

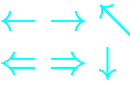
Algorithmen und Datenstrukturen (Th. Ottmann und P. Widmayer)

Folien: Skip-Listen

Autor: Sven Schuierer

Institut für Informatik
Georges-Köhler-Allee
Albert-Ludwigs-Universität Freiburg

1 Überblick



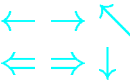
Überblick

Skip-Listen

Perfekte Skip-Listen

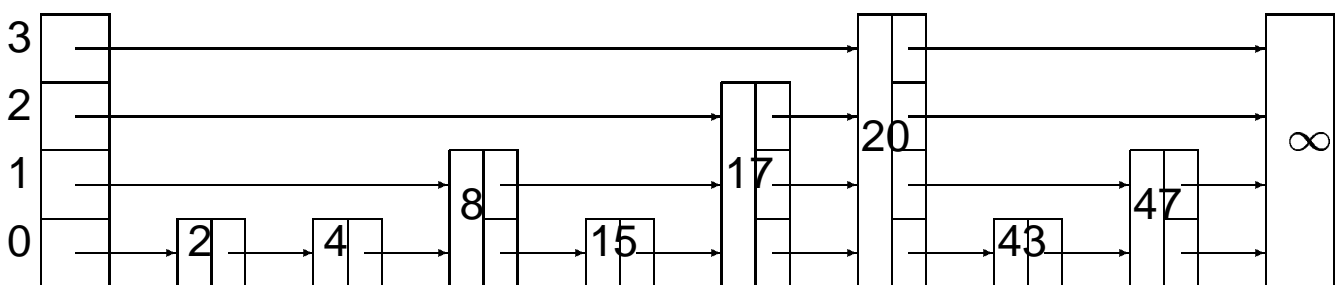
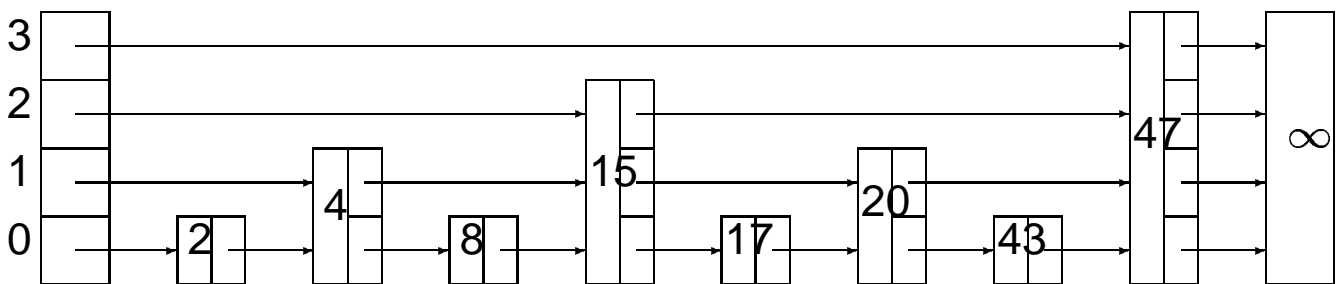
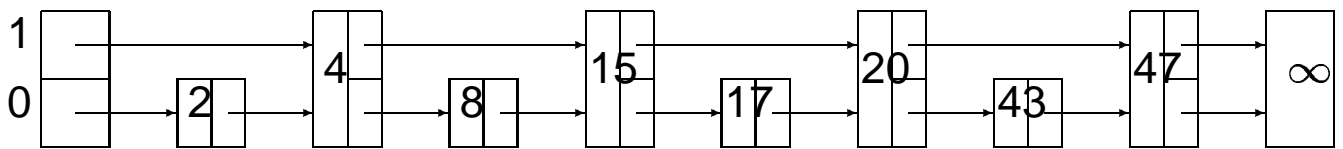
Randomisierte Skip-Listen

2 Skip-Listen

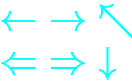


Idee:

Beschleunige die Suche in sortierter verketteter gespeicherter Liste durch zusätzliche Zeiger



3 Perfekte Skip-Listen



Perfekte Skip-Liste:

sortierte, verkettet gespeicherte Liste mit Kopfelement ohne Schlüssel und Endelement mit Schlüssel ∞ und:

Jeder 2^0 -te Knoten hat Zeiger auf den nächsten Knoten auf Niveau 0

Jeder 2^1 -te Knoten hat Zeiger auf den 2^1 entfernten Knoten auf Niveau 1

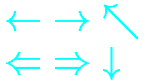
Jeder 2^2 -te Knoten hat Zeiger auf den 2^2 entfernten Knoten auf Niveau 2

