punkte

Ein Feld $A[1..n]$ aus natürlichen Zahlen im Bereich $[1..k]$ sei gegeben. Der gesuchte Algorithmus soll nach einer geeigneten Vorverarbeitung die Frage "wieviele der n Zahlen liegen im geschlossenen Intervall $[a, b]$" in Zeit $\mathcal{O}(1)$ beantworten.

Beispiel: Wir betrachten folgendes Feld $A[1..10]$ ueber dem Bereich $[1..8]$:

$$(7,\ 2,\ 3,\ 3,\ 5,\ 4,\ 4,\ 7,\ 8,\ 4)$$

Fuer $a = 3$ und $b = 6$ ergeben sich 6 Zahlen im Intervall $[a, b]$.

Entwickeln Sie (mit Begründung) einen $\mathcal{O}(n + k)$-Algorithmus in Pseudocode für die Vorverarbeitung (d.h. für den Aufbau geeigneter Hilfsinformation), und geben Sie an, wie man damit die genannten Fragen in Zeit $\mathcal{O}(1)$ beantworten kann. Die Erinnerung an das Linearzeit-Sortieren ist nützlich.

Lösung

Idee: Zunächst mit "Bucketsort" feststellen, wie oft jede Zahl $i \in [1, k]$ vorkommt (Zeit $\mathcal{O}(n)$). Dies liefert ein Feld $B[1..k]$ mit $B[i] =$ Anzahl i in A.

Dann B so modifizieren, daß $B[i] =$ Anzahl j in A mit $j \leq i$. Das geht in Zeit $\mathcal{O}(k)$: Sukzessive für wachsende i: $B[i] \leftarrow B[i] + B[i-1]$.

```
for i = 1 to k do
  B[i] := 0
od
for i = 1 to n do
  B[A[i]] := B[A[i]] + 1
od
for i = 2 to k do
  B[i] := B[i] + B[i - 1]
od
```

Die Frage nach der Anzahl Zahlen im Intervall $[a, b]$ kann dann in Zeit $\mathcal{O}(1)$ durch $B[b] - B[a-1]$ beantwortet werden (wobei $B[0] = 0$ gesetzt wird).

Aufgabe 8: Hashing 2 + 2 + 2 Punkte

Gegeben seien für $m = 11$ die Hashfunktionen

$$h(k) = k \bmod m \qquad \text{sowie} \quad h'(k) = 1 + (k \bmod 6).$$

Fügen Sie die Schlüsselfolge

$$21, 49, 26, 10, 24, 27, 13, 38, 32, 35$$

gemäß der Hashfunktion $h(k)$ in eine geschlossene Hashtabelle ein. Geben Sie die sich ergebende Tabelle an. Verwenden Sie als Kollisionsstrategie

a) Lineares Sondieren:

$$h_i(k) = (h(k) + c \cdot i) \bmod m \quad \text{mit } c = 1;$$

b) Quadratisches Sondieren:

$$h_i(k) = (h(k) + i^2) \bmod m;$$

c) Doppel-Hashing:

$$h_i(k) = (h(k) + h'(k) \cdot i) \bmod m.$$

Lösung

	0	1	2	3	4	5	6	7	8	9	10
a)	10	32	24	13	26	49	27	38	35		21
b)	10		24	13	26	49	27	35	32	38	21
c)	38	35	24	32	26	49	27		13	10	21

Aufgabe 9: Baum Operationen 2 + 3 + 3 Punkte

a) Konstruieren Sie den binären Suchbaum, der sich aus dem leeren Baum ergibt, wenn man nacheinander folgende Elemente einfügt:

$$K, Q, G, M, D, N, O, T, L, B, R, X, P, C, J.$$

Als nächstes entfernen Sie (in dieser Reihenfolge) die Elemente Q, N, G und geben Sie die jeweils erreichte Konfiguration an.

b) Konstruieren Sie den 2-3-Baum, der sich aus dem leeren Baum ergibt, wenn man nacheinander folgende Elemente einfügt:

$$O, E, A, B, P, T, D, C, Q.$$

Geben Sie jeweils die vor Knotenaufspaltungen und Umhängeoperationen erreichten Konfigurationen an.

