

Algorithmen und Datenstrukturen

SoSe25

-Assignment 5-

Moritz Ruge

Matrikelnummer: 5600961

Lennard Wittenberg

Matrikelnummer: —

Mai 2025

Problem 1: Analyse von (a,b)-Bäumen

Bestimmen Sie die maximale Anzahl von Einträgen in einem (a, b) – Baum der Höhe h . Was folgt daraus für die Höhe eines (a, b) – Baums mit n Einträgen?

Gegeben:

- Wurzel hat min 2 Kinder
- Blätter haben min a und max b Kinder
- Wurzel hat min 1 bis $b - 1$ Einträge
- Blätter haben min $a - 1$ bis $b - 1$ Einträge

Gesucht: die maximale Anzahl von Einträgen in ein (a, b) – Baum der Höhe h

Lösung: Schauen wir uns einen (a, b) – Baum pro Ebene an, wobei die Höhe eines Baumes mit der Länge des Pfads von der Wurzel zu einem Blatt definiert ist. Auf Ebenen bezogen hat ein Baum der Höhe h also: $h + 1$ Ebenen (Ebene 0 bis Ebene h).

- Ebene(0) = Wurzel (1 Knoten)
- Ebene(1) = max b Knoten
- Ebene(2) = max b^2 Knoten
- Ebene(3) = max b^3 Knoten
- Ebene(h) = max b^h Knoten \Rightarrow max Einträge sind also b^h

Teil 2: was folgt daraus für die Höhe eines (a, b) Baums mit n Einträgen?

Gegeben:

- max Einträge = b^h
- n Einträge

Gesucht: Höhe für n Einträge

Lösung:

- $n \leq b^h$ | Umstellen nach h , Umkehrung zur Exponentialform
- $a^x \geq y \Rightarrow \log_a(y) \geq x$
- $b^h \geq n \Rightarrow \log_b(n) \geq h$

Beispiel: Angenommen $b = 4$ (jeder Knoten hat max 4 Kinder) und $n = 100$ Einträge

- $\log_4(100) \geq h$
- $\frac{\log_1 0(100)}{\log_1 0(4)} = \frac{2}{0,602} = 3,32$

\Rightarrow Damit wir eine valide Höhe kriegen, runden wir auf

\Rightarrow Die Höhe h muss als min 4 sein!

\Rightarrow Formel anpassen: $h = \lceil \log_b(n) \rceil$

□

Problem 2: Analyse von Skiplisten

Sei L eine Skipliste mit n Einträgen.

a) Zeigen Sie, dass L im Erwartungswert $O(n)$ Knoten besitzt.

Gegeben:

L0:	1	2	3	4	5	6	7	8	9	10
L1:	1		3		5		7		9	
L2:	1				5				9	
L3:					5					

$\Rightarrow L_i \equiv O(\log n)$

\Rightarrow Die Wahrscheinlichkeit, dass ein Element auf die nächste Liste kommt, ist $1/2$, da die Elemente pro Liste halbiert werden.

\Rightarrow Die Wahrscheinlichkeit, dass ein Element in Ebene " i " erscheint, ist dann:

$$Pr = \frac{1}{2} * \frac{1}{2} * \frac{1}{2} * \frac{1}{2} * \frac{1}{2} * \dots * \frac{1}{2} = \left(\frac{1}{2}\right)^i$$

$$Pr = L_1 * L_2 * L_3 * L_4 * L_5 * \dots * L_i$$

Gesucht: Wieviele Knoten besitzt eine Skipliste mit n Einträgen?

Lösung:

- Ein Element auf der Ebene 0, 1, 2 zählt als 3 Knoten – ein Knoten pro Ebene
- Ein Element ist auf Ebene " i " mit der Wahrscheinlichkeit $(1/2)^i$
- Wenn wir den Erwartungswert für die gesamte Knotenanzahl haben wollen, addieren wir die Wahrscheinlichkeiten
- $\Rightarrow 1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \frac{1}{16} + \dots = \sum_{i=0}^{\infty} (1/2)^i$
- Das ist eine geometrische Reihe, die konvergiert (einen Grenzwert bildet)

geometrische Reihe: $\sum_{i=0}^{\infty} q^k$

$\rightarrow |q| < 1$ Konvergiert (nähert sich Grenzwert)