⋮
⋮

Jeder 2^k -te Knoten hat Zeiger auf den 2^k entfernten Knoten auf Niveau k

$$k = \lceil \log n \rceil - 1$$

Perfekte Skip-Listen

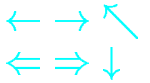


- Listenhöhe: $\log n$
- Gesamtzahl der Zeiger:

$$|kopf| + 1 + \sum_{i=0}^{\log n - 1} \frac{n}{2^i}$$

- Anzahl Zeiger pro Listenelement $\leq \log n$

Implementation von Skip-Listen



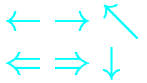
```
class skipListNode {
    /* Knotenklasse für Skip-Listen */

    int key;
    Zusätzliche Information;

    skipListNode [] next;

    /* Konstruktor */
    skipListNode (key, height) {
        this.key = key;
        this.next = new skipListNode [height+1];
    }
}
```

Implementation von Skip-Listen



```
class skipList {
    /* Implementiert eine Skip-Liste */

    maxHeight = 0;    // Max. Höhe der Liste

    skipListNode head;    // Kopf der Liste
    skipListNode tail;    // Ende der Liste

    height;            // Akt. Höhe der Liste

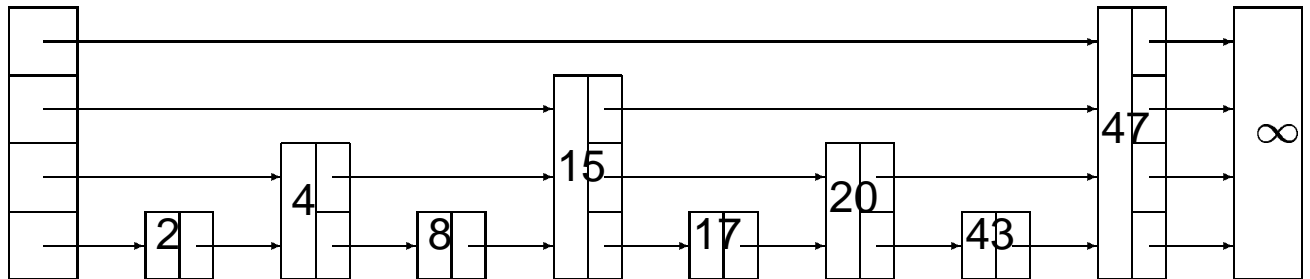
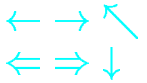
    skipListNode[] update; // Hilfsarray

    /* Konstruktor */
    skipList (maxHeight) {
        this.maxHeight = maxHeight;
        height = 0;

        head = new skipListNode (Integer.MIN_VALUE,
                                  maxHeight);
        tail = new skipListNode (Integer.MAX_VALUE,
                                  0);
        for (i = 0; i <= maxHeight; i++)
            head.next[i] = tail;

        update = new skipListNode[maxHeight+1];
    }
}
```

Suchen in Skip-Listen



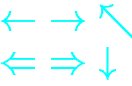
```
public SkipListNode search (key) {  
    /* liefert den Knoten p der Liste mit  
       p.key = key, falls es ihn gibt, und null sonst  
       */  
  
    p = head;  
  
    for (i = height; i >= 0; i--)  
        /* folge den Niveau-i Zeigern */  
        while (p.next[i].key < key) p = p.next[i];  
  
    /* p.key < x ≤ p.next[0].key */  
  
    p = p.next[0];  
    if (p.key == key && p != tail) return p;  
    else return null;  
}
```

Perfekte Skip-Listen:

Suchen: $O(\log n)$ Zeit

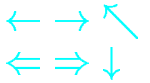
Einfügen, Entfernen: $\Omega(n)$ Zeit

4 Randomisierte Skip-Listen



- Aufgabe der starren Verteilung der Höhen der Listenelemente
- Anteil der Elemente mit bestimmter Höhe wird beibehalten
- Höhen werden gleichmäßig und zufällig über die Liste verteilt

Einfügen in randomisierte Skip-Listen



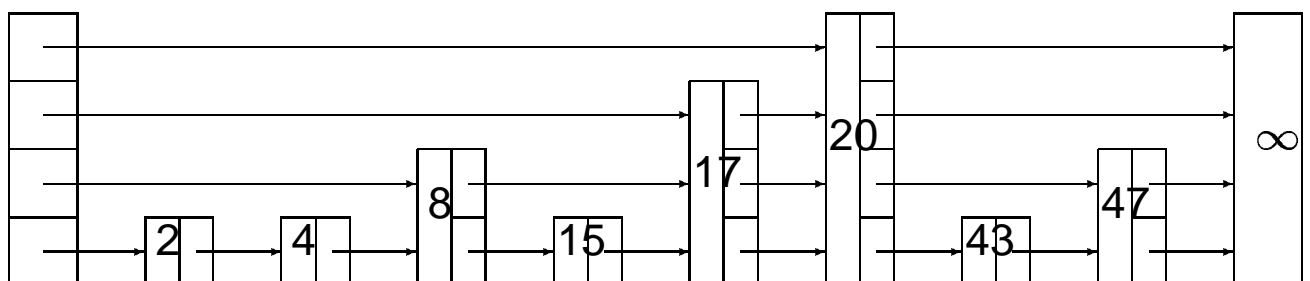
Einfügen von Schlüssel k :

