Andere Methoden zur Kollisionsauflösung

Neben Verkettung gibt es viele andere Methoden, Kollision zu behandeln:

- Linear Probing
- Quadratic Probing
- Double Hashing
- Sekundäre Hashtabelle

und viele andere...

Universelle Familien von Hashfunktionen

Sei $U = \{0, \ldots, p-1\}$, wobei p eine Primzahl ist.
Es sei $h_{a,b}(x) = ((ax + b) \bmod p) \bmod m$.
Wir definieren

$$\mathcal{H} = \{ h_{a,b} \mid 1 \leq a < p,\ 0 \leq b < p \}$$

Theorem

$\mathcal{H}$ ist eine universelle Familie von Hashfunktionen.

Es seien $x, y \in \{0, \ldots, p-1\}$, $x \neq y$.
Wir wollen zunächst zeigen, daß die Funktion

$$f : (a, b) \mapsto (ax + b \bmod p, ay + b \bmod p)$$

für $a, b \in \{0, \ldots, p-1\}$ injektiv und somit auch bijektiv ist.

$$\begin{aligned}
&(ax + b \bmod p, ay + b \bmod p) = (a'x + b' \bmod p, a'y + b' \bmod p) \\
\Leftrightarrow &(ax + b - b' \bmod p, ay + b - b' \bmod p) = (a'x \bmod p, a'y \bmod p) \\
\Leftrightarrow &(b - b' \bmod p, b - b' \bmod p) = ((a' - a)x \bmod p, (a' - a)y \bmod p) \\
\Leftrightarrow &a' = a \wedge b' = b
\end{aligned}$$

Nach wie vor gelte $x, y \in \{0, \ldots, p-1\}$, $x \neq y$.
Für wieviele Paare (a, b) haben $c_x := ax + b \bmod p$ und $c_y := ay + b \bmod p$ den gleichen Rest modulo m?

Wir haben auf der letzten Folie bewiesen, daß sich für jedes Paar (a, b) ein eindeutiges Paar (c_x, c_y) ergibt. Für ein festes c_x gibt es nur

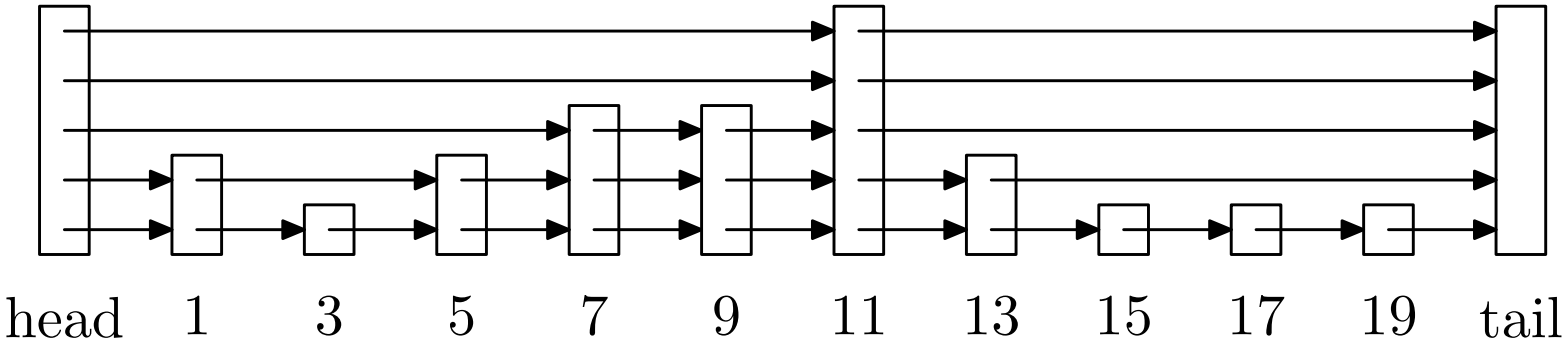
$$\lceil p/m \rceil - 1 = \left\lfloor \frac{p+m-1}{m} \right\rfloor - 1 \leq \frac{p-1}{m}$$

viele mögliche Werte von c_y mit $c_x \equiv c_y \mod m$ und $c_x \neq c_y$.

