Mohamad Raafat Baki

Übung 6 – Bäume:

6.1.1. Binäre Suchbäume:

Skizzieren Sie einen vollständigen balancierten binären Suchbaum mit 7 Knoten, der die Schlüssel 2,4,6,8,10,12 und 14 enthält!

a) Wie viele Blätter enthält dieser Baum?

b) Wie groß ist die Höhe dieses Baumes?

c) Wie viele Vergleichsoperationen compare(key1,key2) braucht man, um in diesem Suchbaum nach einem Schlüssel zu suchen? (compare liefert als Ergebnis eine -1 falls key1<key2, 0 falls key1==key2 und +1 falls key1>key2 ist).

d) Wie groß ist die Höhe eines beliebigen binären Baumes mindestens, wenn dieser n Blätter hat?

e) Wie groß ist die Höhe eines beliebigen binären Baumes mindestens, wenn dieser n Knoten hat?

f) Wie groß ist die Höhe eines beliebigen Baumes höchstens, wenn dieser n Knoten hat?

a) Blätter(n) = 2^h , h = 2 Blätter(n) = 2^2 = 4 .

b) h = log2(n) h = log2(4) = 2 .

c) 3 .

d) h = log2(n) .

e) n = 2^(h+1) -1 2^(h+1) = n -1 h +1 = log2(n+1)

h = log2(n+1) -1 .

f) h = n – 1 .

6.1.2. Durchlaufen eines binären Baumes :

Beschreiben Sie in Pseudocode in rekursiver Form, wie alle Schlüssel eines aufsteigend sortierten binären Suchbaumes in :

a) aufsteigender Reihenfolge und

b) absteigender Reihenfolge

ausgegeben werden können. Benutzen Sie zur Ausgabe eines Schlüssels die Funktion print(key).

struct tree

num key

tree left

tree right

var root = new tree

root.key = 4

var left1 = new tree

root.left = left1

left1.key = 3

left1.left = null

left1.right = null

var right1 = new tree

root.right = right1

right1.key = 6

var right2 = new tree

right1.right = right2

right2.key = 8

right2.left = null

right2.right = null

var right1left2 = new tree

right1.left = right1left2

right1left2.key = 5

right1left2.left = null

right1left2.right = null

a)

func visit\_lwr(node)

if node==null

return

visit\_lwr(node.left)

print node.key

visit\_lwr(node.right)

visit\_lwr(root)

b)

func visit\_rwl(node)

if node==null

return

visit\_rwl(node.right)

print node.key

visit\_rwl(node.left)

visit\_rwl(root)

6.1.3. Einfügen und Löschen :

a) In einen anfangs leeren binären Suchbaum B1 werden die Schlüssel 8, 4, 12, 2, und 10 in dieser Reihenfolge eingefügt. Skizzieren Sie, wie der Baum jetzt aussieht.

b) In einen anfangs leeren zweiten binären Suchbaum B2 werden die Schlüssel 2, 4, 8, 10, und 12 in dieser Reihenfolge eingefügt. Skizzieren Sie, wie der Baum jetzt aussieht.

c) Vergleichen Sie die beiden Suchbäume hinsichtlich ihrer Höhe. Welcher Baum ist für Suchoperationen im worst-case effizienter, begründen Sie kurz!

a)

b)

c) h1 = 2 .

h2 = 4 .

Erster Baum ist effizienter, weil die Binärsuche in dem durchgeführt werden kann. Baum 2 ist eine einfach verketette Liste, in der nur eine lineare Suche durchgeführt werden kann.

6.2.1. Einfügen von 255 Schlüsseln in einen leeren binären Suchbaum :

In einen anfangs leeren binären Suchbaum werden nacheinander die Schlüssel 1,2, 3, … 255 in genau dieser Reihenfolge eingefügt.

a) Welche Höhe hat der so entstandene Baum, wenn beim Einfügen keinerlei Rebalancierung durchgeführt wird?

b) Wie viele Blätter hat der so entstandene Baum, wenn beim Einfügen keinerlei Rebalancierung durchgeführt wird?

c) Welche Höhe hätte ein vollständig balancierter binärer Suchbaum, der die obigen Schlüssel enthält?

d) Wie viele Blätter hätte ein vollständig balancierter binärer Suchbaum, der die obigen Schlüssel enthält?

a) h = n – 1 = 254 . (n = knoten ) .

b) n = 1 . (n = Blätter ) .

c) h = log2(n+1) -1 h = log2(255+1) -1 = 7 .

d) n = 2^h n = 2^7 = 128 . ( n = Blätter ) .