$\rightarrow |q| > 1$ Divergiert (kein Grenzwert)

\Rightarrow Grenzwert: $S = \frac{1}{1-q}$

$\rightarrow S = \frac{1}{1-1/2} = \frac{1}{1/2} = 2$

Jedes Element erzeugt im Schnitt also 2 Knoten. Die gesamte Anzahl an Knoten in einer Skipliste ist dann Anzahl Elemente * Grenzwert für ein Element:

$n * 2 = 2n \rightarrow$ für die O-Notation sind Konstanten irrelevant, also ist der Erwartungswert $O(n)$

□

b) Zeigen Sie, dass für alle $j \geq 1$ die Wahrscheinlichkeit, dass L aus mindestens j Listen besteht, höchstens $\frac{n}{2^j - 1}$ ist.

Hinweis: Für Ereignisse $[A_1, \dots, A_l]$ gilt: $Pr[A_1 \cup \dots \cup A_l] \geq \sum_{i=1}^l Pr[A_i]$

1. Wenn die Liste L mindestens j Ebenen/Verkettete Listen hat, dann muss mindestens ein Knoten n_i aus L aus der Basis-Liste $j - 1$ mal in eine nächst höhere Ebene überführt worden sein. d.h. bei einem Bernouliexperiment (Münzwurf) wurde $j - 1$ mal das selbe Ergebnis erzielt und ein Knoten n_i wurde $j - 1$ mal in eine nächst höhere Ebene überführt.
2. Die Wahrscheinlichkeit p , dass ein Knoten n_i in eine nächsthöhere Ebene überführt wird liegt bei 50%. $\Rightarrow p(Erfolg) = 1/2$ Für den Erwartungswert, dass ein Knoten n_i 1-mal in eine nächsthöhere Ebene überführt wird gilt: $E[x] = p(Erfolg) = 1/2$ Nun wird betrachtet, dass ein Knoten $j - 1$ mal überführt wird. Für diesen Erwartungswert gilt: $E[x] = j - 1 * p(Erfolg)$
 $E[x] = 1/2 * 1/2 * \dots * 1/2(j - 1 \text{ mal})$ $E[x] = 1/2^{(j - 1)}$
3. Für die Anzahl der zu erwartenden Knoten, die $j - 1$ mal überführt worden gilt: $X_i = 1$ wenn die n_i $j - 1$ mal hintereinander überführt worden ist. ansonsten $X_i = 0$ Für die Summe X aller X_i (alle Knoten die $j - 1$ mal überführt worden sind) gilt: $E[x] = n * 1/2^{(j - 1)} = n/2^{(j - 1)}$
4. Für den Fall, dass eine Liste L mit n Einträgen bei $j \geq 1$ Versuchen aus mindestens j Listen besteht gilt: $Pr(X \geq 1) \leq E[x] = n/2^{(j - 1)}$

Problem 3: Implementierung von Skiplisten

Implementieren Sie eine Skipliste in Scala (oder Java)