Weil p verschiedene Werte für c_x existieren, gibt es insgesamt höchstens $p(p-1)/m$ Paare der gesuchten Art.

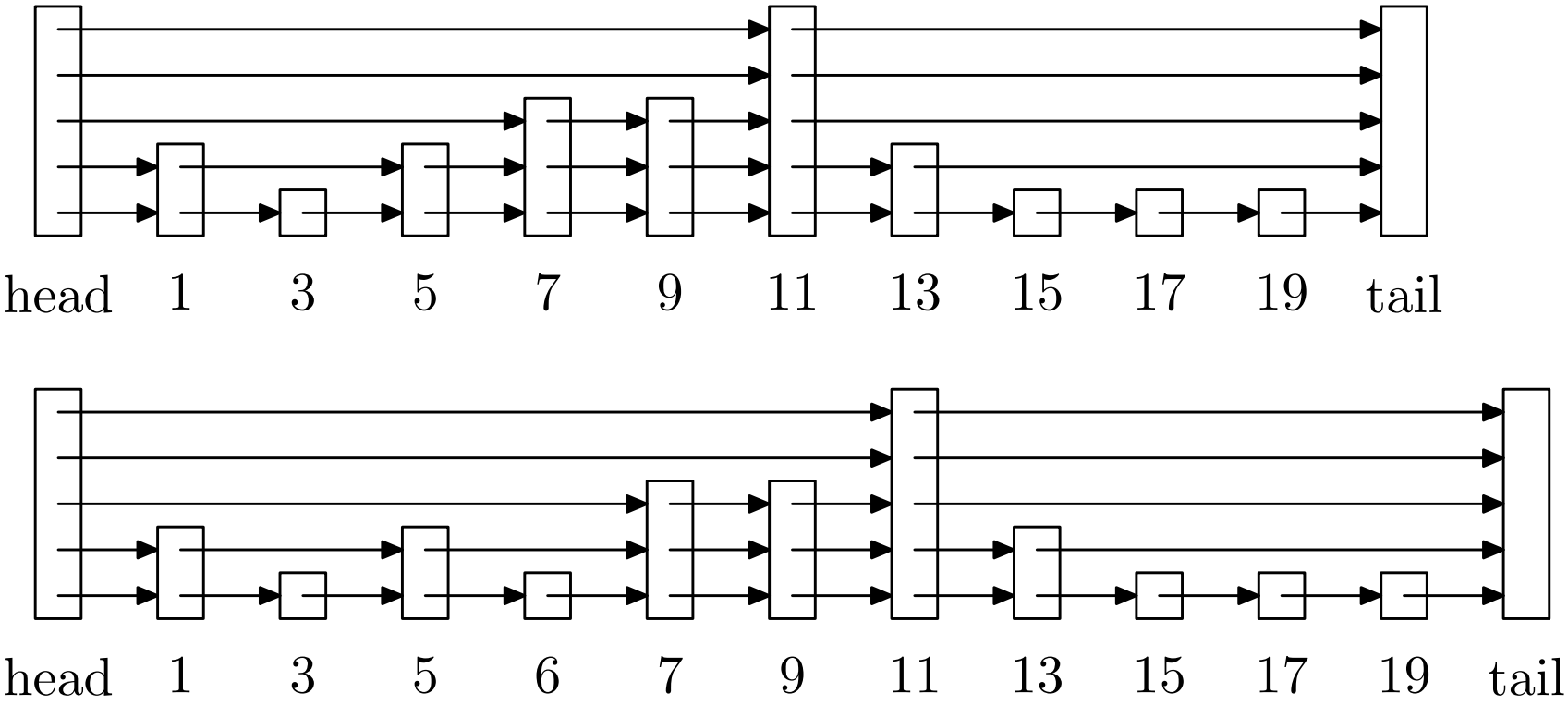
$$\frac{|\{\, h \in \mathcal{H} \mid h(x) = h(y) \,\}|}{|\mathcal{H}|} \leq \frac{p(p-1)/m}{p(p-1)} \leq \frac{1}{m}$$

