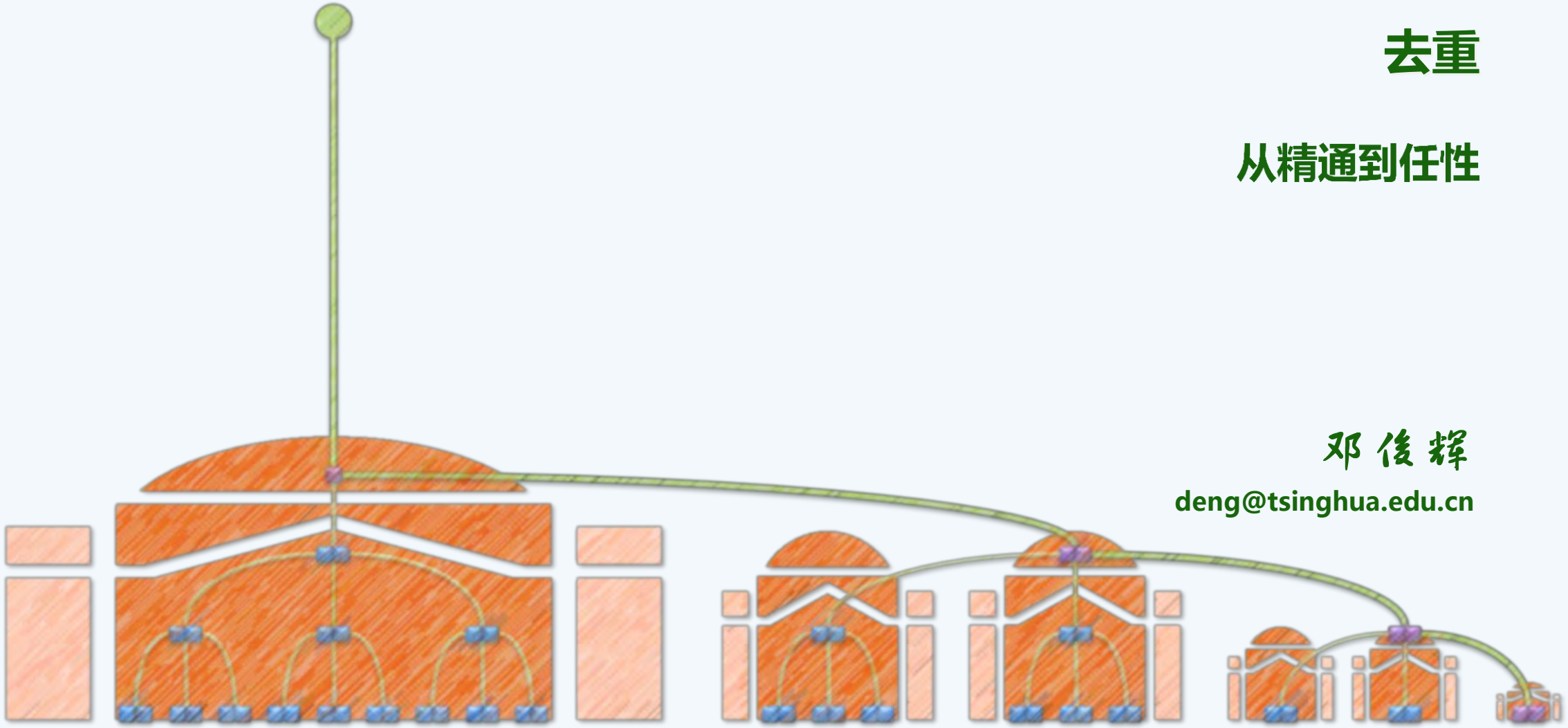
去重

从精通到任性



邓俊辉

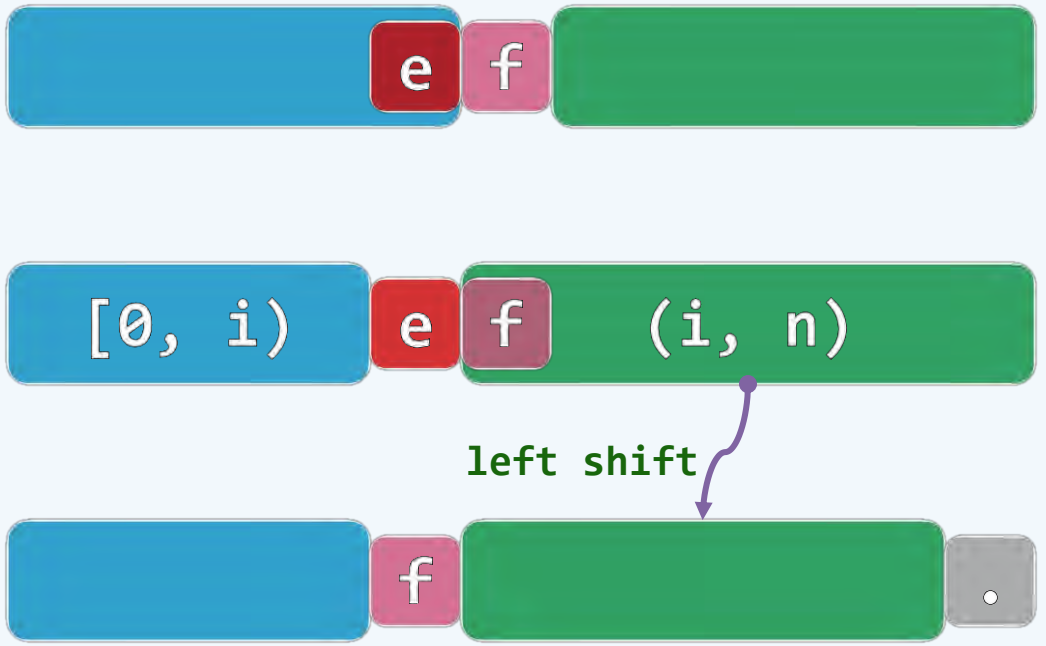
deng@tsinghua.edu.cn

deduplicate()

有些事，你一辈子总也忘不掉。凡是让你揪心的事，在你身上，都会发生两次。或两次以上。

无序向量

```
❖template <typename T> int Vector<T>::deduplicate() {
    int oldSize = _size;
    Rank i = 1;
    while ( i < _size )
       find( _elem[i], 0, i ) < 0 ?
          i++
        : remove(i);
    return oldSize - _size;
  }
```



uniquify()

