9.词典

（xa2）跳转表：算法

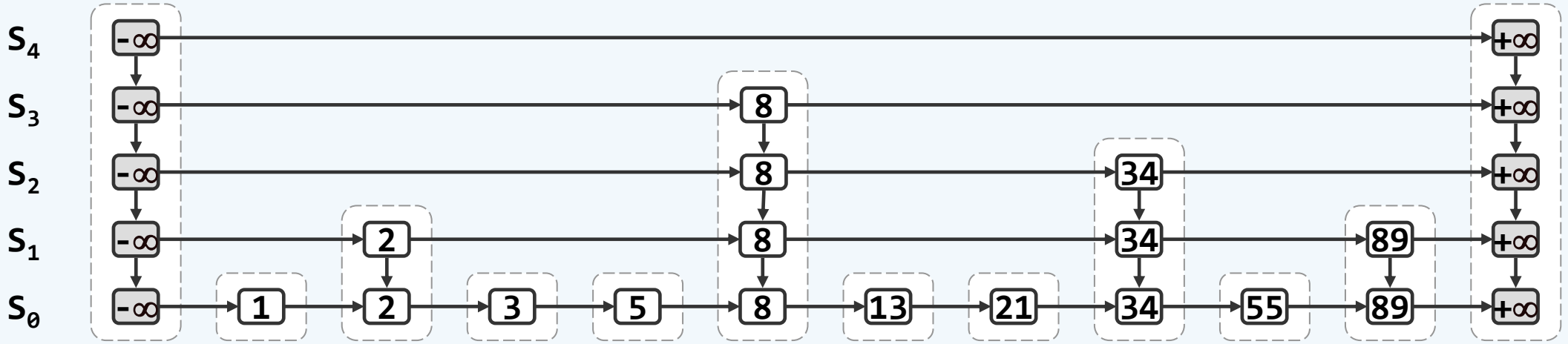
只见参仙老怪梁子翁笑嘻嘻的站起身来，向众人拱了拱手，缓步走到庭中，忽地跃起，左足探出，已落在欧阳克插在雪地的筷子之上，拉开架子，...，把一路巧打连绵的“燕青拳”使了出来，脚下纵跳如飞，每一步都落在竖直的筷子之上。

邓俊辉

deng@tsinghua.edu.cn

查找：策略与算法

❖ 减而治之：由粗到细 = 由高至低



❖ 实例：成功（21、34、1、89），失败（80、0、99）

❖ 观察：查找时间取决于横向、纵向的累计跳转次数

那么，是否可能因层次过多，首先导致纵向跳转过多？

查找：实现

```
template <typename K, typename V> bool Skiplist<K, V>::skipSearch(
    ListNode< Quadlist < Entry< K, V > > * > * & qlist, //从指定层qlist的
    QuadlistNode< Entry< K, V > > * & p, K & k ) { //首节点p出发，向右、向下查找k
    while ( true ) { //在每一层从前向后查找，直到出现更大的key，或者溢出至trailer
       while ( p->succ && ( p->entry.key <= k ) ) p = p->succ; p = p->pred;
       if ( p->pred && ( k == p->entry.key ) ) return true; //命中则成功返回
       qlist = qlist->succ; //否则转入下一层
       if ( ! qlist->succ ) return false; //若已到穿透底层，则意味着失败；否则...
       p  =  p->pred ? p->below : qlist->data->first(); //转至当前塔的下一节点
    } //确认：无论成功或失败，返回的p均为其中不大于e的最大者？
} //体会：得益于哨兵的设置，哪些环节被简化了？
```

