

Norman May

B6, 29, Raum C0.05
 68131 Mannheim
 Telefon: (0621) 181-2517
 Email: norman@pi3.informatik.uni-mannheim.de

Matthias Brantner

B6, 29, Raum C0.05
 68131 Mannheim
 Telefon: (0621) 181-2517
 Email: msb@pi3.informatik.uni-mannheim.de

Algorithmen und Datenstrukturen
 Wintersemester 2004/05

6. Lösungsvorschlag
 10. Dezember 2004

Aufgabe 1

7 Punkte

Gegeben folgende Operationen auf einer Menge S , deren Elemente mit einem Schlüssel versehen sind. Auf der Schlüsselmenge ist eine totale Ordnung definiert.

- $Search(S,k)$: sucht das Element mit dem Schlüssel k in der Menge S
 $Insert(S,x)$: fügt das Element x in S ein
 $Delete(S,x)$: löscht das Element x aus S
 $Minimum(S)$: liefert das Element mit dem minimalen Schlüssel aus S
 $Maximum(S)$: liefert das Element mit dem maximalen Schlüssel aus S
 $Successor(S,x)$: liefert zu x das Element mit dem nächstgrößeren Schlüssel
 $Predecessor(S,x)$: liefert zu x das Element mit dem nächstkleineren Schlüssel

Beachten Sie, dass x einen Verweis auf ein Element darstellt und nicht den Schlüssel eines Elements! Verwenden Sie die Θ -Notation. Gehen Sie davon aus, dass $|S| = n$ ist. Tragen Sie in folgende Tabelle die Worst-Case-Laufzeiten ein.

	unsortierte ein- fach verkettete Liste	unsortierte doppelt verkettete Liste	Binary Heap
$Search(S,k)$			
$Insert(S,x)$			
$Delete(S,x)$			
$Minimum(S)$			
$Maximum(S)$			
$Successor(S,x)$			
$Predecessor(S,x)$			

Lösung

	unsortierte einfach verkettete Liste	unsortierte doppelt verkettete Liste	Binary Heap
$Search(S,k)$	$\Theta(n)$	$\Theta(n)$	$\Theta(n)$
$Insert(S,x)$	$\Theta(1)$	$\Theta(1)$	$\Theta(\lg n)$
$Delete(S,x)$	$\Theta(n)$	$\Theta(1)$	$\Theta(\lg n)$
$Minimum(S)$	$\Theta(n)$	$\Theta(n)$	$\Theta(n)$
$Maximum(S)$	$\Theta(n)$	$\Theta(n)$	$\Theta(1)$
$Successor(S,x)$	$\Theta(n)$	$\Theta(n)$	$\Theta(n)$
$Predecessor(S,x)$	$\Theta(n)$	$\Theta(n)$	$\Theta(n)$

Aufgabe 2
3 Punkte

Beschreiben Sie die Implementation zweier Stacks mittels eines einzelnen Arrays $A[1, \dots, n]$. Die Implementation soll sicherstellen, dass nur dann ein Überlauf in einem der beiden Stacks auftritt, wenn die Gesamtanzahl der in den beiden Stacks gespeicherten Elemente n übersteigt. Die Laufzeit der Operationen *push* und *pop* soll jeweils $O(1)$ sein.

Lösung

Implementierung mittels eines Arrays A mit $length[A] = n$.



Aufgabe 3
6 Punkte

Eine doppelt verkettete Liste läßt sich mit einem einzelnen Zeiger $np[x]$ pro Element x realisieren. Dabei wird davon ausgegangen, dass alle Zeiger als k-Bit-Integer dargestellt werden können. Sei $A \oplus B$ die bitweise XOR-Verknüpfung der k-Bit-Integer A und B, dann hat der Zeiger $np[x]$ den Wert $next[x] \oplus prev[x]$.

Aufgabe 3 a)
2 Punkte

Zeigen Sie, dass für zwei k-Bit-Integer A und B , $(A \oplus B) \oplus B = A$ gilt.

Lösung

Sei $A := \langle a_1, \dots, a_k \rangle$, $a_i \in \{0, 1\}$ und $B := \langle b_1, \dots, b_k \rangle$, $b_i \in \{0, 1\}$

Zu zeigen $(A \oplus B) \oplus B = A$.

Beweis:

Zu zeigen $(a_i \oplus b_i) \oplus b_i = a_i$ für alle $a_i, b_i, i \in \{1, \dots, k\}$

Wahrheitstabelle:

a_i	b_i	$a_i \oplus b_i$	$(a_i \oplus b_i) \oplus b_i$
0	0	0	0
0	1	1	0
1	0	1	1
1	1	0	1

Aufgabe 3 b)

4 Punkte

Geben Sie die Funktionen *Search*, *Insert* und *Delete* an. Beschreiben Sie außerdem, wie die Liste in $O(1)$ umgedreht werden kann.