c) Konstruieren Sie den B-Baum, der sich aus dem leeren Baum ergibt, wenn man nacheinander folgende Elemente einfügt:

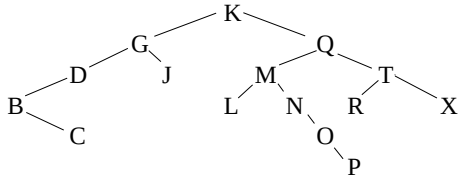
$$N, Z, Q, F, M, P, Y, H, W, C, X, G, J, I, S, U, A, V, B, L, O, E, D, R, T, K.$$

Wie in der Vorlesung sei der Parameter für die Knotengröße hier $t = 3$. Geben Sie jeweils nur die vor Knotenaufspaltungen erreichten Konfigurationen an.

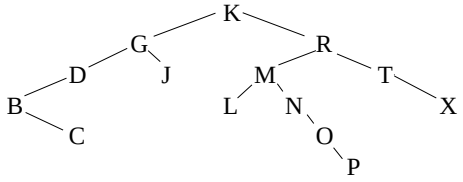
Lösung

Binärer Suchbaum:

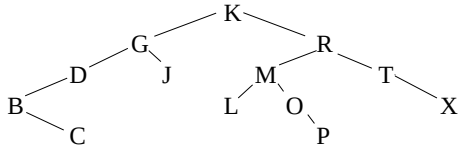
Voller Baum:



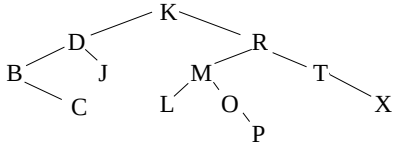
nach Löschen des "Q":



nach Löschen des "N":

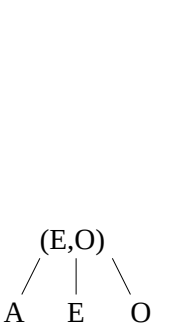


nach Löschen des "G":



2-3-Baum:

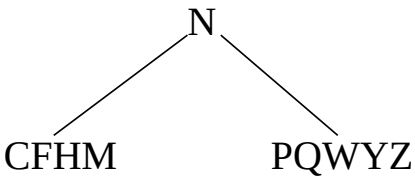
Anfang:



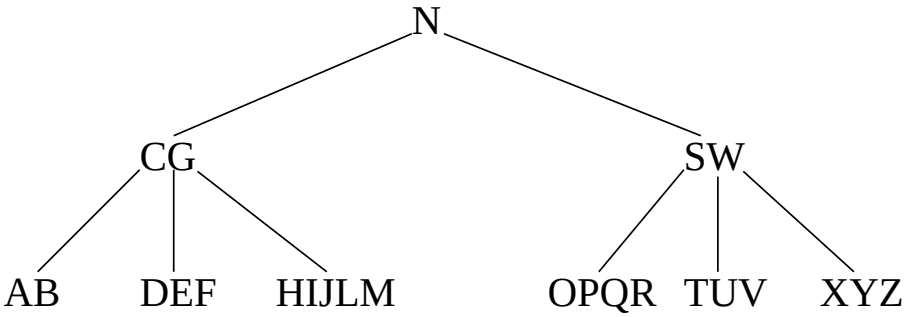
Voller Baum:

B-Baum:

Vor zweitem Aufspalten:



Voller Baum:



Aufgabe 10: Binärer Suchbaum (Programmieren) 2 + 7 Punkte

a) Entwerfen Sie in Modula-3 oder C einen Record bzw. Struct für einen binären Suchbaum, in dem ganze Zahlen abgespeichert sind.

b) Schreiben Sie mit Rückgriff auf diese Datenstruktur zwei kommentierte Prozeduren zum Einfügen eines neuen Elements sowie zum Entfernen eines bestehenden Elements.

Hinweis: Der Baum sei dabei, wie üblich, durch einen Zeiger auf die Wurzel gegeben; der leere Baum durch einen NIL- bzw. NULL-Zeiger repräsentiert. Das Einfügen und Entfernen soll derart erfolgen, daß die Eigenschaften eines binären Suchbaumes erhalten bleiben.

