

词典

跳转表：插入与删除

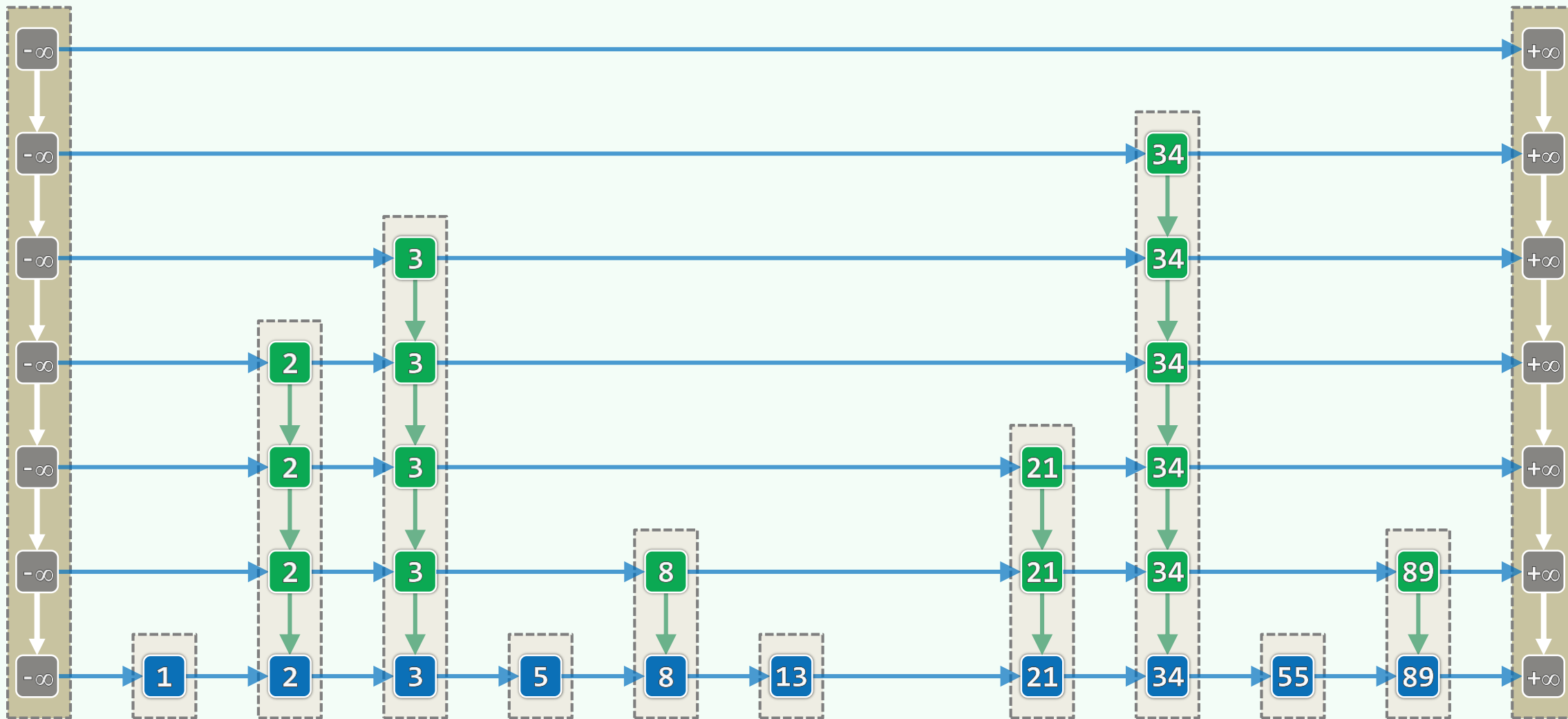
09-G3

如果一个人遇到不可解之事，把脑子想穿了，也找不到其中的原因，怎么办呢？  
他或许会去庙里烧香，把自己的难题交给算命先生，听任他们的摆布。

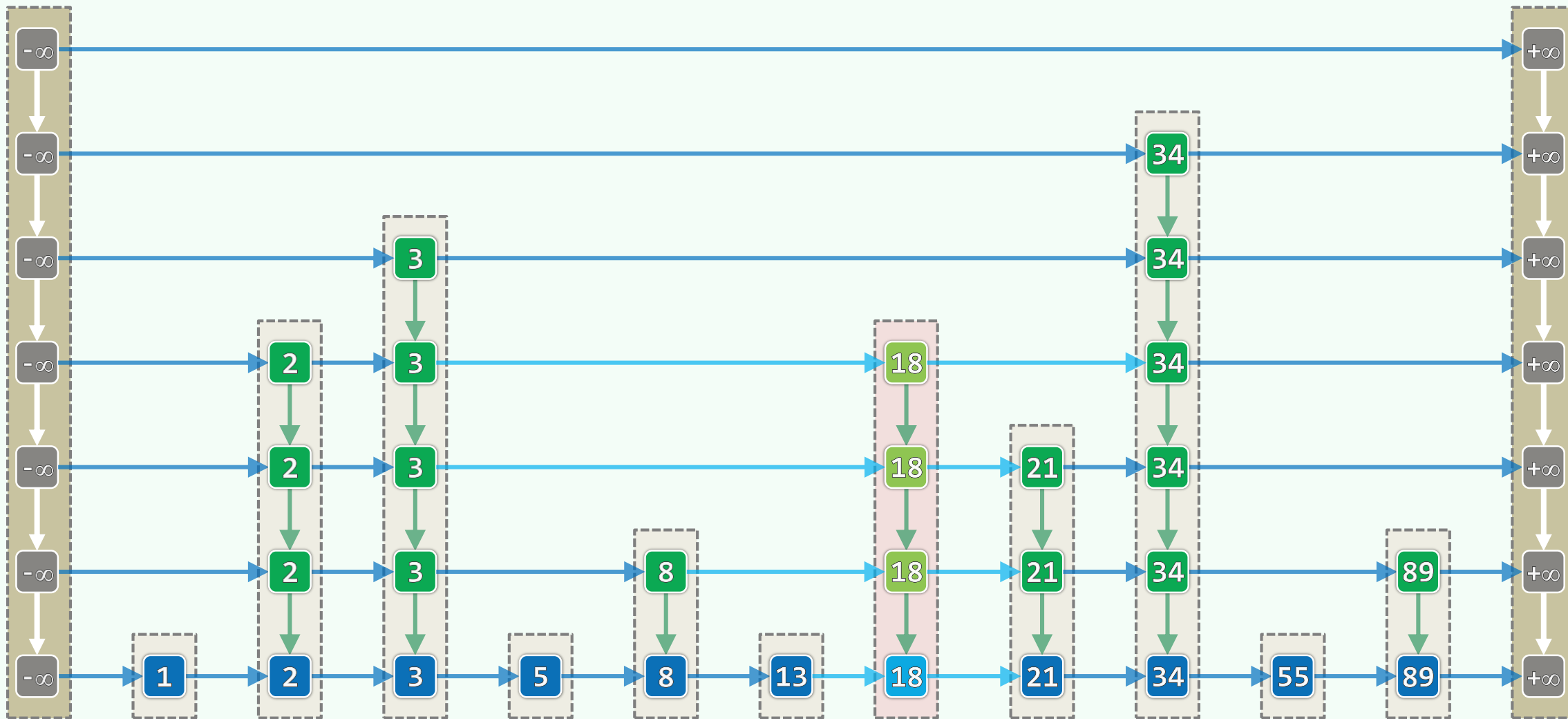
邓俊辉

deng@tsinghua.edu.cn

## 实例：put(18)之前 = remove(18)之后



## 实例：put(18)之后 = remove(18)之前



## 插入算法：整体

```
template <typename K, typename V> bool Skiplist<K, V>::put( K k, V v ) {  
    Entry< K, V > e = Entry< K, V >( k, v ); //待插入的词条（将被同一塔中所有节点共用）  
    QNodePosi< Entry<K, V> > p = search( k ); //查找插入位置：新塔将紧邻其右，逐层生长  
    ListNodePosi< Quadlist< Entry<K, V> >* > qlist = last(); //首先在最底层  
    QNodePosi< Entry<K, V> > b = qlist->data->insert( e, p ); //创建新塔的基座  
    while ( rand() & 1 ) {  
        /* ... 建塔 ... */  
    }  
    return true; //Dictionary允许重复元素，故插入必成功  
} //体会：得益于哨兵的设置，哪些环节被简化了？
```

## 插入算法：建塔

```
while ( rand() & 1 ) { //经投掷硬币，若新塔需再长高，则
    while ( p->pred && !p->above ) p = p->pred; //找出不低于此高度的最近前驱
    if ( !p->pred && !p->above ) { //若该前驱是header，且已是最顶层，则
        insertAsFirst( new Quadlist< Entry<K, V> > ); //需要创建新的一层
        first()->data->header->below = qlist->data->header;
        qlist->data->header->above = first()->data->header;
    }
    p = p->above; qlist = qlist->pred; //上升一层，并在该层
    b = qlist->data->insert( e, p, b ); //将新节点插入p之后、b之上
}
```

## 删除算法 (1/3): 预备

```
template <typename K, typename V> bool Skiplist<K, V>::remove( K k ) {  
  