Skip-Lists



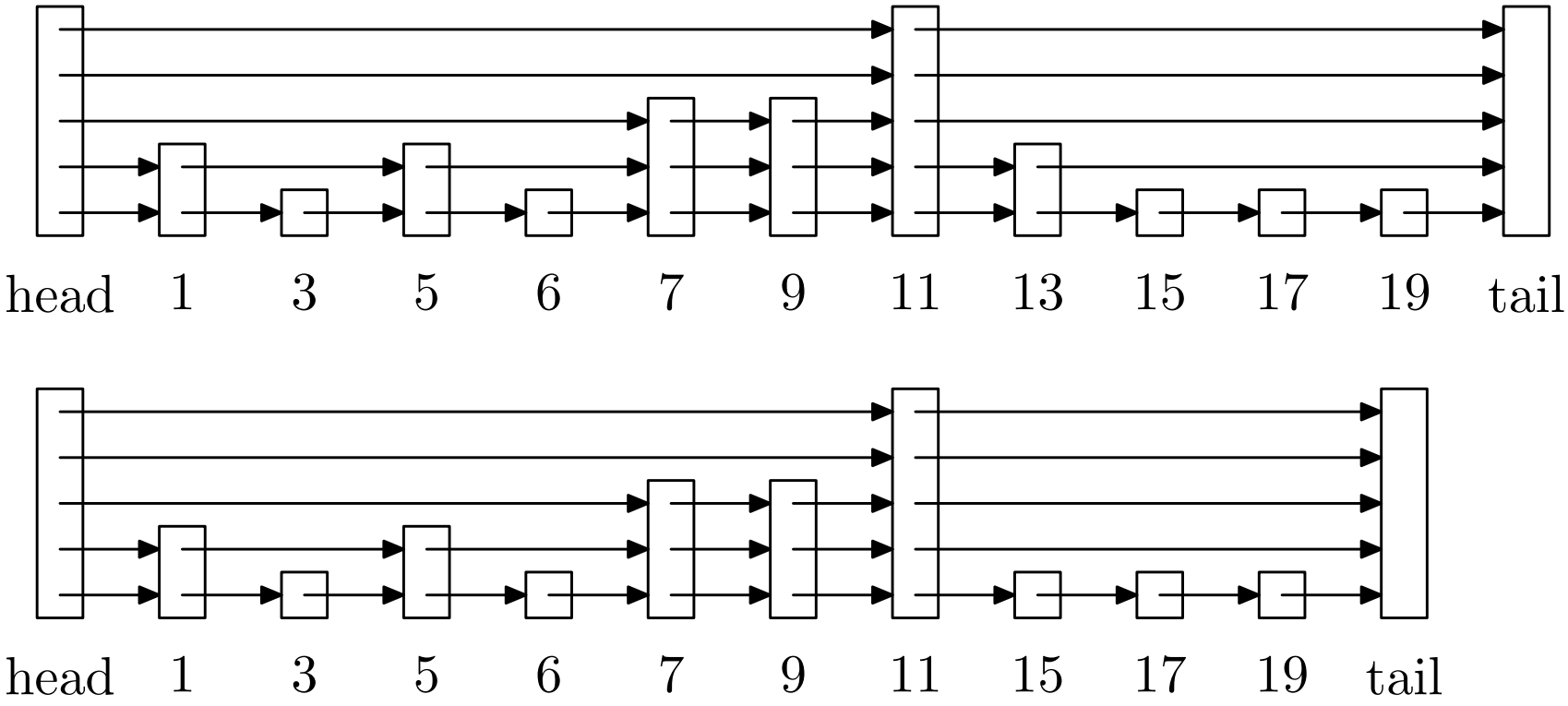
1. Schlüssel nach Größe einsortiert.
2. Es gibt beliebig viele Listen.
3. Anzahl ist geometrisch verteilt.
4. Anzahl ändert sich nicht.
5. Suchen: Von oben nach unten.

Skip-Lists – Einfügen



Einfügen des Elements 6.

Skip-Lists – Löschen



Löschen des Elements 13.

