INSTITUT FÜR INFORMATIK FREIBURG

Vorlesung Informatik 2
Algorithmen und Datenstrukturen

(07 - Skiplisten)

Prof. Dr. Susanne Albers

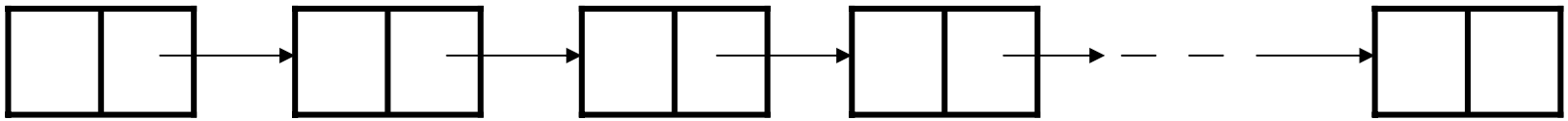
Skiplisten

1. Perfekte Skiplisten
2. Randomisierte Skiplisten
3. Verhalten von randomisierten Skiplisten

1. (Perfekte) Skiplisten

Mögliche Implementationen von Wörterbüchern

a) Verkettete lineare Listen



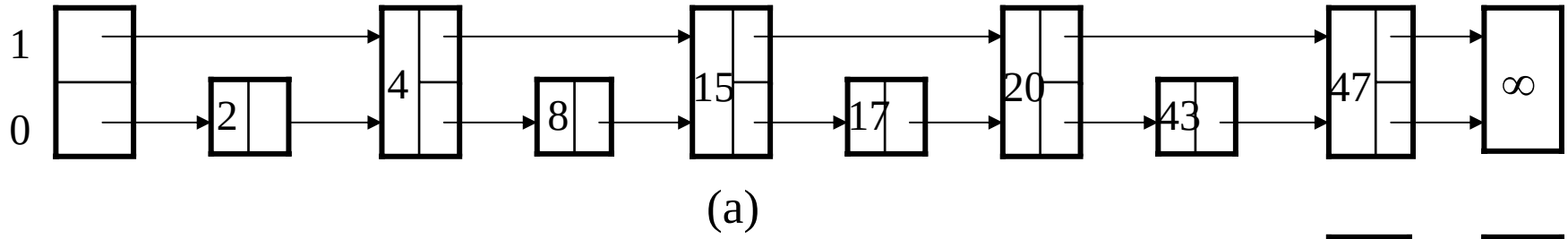
Suchen, Einfügen, Entfernen: O(n) Schritte

b) (Balancierte) Suchbäume

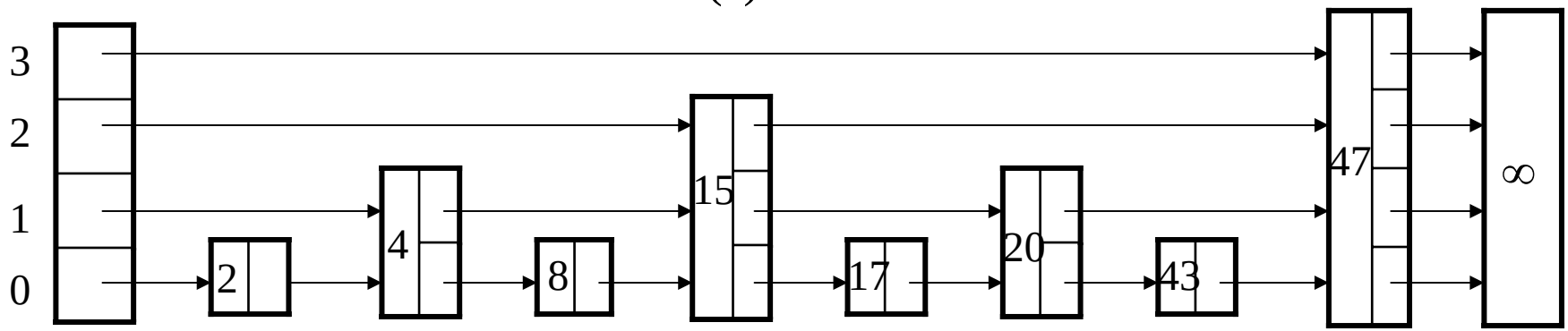
Suchen, Einfügen, Entfernen: O(log n) Schritte

