Grundzüge DS & Alg (WS14/15)

Lösungsvorschlag zu Aufgabenblatt 6

Aufgabe 1

Für "Höhe eines Baumes" benutzen wir im Folgenden die Definition "Höhe ist Anzahl Knoten des Längsten Pfades". Für die Definition mit "Kanten" anstatt Knoten ändern sich die Zahlen nur etwas, der Beweis bleibt der gleiche.

Maximale Höhe

Der degenerierte binäre Suchbaum, bei dem jeder Knoten außer dem (einen) Blatt nur ein Kind hat, hat Höhe n. Er entsteht etwa, wenn n Knoten in aufsteigender Reihenfolge eingefügt werden, da dann immer im rechten Teilbaum eingefügt wird.

Damit ein Suchbaum Höhe h haben kann, muss es mindestens einen Pfad dieser Länge geben. Ein Pfad der Höhe h hat h Knoten. Somit kann es keinen Suchbaum mit Höhe $> n$ geben, da sonst der "verursachende" Pfad $> n$ Knoten hätte.

Somit ist die Höhe n maximal.

Minimale Höhe

Ein voller binärer Suchbaum der Tiefe h hat $2^h - 1$ Knoten (inklusive Blätter). Für einen vollen binären Suchbaum mit n Knoten gilt daher:

$$2^h - 1 = n \quad \iff \quad h = \log_2(n+1)$$

Ein balancierter Baum, dessen Pfadlängen sich maximal um 1 unterscheiden, hat $> 2^{h-1} - 1$ und $\leq 2^h - 1$ Blätter. Für einen balancierten Baum mit n Knoten gilt also:

$$\begin{array}{rccccc} & 2^{h-1} - 1 & < & n & \leq & 2^h - 1 \\ \iff & 2^{h-1} & < & n+1 & \leq & 2^h \\ \iff & h-1 & < & \log_2(n+1) & \leq & h \end{array}$$

Eine geringere Höhe kann nicht erreicht werden, da dann eine dichtere Packung als beim vollen binären Suchbaum gefunden wäre.

Grundzüge DS & Alg (WS14/15)

Lösungsvorschlag zu Aufgabenblatt 6

Aufgabe 2

Successor

Um den Nachfolger von x zu finden müssen zwei Fälle unterschieden werden:

1. x hat ein rechts Kind, m.a.W. einen rechten Teilbaum. Dann ist der Nachfolger von x das Minimum dieses rechten Teilbaumes.
2. x hat kein rechts Kind, m.a.W. es existiert kein rechter Teilbaum. In diesem Fall muss der BST in Richtung Wurzel durchlaufen werden. Der Nachfolger von x ist der erste Knoten, der größer als $x.key$ ist. Dies ist für den Knoten y genau dann der Fall, wenn x im linken Teilbaum von y liegt. Existiert ein solcher Knoten nicht gibt es keinen Nachfolger.

```
1: procedure SUCC(node x)
2:     if x.right ≠ NULL then
3:         return MIN (x.right)
4:     y = x.parent
5:     while y ≠ NULL AND x = y.right  do
6:         x = y
7:         y = y.parent
8:     return y
```

Predecessor

Der Vorgänger lässt sich ähnlich wie der Nachfolger finden, auch hier unterscheidet man zwei Fälle:

1. x hat ein linkes Kind, dann ist es das Maximum des linken Teilbaumes.
2. x hat kein linkes Kind, dann ist analog zu oben der Baum Richtung Wurzel zu durchlaufen, diesmal so lange, bis der aktuelle Knoten zum ersten Mal rechtes Kind ist. Dann ist der Vaterknoten der Vorgänger.

```
1: procedure PRED(node x)
2:     if x.left ≠ NULL then
3:         return MAX (x.left)
4:     y = x.parent
5:     while y ≠ NULL AND x = y.left  do
6:         x = y
7:         y = y.parent
8:     return y
```

Laufzeit

Sollte das Ergebnis im rechten, bzw. linken Teilbaum liegen ist das Minimum, bzw. Maximum dieses Teilbaums zu bestimmen. Das Extremum lässt sich in $O(h')$ bestimmen, wobei $h' =$ Höhe des Teilbaums. Im schlechtesten Fall ist x die Wurzel, der Teilbaum hat deswegen die Höhe $h-1$. In diesem Fall beträgt die Laufzeit damit $O(h)$, wobei $h =$ Höhe des Gesamtbaums.