Datei: SkipList.java

```
1 import java.util.Random;
2 import java.util.NoSuchElementException;
3
4 public class SkipList {
5     private Node head;
6     private int maxLevel;
7     private int level;
8     private Random random;
9
10    public SkipList() {
11        maxLevel = 6; // maximum number of levels
12        // working with low upper-bounds and therefore also with smaller skiplists
13        for readability and debugging.
14        level = 0; // current level of SkipList
15        head = new Node(Integer.MIN_VALUE, maxLevel);
16        random = new Random(); // for randomness of coinflips
17    }
18    public int getLevel(){
19        return this.level;
20    }
21    public Node getHead(){
22        return this.head;
23    }
24    private int randomLevel() { // determine how often a new inserted node will be
25        promoted to a higher level.
26        int lvl = 0;
27        // Keep flipping coins as long as we get heads (probability < 0.5)
28        // and we haven't reached maxLevel
29        while (random.nextDouble() < 0.5 && lvl < this.maxLevel) {
30            lvl++;
31        }
32        return lvl;
33    }
34    // Returns an array of predecessors where predecessors[i] is the last node
35    // at level i whose value is less than the search value.
36    private Node[] findPredecessors(int searchValue) {
37        Node[] predecessors = new Node[maxLevel + 1];
38        Node curr = head;
39        // begin search at highest populated level. continue until node is found on
40        the lowest level if present.
41        for (int i = this.level; i >= 0; i--) {
42            // move right as long as there are values on the same level, that a
43            smaller than the searchValue. else move down 1 level
44            while (curr.forward[i] != null && curr.forward[i].value < searchValue) {
45                curr = curr.forward[i];
46            }
47            predecessors[i] = curr;
48        }
49        return predecessors;
50    }
51    // lookup -> check if a given node is part of the Skiplist
52    public Node search(int searchValue){
53        Node[] preds = this.findPredecessors(searchValue);
54        // At level 0 preds[0] is the greatest node thats still smaller, than the
55        searchValue node.value < searchValue
56        Node curr = preds[0].forward[0]; // expected found position
```

```

53         if(curr.value == searchValue && curr != null){
54             return curr; // if the current node is the wanted node than return that
55             node
56         }
57         else{
58             throw new NoSuchElementException();
59             //return null; // else return null -> the wanted node is not part of the
Skiplist.
60         }
61     }
62     public void insert(int value){ // insertion -> insert a node with a given value
into the Skiplist.
63         Node[] preds = this.findPredecessors(value); // get all predecessors of the
node that have smaller values than value. This variable
64         Node curr = preds[0]; // get the greatest node in the Skiplist that whos value
is still smaller than value.
65         Node successor = curr.forward[0]; // Node to check if a node with the given
value already exists seperatly.
66
67         if(successor != null && successor.value == value){ // node with value exists
no insertion return.
68             return;
69         }
70         // the given value is not part of the Skiplist.
71         int newNodeLvL = randomLevel(); // flip coins to determine the levels where
the newNode should be on.
72         if(newNodeLvL > this.level){ // if the rolled highest level of the newNode is
higher, than the highest currently populated level. add empty levels ontop
73             for(int i = this.level+1; i <= newNodeLvL; i++){
74                 preds[i] = this.head;
75             }
76             this.level = newNodeLvL;
77         }
78         Node newNode = new Node(value, newNodeLvL); // declare and initialize the new
node with the given value and new nodes maxlevel.
79         for(int i = 0; i <= newNodeLvL;i++){ // from the base-level too the highest
populated level
80             // one each level newNode.successor becomes the successor of the current
predecessor, which is greater than newNode.
81             // This includes the case where insertion adds a node at the end of the
skiplist. in theory that is tail.value = +infinity and in implementation practice
its just null.
82             newNode.forward[i] = preds[i].forward[i];
83             preds[i].forward[i] = newNode; // after the copy process the successor of
pred[i] becomes the newNode.
84         }
85     }
86     public void delete(int value){ // deletion -> delete a existing node with given
Value from all Skiplist levels.
87         Node[] preds = this.findPredecessors(value); // get all predecessors of the
node that have smaller values than value. This variable
88         Node curr = preds[0].forward[0]; // curr is now the Node to be deleted.
89
90         if(curr != null && curr.value == value){ // if curr is indeed the node to be
deleted begin deletion process. else do nothing and exit the function.
91             for(int i = 0; i <= this.level;i++){ // from the base-level to the highest
populated level delete curr by:
92                 if(preds[i].forward[i] != curr){ // if on any level the current node
to be deleted isn't there anymore -> stop as curr has been deleted.
93                     return;
94                 }

```

```

95         preds[i].forward[i] = curr.forward[i]; // delete curr by skipping over
           it and "cutting" connection with its predecesser and successor -> cleaned up by
           garbage collector.
96     }
97     // after deleting curr check for empty levels from top to bottom and
           remove them by decrementing the level attribute.
98     while(level > 0 && this.head.forward[level] != null){
99         level--;
100     }
101 }
102 }
103 }

```

Datei: Node.java

```

1 public class Node {
2     int value;
3     Node[] forward; // The forward array stores references to the next nodes at each
                       level. forward[0] is the base-level. when having a base-list of (1,2,3,4) and
                       starting at head.forward[0] = 2, head.forward[0].forward[0] = 3 etc.
4
5     public Node(int value, int level) {
6         this.value = value;
7         this.forward = new Node[level + 1]; // declare an array of size level+1 to be
           able to hold level elements.
8     }
9     public int getValue(){
10         if(this != null){
11             return this.value;
12         }
13         else{
14             return -1;
15         }
16     }
17 }