Skiplisten-Idee

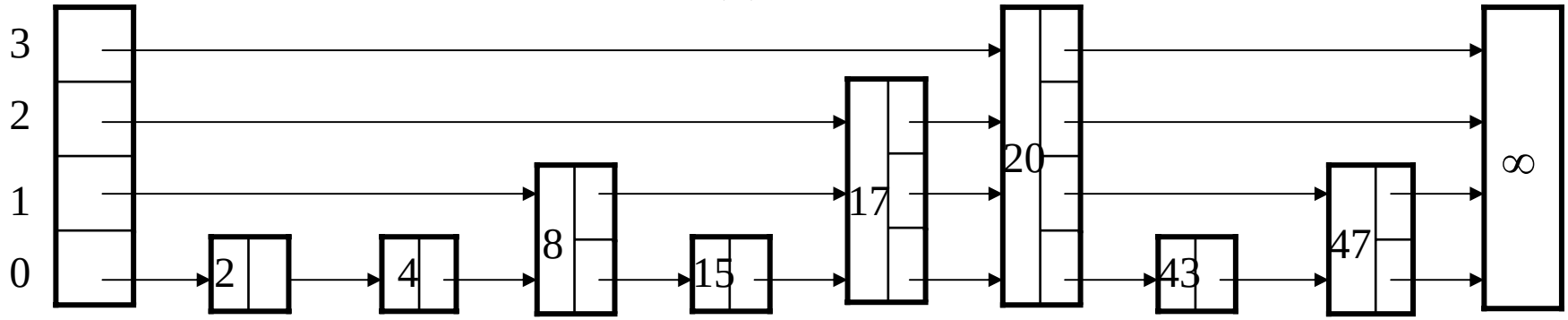
Beschleunige die Suche in sortierter verkettet gespeicheter Liste durch zusätzliche Zeiger



(a)



(b)



(c)

Perfekte Skiplisten

Perfekte Skipliste: sortierte, verkettet gespeichete Liste mit Kopfelement ohne Schlüssel und Endelement mit Schlüssel ∞ und:

Jeder 2^0-te Knoten hat Zeiger auf nächsten Knoten auf Niveau 0
Jeder 2^1-te Knoten hat Zeiger auf 2^1 Positionen entfernten Knoten auf Niveau 1
Jeder 2^2-te Knoten hat Zeiger auf 2^2 Positionen entfernten Knoten auf Niveau 2
...
...
Jeder 2^k-te Knoten hat Zeiger auf 2^k Positionen entfernten Knoten auf Niveau k

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

Perfekte Skiplisten

- Listenhöhe: log n
- Gesamtzahl der Zeiger:

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

- Anzahl Zeiger pro Listenelement $\leq$ log n

Implementation von Skiplisten

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

   protected int key;
   protected Object information;

   protected skipListNode [] next;

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

Implementation von Skiplisten

```
class skipList {
   /* Implementiert eine Skip-Liste */
   public int maxHeight = 0; // Max. Höhe der Liste
   private skipListNode head; // Kopf der Liste
   private skipListNode tail; // Ende der Liste
   private int height; // Akt. Höhe der Liste
   private skipListNode[] update; // Hilfsarray

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

      head = new skipListNode (Integer.MIN VALUE,
                               maxHeight);
      tail = new skipListNode (Integer.MAX VALUE,
                               maxHeight);
      for (int i = 0; i <= maxHeight; i++)
           head.next[i] = tail;
      update = new skipListNode[maxHeight+1];
}
```

Suchen in Skiplisten

```
public skipListNode search (int key) {
   /* liefert den Knoten p der Liste mit p.key = key,
      falls es ihn gibt, und null sonst */

   skipListNode p = head;

   for (int i = height; i >= 0; i--)
      /* folge den Niveau-i Zeigern */
      while (p.next[i].key < key) p = p.next[i];

   /* p.key < key <= p.next[0].key */
   p = p.next[0];
   if (p.key == key && p != tail) return p;
   else return null;
}
```

Perfekte Skiplisten:
Suchen: $O(\log n)$ Zeit
Einfügen, *Entfernen*: $\Omega(n)$ Zeit

2. Randomisierte Skiplisten

- 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 Skiplisten

Einfügen von *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. Sammle alle Zeiger, die über die Einfügestelle hinwegweisen in einem Array und füge *p* in alle Niveau *i* Listen mit $0 \leq i \leq$ Höhe von *p*, ein.