查找：纵向跳转/层高

❖ 引理： 随着k的增加，S_k 为空 的概率急剧 上升

准确地，$Pr(|S_k| = 0) = (1 - 1/2^k)^n \geq 1 - n/2^k$

❖ 于是： 随着k的增加，S_k 非空 的概率急剧 下降

准确地，$Pr(|S_k| > 0) \leq n \times 2^{-k}$

❖ 推论： 跳转表高度 $h = O(\log n)$ 的概率 极大

❖ 比如： 第 $k = 3\log n$ 层为空（当且仅当 $h < k$）的概率为

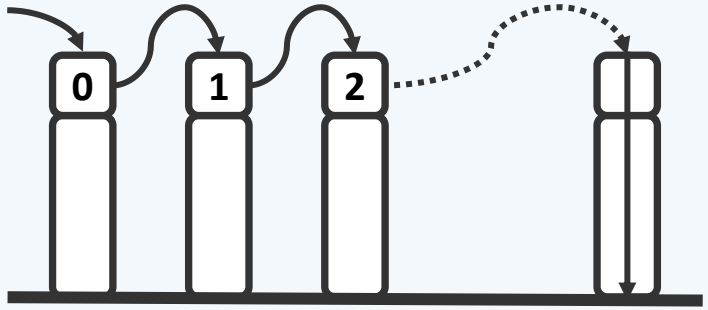
$Pr(h < k) \geq Pr(|S_k| = 0) \geq 1 - n/2^k = 1 - n/n^3 = 1 - n^{-2} \mapsto 1$

❖ 结论：查找过程中，纵向 跳转的次数，累计不过 expected-$O(\log n)$

查找：横向跳转/紧邻塔顶

❖ 那么： 横向跳转是否可能很多次？比如ω(logn)，甚至Ω(n)？

❖ 观察： 在同一水平列表中，横向跳转所经节点必然依次紧邻，而且每次抵达都是塔顶



❖ 于是： 若将沿同层列表跳转的次数记作Y，则有几何分布：

$$Pr(Y = k) = (1 - p)^k \cdot p$$

❖ 定理： $E(Y) = (1 - p) / p = (1 - 0.5) / 0.5 =$ 1 次

同层列表中紧邻的塔顶节点，平均不过1 + 1 = 2 个

❖ 推论： 查找过程中横向跳转所需时间 ≤ expected(2h) = expected-O(logn)

❖ 结论： 跳转表的每次查找可在expected-O(logn)时间内完成

插入：算法实现

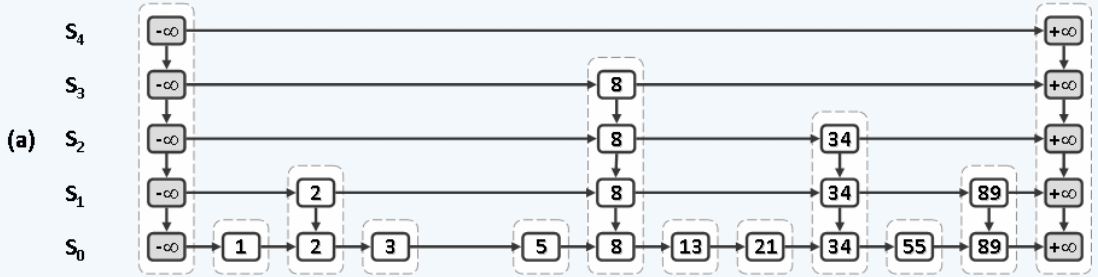
```
template <typename K, typename V> bool Skiplist< K, V >::put( K k, V v ) {
   Entry< K, V > e = Entry< K, V >( k, v ); //将被随机地插入多个副本的新词条
   if ( empty() ) insertAsFirst( new Quadlist< Entry< K, V > > ); //首个Entry
   ListNode< Quadlist< Entry< K, V > > * > * qlist = first(); //从顶层列表的
   QuadlistNode< Entry< K, V > > * p = qlist->data->first(); //首节点开始
   if ( skipSearch( qlist, p, k ) ) //查找适当的插入位置——若已有雷同词条，则
      while ( p->below ) p = p->below; //强制转到塔底
   qlist = last();
   QuadlistNode< Entry< K, V > > * b = qlist->data->insertAfterAbove( e, p );
   /* ... 以下，紧邻于p的右侧，以新节点b为基座，自底而上逐层长出一座新塔 ... */
```