Lösung

Selbstverständlich ist die Lösung dieser Aufgabe nicht eindeutig.

Version in C

```
typedef struct _treenode {
  struct _treenode *left, *right;
  int value;
} treenode;

/* Prozedur bekommt Zeiger auf Zeiger auf Wurzelknoten;
   wird der Baum durch einen Zeiger
treenode *root;
   repraesentiert kann man durch
insert(&root, 42);
   einen neuen Knoten mit der Zahl 42 anlegen.
*/

void insert( treenode **rootptr, const int value )
{
  treenode **pivot = rootptr;

  while( (*pivot) != NULL ) {
    if( value == (*pivot)->value )
      /* der Wert ist bereits im Baum eingetragen, wir sind fertig */
      return;

    /* naechsten Teilbaum bestimmen */
    if( value < (*pivot)->value )
      pivot = &(*pivot)->left;
    else
```

```
      pivot = &(*pivot)->right;
  }

  /* neuen Knoten erzeugen */
  (*pivot) = (treenode *)(malloc( sizeof( treenode ) ) );

  /* hier muesst man testen, ob man den Speicher bekommen hat */

  /* und Knoten mit Daten fuellen */
  (*pivot)->value = value;
  (*pivot)->left  = NULL;
  (*pivot)->right = NULL;
}

void delete( treenode **rootptr, const int value )
{
  treenode **pivot = rootptr;
  treenode **walk;
  treenode *del;

  if( (*pivot) == NULL )
    /* der Baum ist leer */
    return;

  /* Knoten mit Element suchen */
  while( value != (*pivot)->value ) {
    /* in welchem Teilbaum ist es? */
    if( value < (*pivot)->value )
      pivot = &(*pivot)->left;
    else
      pivot = &(*pivot)->right;

    if( (*pivot) == NULL )
      /* der Teilbaum ist leer */
      return;
  }

  /* (*pivot) zeigt nun auf den zu loeschenden Knoten */

  if( (*pivot)->left == NULL || (*pivot)->right == NULL ) {
    del = (*pivot);
    /* es gibt nur einen Teilbaum, den wir direkt nach oben ziehen koennen */
    if( *pivot)->left == NULL )
      (*pivot) = (*pivot)->right;
```

```
    else
      (*pivot) = (*pivot)->left;
    /* alten Knoten freigeben */
    free( del );
  }
  else {
    /* der zu loeschende Knoten hat zwei Soehne */
    /* suche groessten Wert im linken Teilbaum */
    walk = &(*pivot)->left;
    while ( (*walk)->right != NULL )
      walk = &(*walk)->right;

    /* Wert nach oben tauschen */
    del->value = (*walk)->value;

    /* rekursiv weiterloeschen */
    delete( walk, (*walk)->value );
  }

}
```

Version in Modula-3

```
TYPE TNode = REF RECORD
  left, right : TNode;
  value : INTEGER;
END;

PROCEDURE Insert(VAR tree : TNode; value : INTEGER) =
VAR
  tmp, run : TNode;
BEGIN
  tmp := NEW(TNode);
  tmp^.value := value;
  tmp^.left := NIL;
  tmp^.right := NIL;
  IF tree = NIL THEN
    tree := tmp;
  ELSE
    run := tree;
    IF run^.value > value THEN
      IF run^.right = NIL THEN
        run^.right = tmp;
      ELSE
```