Einfügen in randomisierte Skiplisten

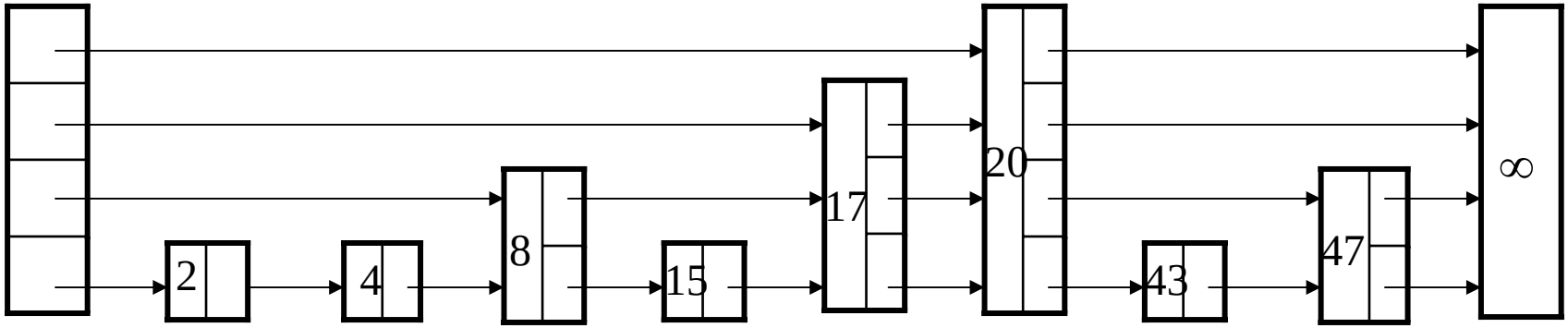
```
public void insert (int key) {
   /* fügt den Schlüssel key in die Skip-Liste ein */
   skipListNode p = head;
   for (int i = height; i >= 0; i--) {
      while (p.next[i].key < key) p = p.next[i];
      update[i] = p;
   }
   p = p.next[0];
   if (p.key == key) return; // Schüssel vorhanden

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

   p = new skipListNode (key, newheight);
   for (int 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;
   }
}
```

Beispiel für Einfügen

Schlüssel 16:



Erzeugen zufälliger Höhen

```
private int randheight () {
  /* liefert eine zufällige Höhe zwischen 0 und
     maxHeight */

  int height = 0;

  while (rand () % 2 == 1)
     height++;

  return height;
}
```

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

3. Verhalten von randomisierten Skiplisten

1. Unabhängig von der Erzeugungshistorie
2. Erwartete Anzahl von Zeigern in einer Liste der Länge n ist 2n
3. Erwartete Höhe einer Liste mit n Elementen ist in O(log n) (genauer: $\leq 2 \log n + 2$)
4. Erwartete Suchkosten für die Suche nach einem Element in einer Liste mit n Elementen sind in O(log n).
5. Einfügen und Entfernen von Elementen in Liste mit n Elementen in erwarteter Zeit O(log n) möglich.

Erwartete Anzahl von Zeigern

z_i = # Zeiger von Element i

Z = # Zeiger in Liste mit n Elementen = $\sum_{i=1}^{n} z_i$

$$E(Z) = \sum_{i=1}^{n} E(z_i)$$

$$E(z_j) = \sum_{j=1}^{\infty} (1/2^j)j$$

$$E(Z) = \sum_{i=1}^{n} 2$$

Erwartete Höhe einer Liste mit n Elementen

$$h + 1 = \max_{1 \le i \le n} z_i$$

$$P(z_i > t) = \sum_{j=t+1}^{\infty} (1/2)^j$$

$$P(\max_{1 \le i \le n} z_i > t) \le P(z_1 > t) + P(z_2 > t) + \ldots + P(z_n > t)$$

$$P(h = j) \le P(h > j - 1)$$

$$E(h) = \sum_{j=0}^{\infty} jP(h = j)$$

$$= \sum_{j=0}^{2\log n+1} jP(h = j) \sum_{j=2\log n+2}^{\infty} jP(h = j)$$

Entfernen eines Schlüssels

Entfernen von *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.

Entfernen eines Schlüssels

```
public void delete (int key) {
   /* enfernt den Schlüssel key aus der Skip-Liste */
   skipListNode p = head;
   for (int 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 (int 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--;
}
```