1. Suche (erfolglos) nach k ; die Suche endet bei dem Knoten q mit größtem Schlüssel, der kleiner als k ist.
2. Füge neuen Knoten p mit Schlüssel k und zufällig gewählter Höhe nach Knoten q ein.
3. Wähle die Höhe von p , so daß für alle i gilt:

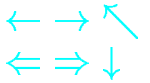
$$P(\text{Höhe von } p = i) = \frac{1}{2^{i+1}}$$

4. Sammele alle Zeiger, die über die Einfügestelle hinwegweisen in einem Array und füge p in alle Niveau i Listen mit $0 \leq i \leq \text{Höhe von } p$, ein.

Beispiel: Einfügen von Schlüssel 16:



Einfügen in randomisierte Skip-Listen



```
public void insert (key) {
    /* fügt den Schlüssel key in Skip-Liste ein */

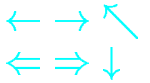
    /* Suche den Schlüssel key und speichere in
       update[i] den Zeiger auf Niveau i unmittelbar
       vor key */
    p = head;
    for (i = height; i >= 0; i--) {
        while (p.next[i].key < key) p = p.next[i];
        update[i] = p;
    }

    p = p.next[0];
    if (p.key == key) return; // Schlüssel vorhanden

    newheight = randheight ();
    if (newheight > height) {
        /* Höhe der Skip-Liste anpassen */
        for (i = height + 1; i <= newheight; i++)
            update[i] = head;
        height = newheight;
    }

    p = new skipListNode (key, newheight);
    for (i = 0; i <= newheight; i++) {
        /* füge p in Niveau i nach update[i] ein */
        p.next[i] = update[i].next[i];
        update[i].next[i] = p;
    }
}
```

Erzeugen zufälliger Höhen

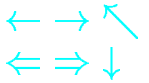


```
int randheight () {  
    /* liefert eine zufällige Höhe zwischen 0 und  
       maxHeight */  
  
    height = 0;  
  
    while (rand () % 2 == 1 && height < maxHeight)  
        height++;  
  
    return height;  
}
```

Es gilt:

$$P(\text{randheight} = i) = \frac{1}{2^{i+1}}$$

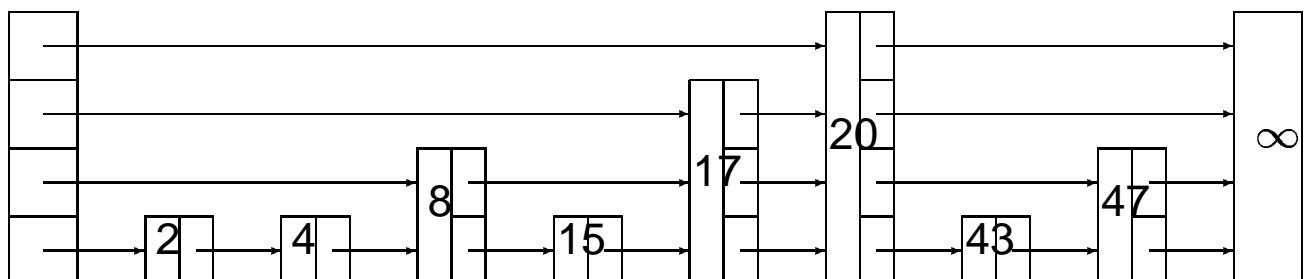
Entfernen eines Schlüssels



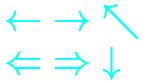
Entfernen eines Schlüssels k :

1. Suche (erfolgreich) nach k
2. Entferne Knoten p mit $p.key = k$ aus allen Niveau i Listen, mit $0 \leq i \leq$ Höhe von p , und adjustiere ggfs. die Listenhöhe.

Beispiel: Entfernen von Schlüssel 20:



Entfernen eines Schlüssels



```
public void delete (key) {
    /* entfernt den Schlüssel key aus der Skip-Liste
       */
    p = head;
    for (i = height; i >= 0; i--) {
        /* folge den Niveau-i Zeigern */
        while (p.next[i].key < key) p = p.next[i];
        update[i] = p;
    }

    p = p.next[0];
    if (p.key != key) return; /* Schlüssel nicht
                               vorhanden */

    for (i = 0; i < p.next.length; i++) {
        /* entferne p aus Niveau i */
        update[i].next[i] = update[i].next[i].next[i];
    }

    /* Passe die Höhe der Liste an */
    while (height >= 0 && head.next[height] == tail)

        height--;
}
```

Implementation von Skip-Listen, 6, 7

Perfekte Skip-Listen, 4, 5

Randomisierte Skip-Listen, 9

Skip-Listen, 3

Suchen in Skip-Listen, 8