    QNodePosi< Entry<K, V> > p = search( k ); //查找目标词条  
  
    if ( !p->pred || (k != p->entry.key) ) return false; //若不存在, 直接返回  
  
    ListNodePosi< Quadlist< Entry<K, V> >* > qlist = last(); //从底层Quadlist开始  
  
    while ( p->above ) { qlist = qlist->pred; p = p->above; } //升至塔顶  
  
    /* ... 2. 拆塔 ... */  
  
    /* ... 3. 删除空表 ... */  
  
    return true; //删除成功  
  
} //体会: 得益于哨兵的设置, 哪些环节被简化了?
```

## 删除算法 (2/3): 拆塔

```
template <typename K, typename V> bool Skiplist<K, V>::remove( K k ) {
```

```
    /* ... 1. 预备 ... */
```

```
    do { QNodePosi< Entry<K, V> > lower = p->below; //记住下一层节点, 并
```

```
        qlist->data->remove( p ); //删除当前层节点, 再
```

```
        p = lower; qlist = qlist->succ; //转入下一层
```

```
    } while ( qlist->succ ); //直到塔基
```

```
    /* ... 3. 删除空表 ... */
```

```
    return true; //删除成功
```

```
} //体会: 得益于哨兵的设置, 哪些环节被简化了?
```

## 删除算法 (3/3): 删除空表

```
template <typename K, typename V> bool Skiplist<K, V>::remove( K k ) {  
  
    /* ... 1. 预备 ... */  
  
    /* ... 2. 拆塔 ... */  
  
    while ( (1 < height()) && (first()->data->_size < 1) ) { //逐层清除  
  
        List::remove( first() );  
  
        first()->data->header->above = NULL;  
  
    } //已不含词条的Quadlist (至少保留最底层空表)  
  
    return true; //删除成功  
  
} //体会: 得益于哨兵的设置, 哪些环节被简化了?
```