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:

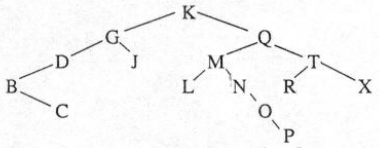
$$N, Z, Q, F, M, P, Y, H, W, C, X, G, J, I, S, U, A, V, B, L, O, E, D, R, T, K.$$

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

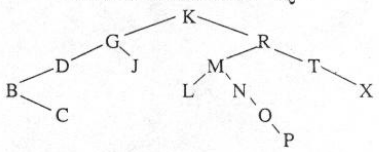
Lösung

Binärer Suchbaum:

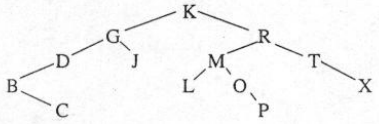
Voller Baum:



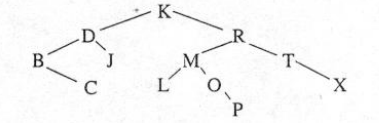
nach Löschen des "Q":



nach Löschen des "N":

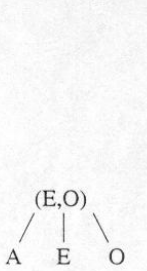


nach Löschen des "G":



2-3-Baum:

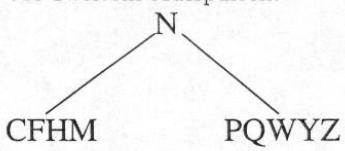
Anfang:



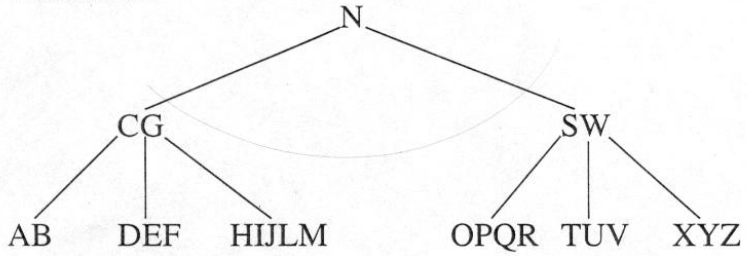
Voller Baum:

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
```

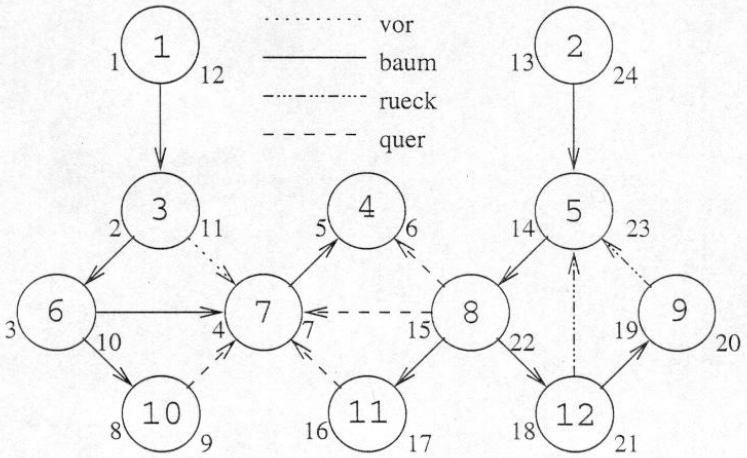
```
        run := run^.right;
      END;
    ELSIF run^.value < value THEN
      IF run^.left = NIL THEN
        run^.left = tmp;
      ELSE
        run := run^.left;
      END;
    END;
  END;
END;

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)?

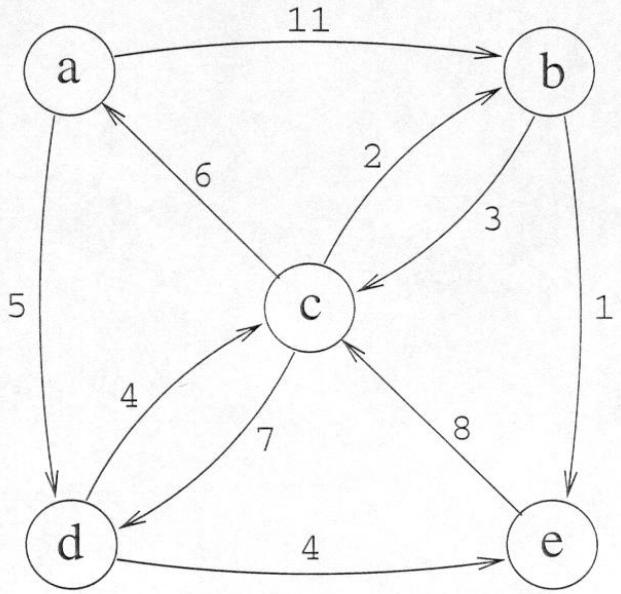
Lösung

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

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

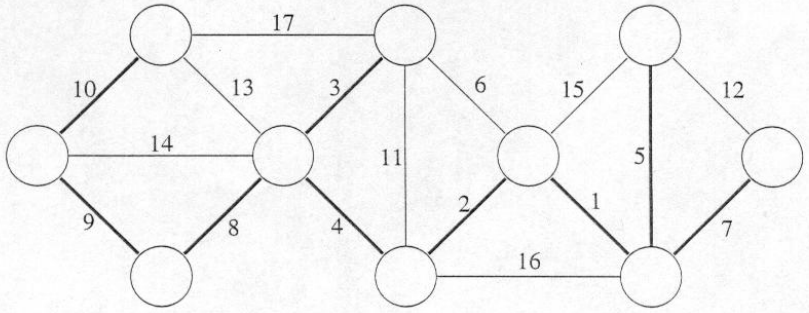
Was ist die asymptotische Laufzeit des Dijkstra-Algorithmus (wieder in $|V|$ und $|E|$)?

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