Java

```
public class Skiplist<K extends Comparable<K>, D>
                    extends Dictionary<K, D> {
  int size;
  double prob = 0.5;
  Random rand;
  class Node {
    Array<Node> succ;
    K key;
    D data;
  }
  Node head, tail;
```

Java

```
public Skiplist() {
    head = new Node();
    tail = new Node();
    head.succ = new Array⟨Node⟩();
    tail.succ = new Array⟨Node⟩();
    head.succ.set(0, tail);
    size = 0;
    rand = new Random();
}
```

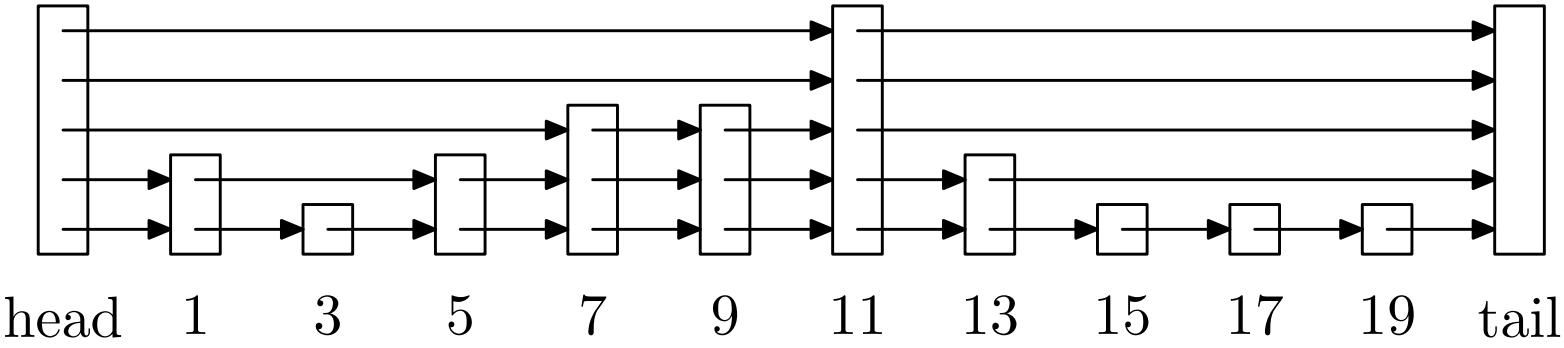
Es gibt noch weitere Konstruktoren, die es erlauben, den Zufallsgenerator und p vorzugeben.

Java

```
public D find(K k) {
  Node n = findnode(k);
  if(n ≡ null) return null;
  return n.data;
}
```

Java

```
public boolean iselement(K k) {
  return findnode(k) ≠ null;
}
```



Java

```
Node findnode(K k) {
  Node n = head;
  for(int i = head.succ.size() − 1; i ≥ 0; i−−)
    while(n.succ.get(i) ≠ tail &&
          n.succ.get(i).key.compareTo(k) ≤ 0)
            n = n.succ.get(i);
  if(n ≡ head ||!n.key.equals(k)) return null;
  return n;
}
```

Java

```
public void insert(K k, D d) {
  delete(k);
  int s = 1;
  while(rand.nextDouble() ≥ prob) s++;
  Node n = new Node();
  n.key = k; n.data = d;
  n.succ = new Array⟨Node⟩(s);
  Node m = head;
  for(int i = 0; i < s; i++)
    if(i ≥ head.succ.size()) head.succ.set(i, tail);
  for(int i = s − 1; i ≥ 0; i−−) m = insert_on_level(i, m, n);
  size++;
}
```

Java

```
Node insert_on_level(int i, Node m, Node n) {
  while(m.succ.get(i) ≠ tail
        && m.succ.get(i).key.compareTo(n.key) < 0) {
      m = m.succ.get(i);
  }
  n.succ.set(i, m.succ.get(i));
  m.succ.set(i, n);
  return m;
}
```

Java

```
public void delete(K k) {
  Node n = head;
  if(iselement(k)) size--; else return;
  for(int i = head.succ.size() - 1; i ≥ 0; i--) {
    while(n.succ.get(i) ≠ tail &&
          n.succ.get(i).key.compareTo(k) < 0) n = n.succ.get(i);
    if(n.succ.get(i) ≠ tail && n.succ.get(i).key.equals(k))
      n.succ.set(i, n.succ.get(i).succ.get(i));
  }
}
```