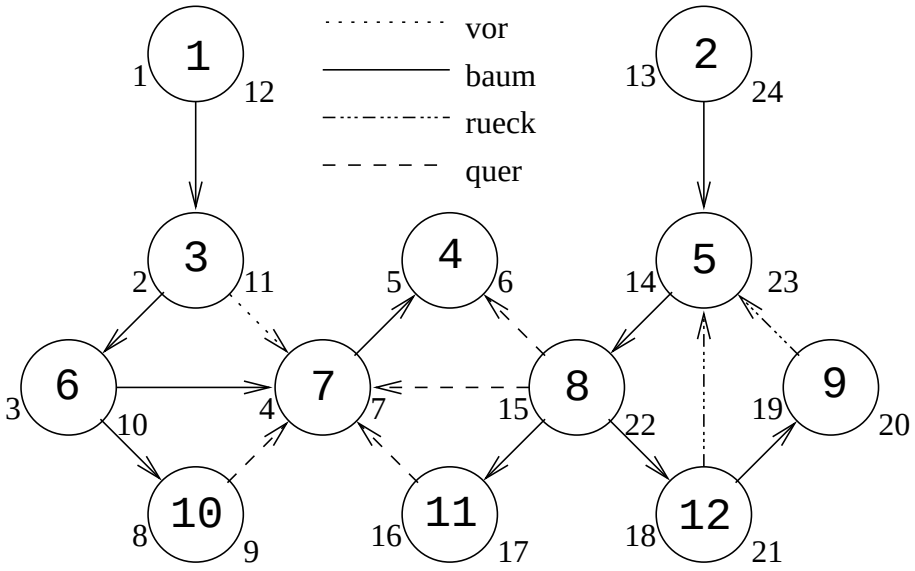
```
        run := run^.right;
      END;
    ELSIF run^.value < value THEN
      IF run^.left = NIL THEN
        run^.left = tmp;
      ELSE
        run := run^.left;
      END;
    END;
  END;
END;

PROCEDURE Delete(VAR tree : TNode; value : INTEGER) :
VAR
  tmp, pre : TNode;
BEGIN
  IF tree # NIL THEN
    pre := NIL;
    tmp := tree;
    WHILE (tmp # NIL) AND (tmp^.value # value) DO
      pre := tmp;
      IF tmp^.value < value THEN
        tmp := tmp^.left;
      ELSIF tmp^.value = value THEN
        tmp := tmp^.right;
      END;
    END;
    IF tmp # NIL THEN
      IF tmp^.left = NIL THEN
        IF pre^.left = tmp THEN
          pre^.left := tmp^.right;
        ELSE
          pre^.right := tmp^.right;
        END;
      ELSIF tmp^.right = NIL THEN
        IF pre^.left := tmp THEN
          pre^.left := tmp^.left;
        ELSE
          pre^.right = tmp^.left;
        END;
      ELSE (* zu loeschender Knoten hat zwei Nachfolger *)
        pre := tmp^.left;
        WHILE pre^.right # NIL DO
          pre := pre^.right;
```

```
        END;
        tmp^.value := pre^.value;
        Delete(pre, pre^.value);
      END;
    END;
  END;
END;
```

Aufgabe 11: Graph Algorithmen 1 + 1 + 3 + 3 + 3 Punkte

Gegeben sei der Folgende Graph G:



a) Stellen Sie die Adjazenzmatrix von G auf (wobei die 1-Einträge genügen).

b) Stellen Sie G mit Hilfe von Adjazenzlisten dar.

c) Führen Sie in G eine Tiefensuche durch und geben Sie die dabei entstehenden `d`- und `f`-Werte der Knoten sowie die Klassifizierung der Kanten an. In den `for`-Schleifen der Tiefensuche sei dabei die Reihenfolge der Knoten nach Nummerierung angenommen.

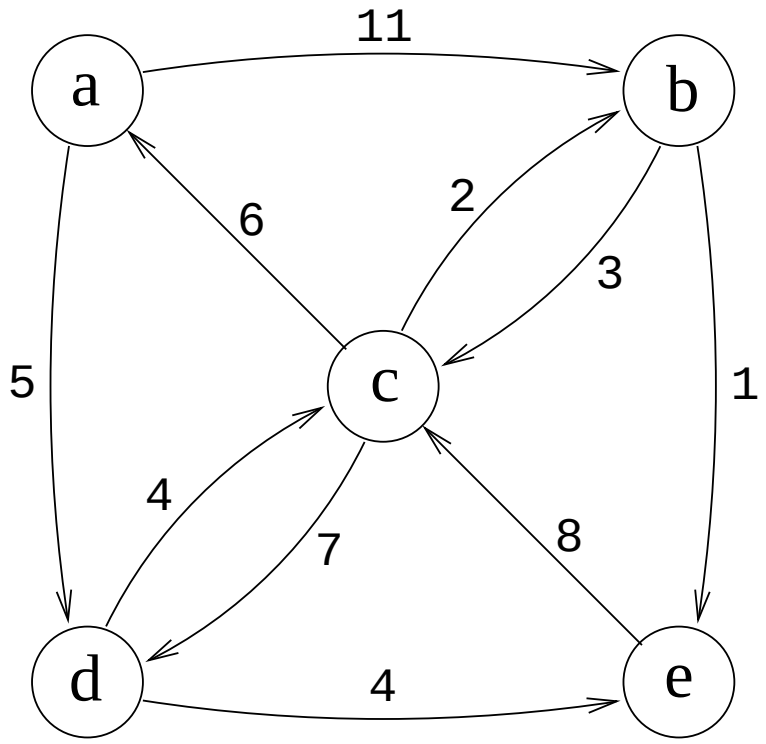
Was ist die asymptotische Laufzeit der Tiefensuche (wie üblich in den Größen $|V|$ und $|E|$ mit Knotenmenge V und Kantenmenge E)?

