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 darasu 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.

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

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 höchere Ebene überführt worden sein. d.h bei einem Bernoulisexperiment (Münzwurf) wurde $j-1$ mal das selbe Ergebnis erziehlt und ein Knoten n_i wurde $j-1$ mal in eine nächst höchere Ebene überführt.

2. Die Wahrscheinlichkeit p, dass ein Knoten n_i in eine nächsthöchere 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: $E[x] = j - 1 * p(Erfolg)$ $E[x] = 1/2 * 1/2 * ... * 1/2(j-1mal)$ $E[x] = 1/2^(j-1)$

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

```java
import java.util.Random;
import java.util.NoSuchElementException;

public class SkipList {
    private Node head;
    private int maxLevel;
    private int level;
    private Random random;

    public SkipList() {
        maxLevel = 6; // maximum number of levels
        // working with low upper-bounds and therefore also with smaller skriplists for readability and debugging.
        level = 0; // current level of SkipList
        head = new Node(Integer.MIN_VALUE, maxLevel);
        random = new Random(); // for randomness of coinflips
    }
    public int getLevel(){
        return this.level;
    }
    public Node getHead(){
        return this.head;
    }
    private int randomLevel() { // determine how often a new inserted node will be promoted to a higher level.
        int lvl = 0;
        // Keep flipping coins as long as we get heads (probability < 0.5)
        // and we haven't reached maxLevel
        while (random.nextDouble() < 0.5 && lvl < this.maxLevel) {
            lvl++;
        }
        return lvl;
    }
    // Returns an array of predecessors where predecessors[i] is the last node
    // at level i whose value is less than the search value.
    private Node[] findPredecessors(int searchValue) {
        Node[] predecessors = new Node[maxLevel + 1];
        Node curr = head;
        // begin search at highest populated level. continue until node is found on the lowest level if present.
        for (int i = this.level; i >= 0; i--) {
            // move right as long as there are values on the same level, that a smaller than the searchValue. else move down 1 level
            while (curr.forward[i] != null && curr.forward[i].value < searchValue) {
                curr = curr.forward[i];
            }
            predecessors[i] = curr;
        }

        return predecessors;
    }
    // lookup -> check if a given node is part of the Skiplist
    public Node search(int searchValue){
        Node[] preds = this.findPredecessors(searchValue);
         // At level 0 preds[0] is the greatest node thats still smaller, than the searchValue node.value < searchValue
        Node curr = preds[0].forward[0]; // expected found position
```

```
53
54          if(curr.value == searchValue && curr != null){
55              return curr; // if the current node is the wanted node than return that
    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-levek 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                  }
```

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

Datei: Node.java

```
public class Node {
    int value;
    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.

    public Node(int value, int level) {
        this.value = value;
        this.forward = new Node[level + 1]; //  declare an array of size level+1 to be able to hold level elements.
    }
    public int getValue(){
        if(this != null){
            return this.value;
        }
        else{
            return -1;
        }
    }
}
```

Datei: SkipList_test.java

```
/*scources: https://www.geeksforgeeks.org/java-util-random-nextint-java/
            ADS Skript
            https://github.com/LJWittenberg/Java-code-for-Refactoring (refactorcode unteranderen)
            https://www.tutorialspoint.com/java/util/random_nextdouble.htm
            https://stackoverflow.com/questions/2707322/what-is-null-in-java
            https://www.baeldung.com/java-skiplist#bd-5-delete-operation
            https://stackoverflow.com/questions/7630014/difficulty-throwing-nosuchelementexception-in-java
            https://rollbar.com/guides/java/how-to-throw-exceptions-in-java/

*/
import java.util.Random;
public class SkipList_test {
    // declare and initialize 3 test Skiplists.
    // 1. 1-20 inserted in assending order
    // 2. 20 random Int values between 1-50
    // 3. 1-20 inserted in reverse order
    public static void main(String[] args){
        SkipList sklst1 = new SkipList();
        SkipList sklst2 = new SkipList();
        SkipList sklst3 = new SkipList();
        SkipList sklst0 = new SkipList(); // empty List
        for(int i = 1; i <= 20; i++){
            sklst1.insert(i);
```

```
        }
        PrintSkipList(sklst1);
        Random randomgen = new Random();
        for(int i = 1; i <= 20; i++){
            sklst2.insert(randomgen.nextInt(1,50));
        }
        PrintSkipList(sklst2);
        for(int j = 20; j >= 1; j--){
            sklst3.insert(j);
        }
        PrintSkipList(sklst3);
        PrintSkipList(sklst0);
        // lookup test:
        Node searchResult = new Node(0,10);
        searchResult = searchHandler(sklst1,10); // look for value that exists.
    expected output: the wanted Node
        if(searchResult.getValue() == 0){
            System.out.println("The wanted Node does not exit in the selected SkipList
     returning std.Failuire Value: 0");
        }
        else{
            System.out.println(searchResult.getValue());
        }
        searchResult = searchHandler(sklst1,30); // look for value that does not exist
    . expected output: the wanted Node
        if(searchResult.getValue() == 0){
            System.out.println("The wanted Node does not exit in the selected SkipList
     returning std.Failuire Value: 0");
        }
        else{
            System.out.println(searchResult.getValue());
        }
        // deletion test:
        sklst0.delete(1); // deleting from empty List.
        // deleting 3 nodes from the assending list
        sklst1.delete(10);
        sklst1.delete(1);
        sklst1.delete(20);
        PrintSkipList(sklst1);
        // deleting an node that does not exists
        sklst3.delete(30);
        PrintSkipList(sklst3);
        // deleting a node in the highest populated level.
        int hgtLevel3 = sklst3.getLevel();
        Node hgtNode3 = sklst3.getHead().forward[hgtLevel3];
        int hgtNode3Val = hgtNode3.getValue();
        sklst3.delete(hgtNode3Val);
        PrintSkipList(sklst3);
        return;
    }
    // helperfunction to visualize the SkipList
    public static void PrintSkipList(SkipList sklst){
        int printLevels = sklst.getLevel();

        for(int i = printLevels; i >= 0; i--){
            Node curr = sklst.getHead();
            System.out.print("Level "+ i + ": HEAD ->");
            while(curr.forward[i] != null){
                curr = curr.forward[i];
                System.out.print(curr.getValue() + " -> ");
            }
            System.out.println("TAIL");
```

```
82
83          }
84          System.out.println();
85          return;
86      }
87      // helperfunction to avoid manuell null-handling every time SkipList.search() is
        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  }
```