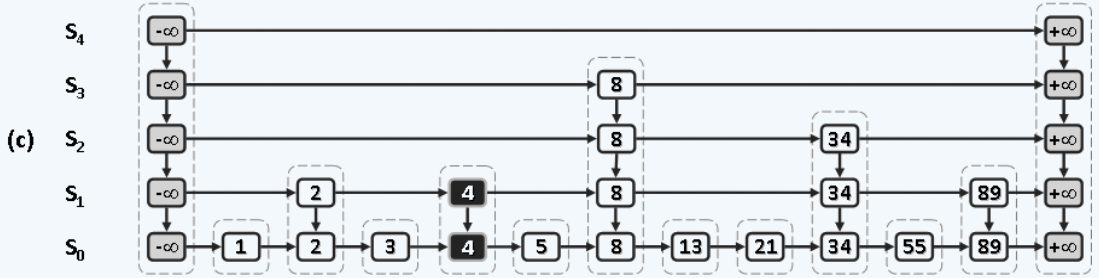
插入：算法实现

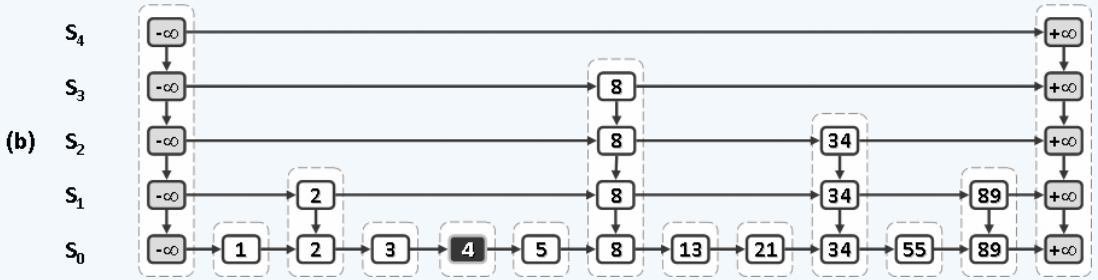
```
while ( rand() & 1 ) { //经投掷硬币，若新塔需再长高，则先找出不低于此高度的...
   while ( qlist->data->valid(p) && ! p->above ) p = p->pred; //最近前驱
   if ( ! qlist->data->valid(p) ) { //若该前驱是header
      if ( qlist == first() ) //且当前已是最顶层，则意味着必须
         insertAsFirst( new Quadlist< Entry< K, V > > ); //先创建新层，再
      p = qlist->pred->data->first()->pred; //将p转至上一层的header
   } else p = p->above; ////否则，可径自将p提升至该高度
   qlist = qlist->pred; //上升一层，并在该层将新节点
   b = qlist->data->insertAfterAbove( e, p, b ); //插至p之后、b之上
} //while ( rand() & 1 )
return true; //Skiplist允许重复元素，故插入必成功
}
```

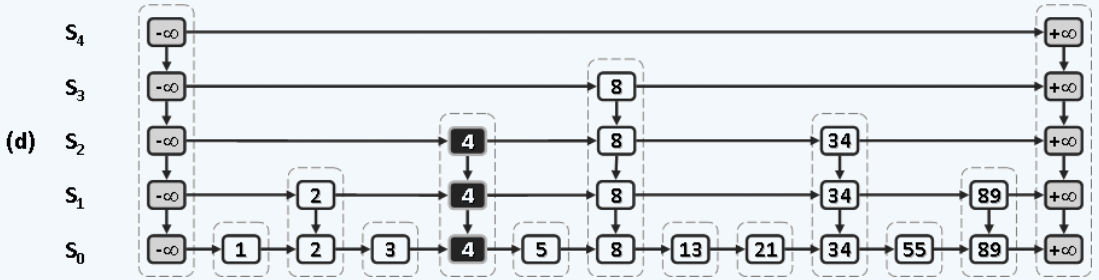
插入：实例

❖ 实例：一般（4、20、40），边界（0、99）









删除：算法实现

```
//插入的逆过程

template <typename K, typename V> bool Skiplist< K, V >::remove( K k ) {
   if ( empty() ) return false; //空表
   ListNode< Quadlist< Entry< K, V > > * > * qlist = first(); //从顶层Quadlist
   QuadlistNode< Entry< K, V > > * p = qlist->data->first(); //的首节点开始
   if ( ! skipSearch( qlist, p, k ) ) //目标词条不存在，则
      return false; //直接返回
   /* ... */
```

删除：算法实现

```
do { //若目标词条存在，则逐层拆除与之对应的塔
   QuadlistNode< Entry< K, V > > * lower = p->below; //记住下一层节点
   qlist->data->remove(p); //删除当前层的节点后，再
   p = lower; qlist = qlist->succ; //转入下一层
} while ( qlist->succ ); //如上不断重复，直到塔基
while ( ! empty() && first()->data->empty() ) //逐一地
   List::remove( first() ); //清除已可能不含词条的顶层Quadlist
return true; //删除操作成功完成
} //体会：得益于哨兵的设置，哪些环节被简化了？
```