Wenn der Baum nach oben durchlaufen werden muss kann der maximal bis zu der Wurzel durchlaufen werden. Im Worst Case ist x das tiefste Blatt und damit der zu durchlaufende Pfade genau so lang wie der Baum Hoch. Damit ergibt sich auch hier eine Laufzeit von $O(h)$.

Damit ist insgesamt die Laufzeit in $O(h)$.

Aufgabe 3

a) Wir fügen jedem Knoten das Feld "GroesseTeilbaum" hinzu, welches speichert, wie groß der Teilbaum unter dem Knoten ist (mit $TNULL.GroesseTeilbaum = 0$). Dadurch lässt sich das Problem mit folgendem Ansatz lösen:

Wir prüfen zuerst, ob wir den Knoten schon gefunden haben. Falls nicht, schauen wir uns den linken Teilbaum an. Wenn dieser kleiner ist wie der gesuchte Rang, so ziehen wir die Größe vom Rang ab (da all diese Elemente nicht in Frage kommen könne) und prüfen ob wir im rechten Teilbaum weiter suchen müssen oder ob wir das Element direkt angeben können.
Ist die Größe des linken Teilbaums größer wie der gesuchte Rang, so suchen wir im linken Teilbaum nach dem passenden Element.
Wenn wir das Element mit Rang 1 suchen, also das "kleinste" Element, so gehen wir immer nach "links" (also zum kleinsten Element hin) bis es kein linkeres Element mehr gibt.

```
1:  procedure FindK(node x, value k)
2:      if k > x.GroesseTeilbaum OR k < 1 then
3:          k invalid
4:      else if x.left.GroesseTeilbaum < k then
5:          k = k − x.left.GroesseTeilbaum
6:          if k = 1 then
7:              return x
8:          else
9:              FindK (x.right, k − 1)
10:     else
11:         FindK (x.left, k)
```

Der Algorithmus läuft den Baum von "oben nach unten" durch, womit er offensichtlich Laufzeit $O(h)$ mit $h =$ Höhe des Baumes besitzt.

b) Prinzipielle müssen wir nur dafür sorgen, dass das Feld "GroesseTeilbaum" geupdatet wird, falls ein Element eingefügt oder gelöscht wird. Dazu fügen wir einfach folgende Zeilen hinzu:

```
1:  procedure INSERT(node x, key k)
2:      if x = TNULL then
3:          x = NEWNODE (k)
4:      else if k < x.key then
5:          INSERT (x.left, k)
6:      else if k > x.key then
7:          INSERT (x.right, k)
8:      else
9:          k already in tree
10:     x.GroesseTeilbaum = x.left.GroesseTeilbaum + x.right.GroesseTeilbaum + 1
```

```
1:  procedure NEWNODE(key k)
2:      create node x
3:      x.left = TNULL
4:      x.right = TNULL
5:      x.key = k
6:      x.GroesseTeilbaum = 1
```

```
1:  procedure DELETE(node x, key k)
2:      if x = TNULL then
3:          k not in tree
4:      else if k < x.key then
5:          DELETE (x.left, k)
6:      else if k > x.key then
7:          DELETE (x.right, k)
8:      else if x.left = TNULL then
9:          x = x.right
10:         free memory for the node
11:     else if x.right = TNULL then
12:         x = x.left
13:         free memory for the node
14:     else
15:         z = MIN (x.right)
16:         copy data from z to x
17:         DELETE (x.right, z.key)
18:     x.GroesseTeilbaum = x.left.GroesseTeilbaum + x.right.GroesseTeilbaum + 1
```