Lösung

a)	1	2	3	4	5	6	7	8	9	10	11	12	b)				
1			1										→	3			
2					1								→	5			
3						1	1						→	6	7		
4													→				
5								1					→	8			
6							1			1			→	7	10		
7				1									→	4			
8				1			1				1	1	→	4	7	11	12
9					1								→	5			
10							1						→	7			
11							1						→	7			
12					1				1				→	5	9		

c) Laufzeitkomplexität: $\mathcal{O}(|V| + |E|)$.

Für Aufgabenteil d) sei der folgende kantenbewertete Graph H gegeben:



d) Führen sie den Dijkstra-Algorithmus beginnend bei Knoten `a` durch und notieren Sie dabei die Zwischenergebnisse.

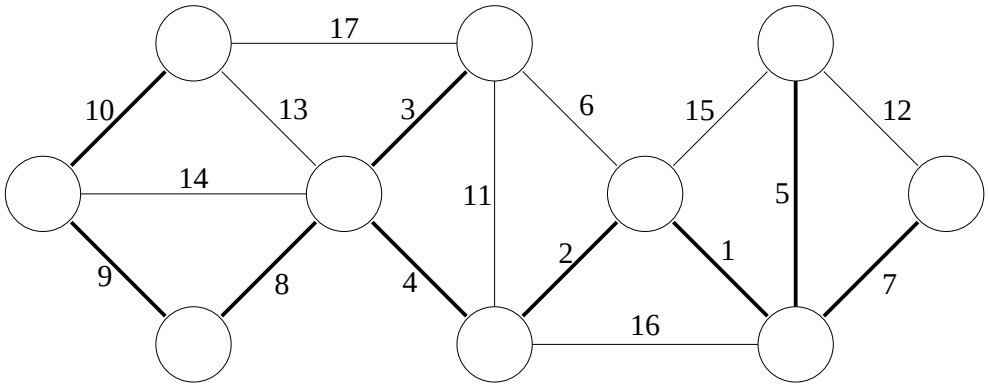
Was ist die asymptotische Laufzeit des Dijkstra-Algorithmus (wieder in $|V|$ und $|E|$)?

Lösung

Laufzeitkopmlexität: $\mathcal{O}(|V|^2)$.

Schritt	neu	S	$d(a)$	$d(b)$	$d(c)$	$d(d)$	$d(e)$
0	(a)	$\{a\}$	0	11	∞	5	∞
1	d	$\{a, d\}$	0	11	9	5	9
2	c	$\{a, c, d\}$	0	11	9	5	9
3	e	$\{a, c, d, e\}$	0	11	9	5	9
4	b	$\{a, b, c, d, e\}$	0	11	9	5	9

Für Aufgabenteil e) sei der folgende kantenbewertete ungerichtete Graph K gegeben:



e) Formulieren Sie in Pseudocode den Kruskal-Algorithmus zur Bestimmung eines minimalen spannenden Baumes.

Illustrieren Sie den Kruskal-Algorithmus anhand des gegebenen Beispiels durch Nummerierung der Kanten gemäß Verlauf des Algorithmus.