6.2.2. mögliche Suchpfade in einem binären Suchbaum :

In einem binären Suchbaum, dessen genaue Struktur nicht bekannt ist, seien die natürlichen Zahlen zwischen 1 und 1000 gespeichert. In diesem Baum wird nun nach der Zahl 363 gesucht. Welche der folgenden Zahlensequenzen könnte ein möglicher Suchpfad in dem Baum sein und welche nicht?

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

b) 924, 220, 911, 244, 898, 258, 362, 363.

c) 295, 202, 911, 240, 912, 245, 363.

d) 2, 399, 387, 219, 266, 382, 381, 278, 363.

e) 935, 278, 347, 621, 299, 392, 358, 363.

a,b,d,e ist möglicher Suchpfad.

C ist kein möglicher Suchpfad.

6.3.1. Anzahl der Knoten eines Baumes in Pseudocode berechnen :

Implementieren Sie einen Algorithmus nodes(t), der die Anzahl der Knoten eines binären Baumes t ermittelt. Verwenden Sie bei der Implementierung die rekursive Definition von binären Bäumen.

struct tree

num key

tree left

tree right

var root = new tree

root.key = 4

var left1 = new tree

root.left = left1

left1.key= 1

left1.left = null

left1.right = null

var right1 = new tree

root.right= right1

right1.key =7

var right2 = new tree

right1.right = right2

right2.key= 9

right2.left= null

right2.right= null

var left1right1 = new tree

right1.left= left1right1

left1right1.key = 5

left1right1.left = null

left1right1.right = null

var a = 0

func anzahl(node)

if (node.left) != null

anzahl(node.left)

if (node) != null

a = a + 1

if (node.right) != null

anzahl(node.right)

return a

print anzahl(root)

6.3.2. Höhe eines Baumes in Pseudocode berechnen :

Implementieren Sie einen Algorithmus height(t), der die Höhe eines binären Baumes t ermittelt. Verwenden Sie bei der Implementierung die rekursive Definition von binären Bäumen.

struct tree

num key

tree left

tree right

var root = new tree

root.key = 4

var left1 = new tree

root.left = left1

left1.key= 1

left1.left = null

left1.right = null

var right1 = new tree

root.right= right1

right1.key =7

var right2 = new tree

right1.right = right2

right2.key= 9

right2.left= null

right2.right= null

var left1right1 = new tree

right1.left= left1right1

left1right1.key = 5

left1right1.left = null

left1right1.right = null

func hight(t)

var k = new tree

k = t

if k == null

return -1

else

var l = hight(k.left)

var r = hight(k.right)

if l > r

return l + 1

else

return r + 1

print hight(root)

6.3.3. Anzahl der Blätter in Pseudocode berechnen :

Implementieren Sie einen Algorithmus leaves(t), der die Anzahl der Blätter eines binären Baumes t ermittelt. Verwenden Sie bei der Implementierung die rekursive Definition von binären Bäumen. Tipp: Implementieren Sie zuerst eine Hilfsfunktion leaf(t) die prüft ob der Knoten t ein Blatt ist.

struct tree

num key

tree left

tree right

var root = new tree

root.key = 4

var left1 = new tree

root.left = left1

left1.key= 1

left1.left = null

left1.right = null

var right1 = new tree

root.right= right1

right1.key =7

var right2 = new tree

right1.right = right2

right2.key= 9

right2.left= null

right2.right= null

var left1right1 = new tree

right1.left= left1right1

left1right1.key = 5

left1right1.left = null

left1right1.right = null

func leaf(t)

var s = new tree

s = t

if s.left == null && s.right == null

return 1

else

return 0

var k = 0

func leafes(t)

var s = new tree

s = t

if s == null

return

if leaf(t) == 1

k = k+1

return

leafes(s.left)

leafes(s.right)

leafes(root)

print k

6.4.1. AVL Bäume erkennen :

Welche der abgebildeten Bäume sind AVL-Bäume? Dabei soll es nicht auf die SuchbaumEigenschaft ankommen, die Zahlen seien lediglich zur Nummerierung der Knoten, nicht als Schlüssel zu sehen.

a,b,e,f sind AVL Bäume.

c und d sind keine AVL Bäume.