Lösung

List-Search (L, k)

```

 $p \leftarrow nil$ 
 $x \leftarrow head[L]$ 
while  $x \neq nil$  and  $key[x] \neq k$ 
  do  $n \leftarrow np[x] \oplus p$ 
      $p \leftarrow x$ 
      $x \leftarrow n$ 
return  $x$ 

```

List-Insert (L, x)

```

 $np[x] \leftarrow head[L]$ 
if  $head[L] \neq nil$ 
  then  $np[head[L]] \leftarrow np[head[L]] \oplus x$ 
 $head[L] \leftarrow x$ 

```

List-Delete (L, x)

```

 $p \leftarrow nil$ 
 $i \leftarrow head[L]$ 
while  $i \neq x$ 
  do  $n \leftarrow np[i] \oplus p$ 
      $p \leftarrow i$ 
      $i \leftarrow n$ 
 $n \leftarrow np[i] \oplus p$ 
if  $p \neq nil$ 
  then  $np[p] \leftarrow (np[p] \oplus x) \oplus n$ 
  else  $head[L] \leftarrow n$ 
if  $n \neq nil$ 
  then  $np[n] \leftarrow (np[n] \oplus x) \oplus p$ 
  else  $tail[L] \leftarrow p$ 

```

Um eine Liste L in $O(1)$ umdrehen zu können, benötigt man $head[L]$ und $tail[L]$ oder einen Sentinel $nil[L]$. Zum Umdrehen muss dann lediglich der Wert von $head[L]$ mit dem von $tail[L]$ bzw. der Wert von $prev[nil[L]]$ mit dem von $next[nil[L]]$ vertauscht werden.

Aufgabe 4**8 Punkte**

Aufgabe 4 a)**4 Punkte**

Implementieren Sie einen Binärbaum mit der folgenden Schnittstelle in einer von ihnen gewählten Programmiersprache (Java, C, C++ oder Scheme). Auf der Webseite finden Sie als Beispiel einen Klassenrumpf in Java.

```
public class BinaryTree {  
    /**  
     * insert a new node with  
     * a given key and eliminate duplicates  
     * @param value – the value of the key to insert  
     */  
    public void insert(int value);  
  
    /**  
     * lookup a Subtree with a given key  
     * @param value – the value of the Subtree to lookup  
     */  
    public Subtree lookup(int value);  
  
    /**  
     * print an ordered sequence  
     * of the keys stored in the tree  
     */  
    public void printOrdered();  
}
```

Lösung

Lösung siehe BinaryTree.java

Aufgabe 4 b)**4 Punkte**

Implementieren Sie eine nichtrekursive Funktion, die in $O(n)$ ihren Binärbaum mit n Knoten durchläuft und die Struktur des Baumes mit allen Schlüsseln ausgibt. Die Funktion soll, abgesehen von dem Eingabebaum, nur konstant viel Speicher benötigen. Außerdem soll der Eingabebaum nicht modifiziert werden. Eine Beispielausgabe nachdem die Werte 2,1,4,0 und 3 in dieser Reihenfolge eingefügt wurde könnte wie folgt aussehen:

2

1
0
4
3

Linke Kindern werden zuerst ausgegeben. Die Einrückung gibt die Ebene im Baum an. Bei einer Implementierung in Scheme sind globale Variablen und Zuweisungen erlaubt.

Lösung

Lösung siehe BinaryTree.java

Aufgabe 5

5 Punkte

Eine Möglichkeit zur Darstellung von Bäumen ist ein sog. unbeschränkter Baum. Dabei besitzt jeder Knoten einen Zeiger zu seinem Eltern-Element `parent(x)`. Anstatt jedoch einen Zeiger für jedes Kind zu speichern, werden lediglich zwei Zeiger verwaltet. Dabei zeigt `left-child(x)` auf das erste Kind eines Knotens `x` und `right-sibling(x)` zum nächsten rechten Geschwister Knoten von `x`.

Aufgabe 5 a)

3 Punkte

Wie kann man einen unbeschränkten Baum mit nur zwei Zeigern und einem Wahrheitswert in jedem Knoten darstellen?

Lösung

Ein Zeiger zeigt auf das erste Kind. Der Wahrheitswert gibt an, ob der zweite Zeiger auf den nächsten Geschwisterknoten zeigt oder im Falle des letzten Kindes ein Elternzeiger ist.

Aufgabe 5 b)

2 Punkte

Erläutern Sie die Vor- und Nachteile Ihre Repräsentation.

Lösung

- Im Fall, dass die Repräsentation eines Wahrheitswertes weniger Platz verbraucht, wie die eines Zeigers, kann man Speicher sparen.
- Der Zugriff auf den Elternknoten ist in Worst-Case $O(n)$, wobei n die maximale Anzahl der Kinder ist.
- Tradeoff Memory - Performance