Lösung

Reihenfolge der Selektion der Kanten: (1, 2, 3, 4, 5, 7, 8, 9, 10)

Gegeben: kantenbewerteter Graph (V, E)
Gesucht: spannender Baum B mit minimalem Gewicht
. initialisiere $B = \emptyset$
. baue Heap aus Kanten auf (nach Gewicht, mit billigster Kante als Wurzel)
. solange Heap nicht leer:
. entferne billigste Kante e (= Wurzel) aus Heap
. falls $B \cup \{e\}$ keine Zykel enthält, füge e zu B hinzu
. sonst verwerfe e

Aufgabe 12: Dynamisches Programmieren 3 + 6 + 1 Punkte

Wir betrachten ein endliches Regelsystem zur Ersetzung von Dezimalziffern. Eine Regel hat die Form $z \to z'z''$ mit Dezimalziffern z, z', z''. Eine Ziffernfolge $z_1 \dots z_n$ heisst erzeugbar aus z, wenn man durch endlichmalige Anwendung der Regeln von z zu $z_1 \dots z_n$ gelangt (Kurzschreibweise: $z \to^* z_1 \dots z_n$).

Beispiel: Mit den Regeln $1 \to 11$, $1 \to 12$ und $2 \to 73$
sind nacheinander z.B. folgende Ziffernfolgen aus 1 erzeugbar:
1 (mit nullmaliger Anwendung der Regeln), 11, 111, 1211, 17311

Sie sollen einen Algorithmus entwickeln, der zu gegebener Ziffernfolge $z_1 \dots z_n$ diejenigen z bestimmt, aus denen man $z_1 \dots z_n$ erzeugen kann.

Hinweis: Bestimmen Sie nach der Methode des dynamischen Programmierens die Mengen

$$M_{ij} = \{z \mid z \to^* z_i \dots z_j\} \qquad (\text{ für } 1 \le i \le j \le n)$$

in geeigneter Reihenfolge. Beachten Sie dabei: Es gilt $z \to^* z_i \dots z_j$ für $i < j$ gdw. es gibt eine (zuerst angewendete) Regel $z \to z'z''$ und ein geeignetes k, so dass $z' \to^* z_i \dots z_k$ und $z'' \to^* z_{k+1} \dots z_j$ (mit $i \le k < j$).

a) Entwerfen Sie ein geeignetes Tabellenschema zur Berechnung der M_{ij} und füllen Sie es für das genannte Beispiel 17311 (und die genannten Regeln) zur Hälfte aus.

b) Formulieren Sie (in Pseudocode) einen Algorithmus, der bei Vorgabe von $z_1 \dots z_n$ die Menge M_{1n} liefert. Der Algorithmus sollte sich auf ein festes System R von Regeln beziehen. Sie dürfen indizierte Variablen für endliche Ziffernmengen und Operationen wie die Vereinigung (mit Zeitaufwand $\mathcal{O}(1)$) benutzen.

c) Führen Sie eine asymptotische Laufzeitabschätzung (in Abhängigkeit der Länge n der gegebenen Ziffernfolge; unabhängig von der Größe des Regelsystems) durch.

Lösung

a) Tabellenschema:

	1	2	3	4	5
1	{1}	-	{1}	{1}	{1}
2		{7}	{2}	-	-
3			{3}	-	-
4				{1}	{1}
5					{1}

b) Gegeben: R, $z_1, \ldots, z_n$
Gesucht: M_{1n}

```
1 initialisiere alle M_ij mit ∅
2 für alle i : 1 ≤ i ≤ n
3     setze M_ii = {z_i}
4 für alle d : 2 ≤ d ≤ n − 1
5     für alle i : 1 ≤ i ≤ n − d
6         j = i + d
7         für alle k : i ≤ k ≤ j − 1
8             für alle x ∈ M_ik
9                 für alle y ∈ M_k+1j
10                    falls z → xy ∈ R füge z zu M_ij hinzu
```

c) Laufzeitabschätzung

Asymptotischer Aufwand (in n):

(a)	für Zeilen	8.-10.	$\mathcal{O}(1)$
(b)	für Zeilen	4.-10.	$\mathcal{O}(n^3)$
(c)	für Zeilen	2.-3.	$\mathcal{O}(n)$
(d)	für Zeilen	8.-10.	$\mathcal{O}(n^2)$

Insgesamt: $\mathcal{O}(n^3)$.