```

Datei: SkipList_test.java

```

1 /*sources: https://www.geeksforgeeks.org/java-util-random-nextint-java/
2            ADS Skript
3            https://github.com/LJWittenberg/Java-code-for-Refactoring (refactorcode
           unteranderen)
4            https://www.tutorialspoint.com/java/util/random\_nextdouble.htm
5            https://stackoverflow.com/questions/2707322/what-is-null-in-java
6            https://www.baeldung.com/java-skiplist#bd-5-delete-operation
7            https://stackoverflow.com/questions/7630014/difficulty-throwing-
           nosuchelementexception-in-java
8            https://rollbar.com/guides/java/how-to-throw-exceptions-in-java/
9
10 */
11 import java.util.Random;
12 public class SkipList_test {
13     // declare and initialize 3 test Skiplists.
14     // 1. 1-20 inserted in ascending order
15     // 2. 20 random Int values between 1-50
16     // 3. 1-20 inserted in reverse order
17     public static void main(String[] args){
18         SkipList sklst1 = new SkipList();
19         SkipList sklst2 = new SkipList();
20         SkipList sklst3 = new SkipList();
21         SkipList sklst0 = new SkipList(); // empty List
22         for(int i = 1; i <= 20; i++){
23             sklst1.insert(i);

```

```

24     }
25     PrintSkipList(sklst1);
26     Random randomgen = new Random();
27     for(int i = 1; i <= 20; i++){
28         sklst2.insert(randomgen.nextInt(1,50));
29     }
30     PrintSkipList(sklst2);
31     for(int j = 20; j >= 1; j--){
32         sklst3.insert(j);
33     }
34     PrintSkipList(sklst3);
35     PrintSkipList(sklst0);
36     // lookup test:
37     Node searchResult = new Node(0,10);
38     searchResult = searchHandler(sklst1,10); // look for value that exists.
39     expected output: the wanted Node
40     if(searchResult.getValue() == 0){
41         System.out.println("The wanted Node does not exit in the selected SkipList
returning std.Failuire Value: 0");
42     }
43     else{
44         System.out.println(searchResult.getValue());
45     }
46     searchResult = searchHandler(sklst1,30); // look for value that does not exist
47     . expected output: the wanted Node
48     if(searchResult.getValue() == 0){
49         System.out.println("The wanted Node does not exit in the selected SkipList
returning std.Failuire Value: 0");
50     }
51     else{
52         System.out.println(searchResult.getValue());
53     }
54     // deletion test:
55     sklst0.delete(1); // deleting from empty List.
56     // deleting 3 nodes from the assending list
57     sklst1.delete(10);
58     sklst1.delete(1);
59     sklst1.delete(20);
60     PrintSkipList(sklst1);
61     // deleting an node that does not exists
62     sklst3.delete(30);
63     PrintSkipList(sklst3);
64     // deleting a node in the highest populated level.
65     int hgtLevel3 = sklst3.getLevel();
66     Node hgtNode3 = sklst3.getHead().forward[hgtLevel3];
67     int hgtNode3Val = hgtNode3.getValue();
68     sklst3.delete(hgtNode3Val);
69     PrintSkipList(sklst3);
70     return;
71 }
72 // helperfunction to visualize the SkipList
73 public static void PrintSkipList(SkipList sklst){
74     int printLevels = sklst.getLevel();
75
76     for(int i = printLevels; i >= 0; i--){
77         Node curr = sklst.getHead();
78         System.out.print("Level "+ i + ": HEAD ->");
79         while(curr.forward[i] != null){
80             curr = curr.forward[i];
81             System.out.print(curr.getValue() + " -> ");
82         }
83         System.out.println("TAIL");
84     }
85 }

```



```

82     }
83     System.out.println();
84     return;
85 }
86 // helperfunction to avoid manuell null-handling every time SkipList.search() is
87 // used.
88 public static Node searchHandler(SkipList lst ,int value){
89     Node result = new Node(0,lst.getLevel());
90     try{
91         result = lst.search(value);
92     } catch (Exception e){
93         e.printStackTrace();
94     }
95     return result;
96 }
97 }
98 }

```