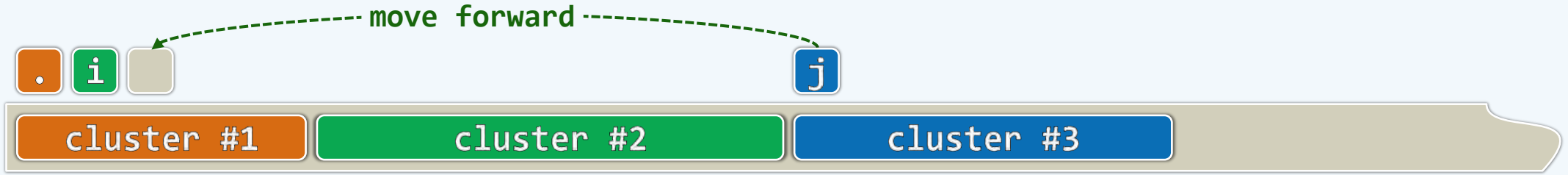
鳳兮鳳兮，故是一鳳

低效算法

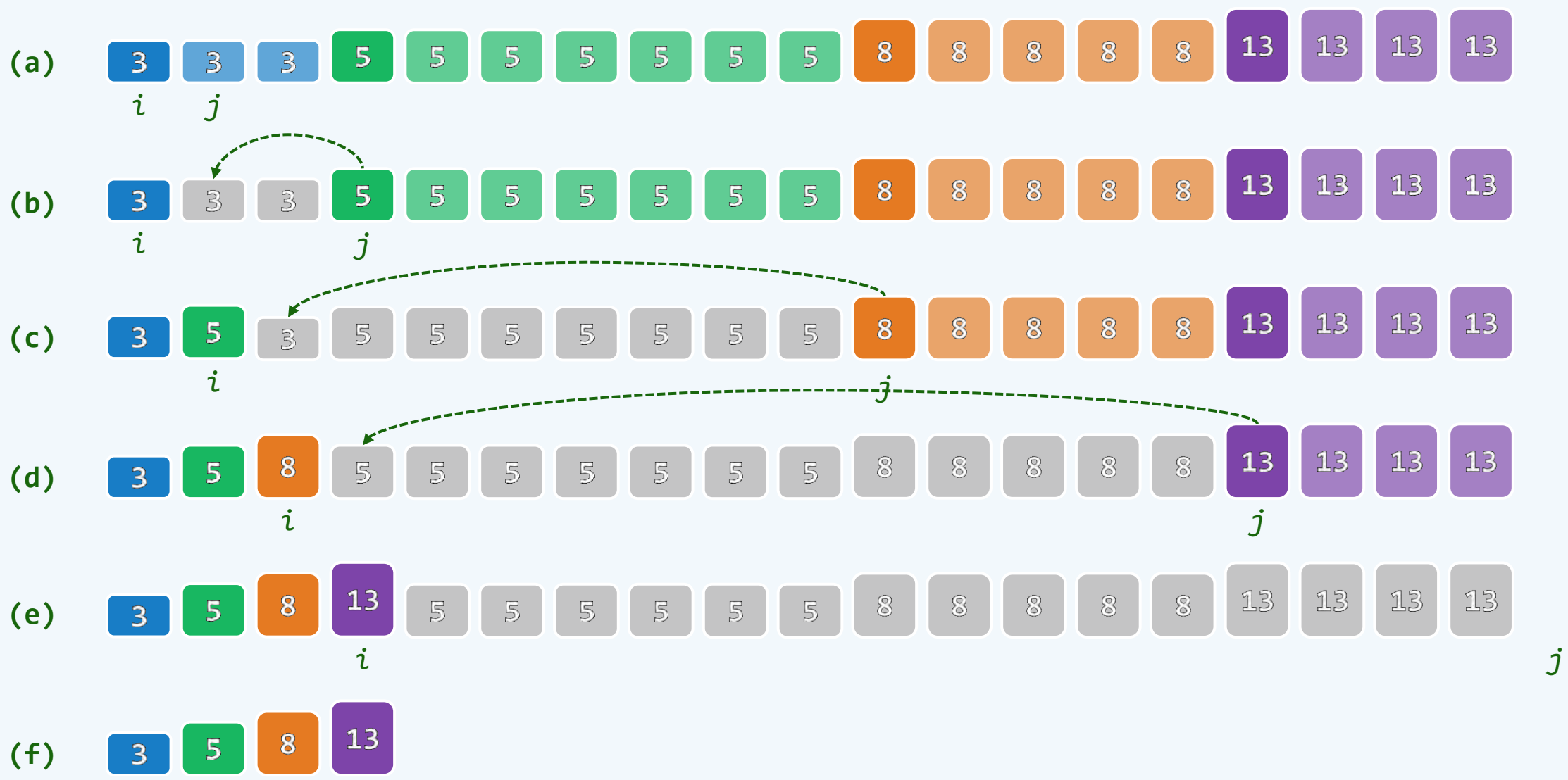
```
❖template <typename T> int Vector<T>::uniquify() {
    int oldSize = _size; int i = 1;
    while ( i < _size )
       _elem[i-1] == _elem[i] ? remove( i ) : i++;
    return oldSize - _size;
}
```

高效算法

```
❖ template <typename T> int Vector<T>::uniquify() {
     Rank i = 0, j = 0;
     while ( ++j < _size )
        if ( _elem[ i ] != _elem[ j ] ) _elem[ ++i ] = _elem[ j ];
     _size = ++i; shrink();
     return j - i;
  }
```



实例


(a)
3 3 3 5 5 5 5 5 5 5 8 8 8 8 8 13 13 13 13
i j
(b)
3 3 3 5 5 5 5 5 5 5 8 8 8 8 8 13 13 13 13
i j
(c)
3 5 3 5 5 5 5 5 5 5 8 8 8 8 8 13 13 13 13
i j
(d)
3 5 8 5 5 5 5 5 5 5 8 8 8 8 8 13 13 13 13
i j
(e)
3 5 8 13 5 5 5 5 5 5 8 8 8 8 8 13 13 13 13
i j
(f)
3 5 8 13

Bitmap

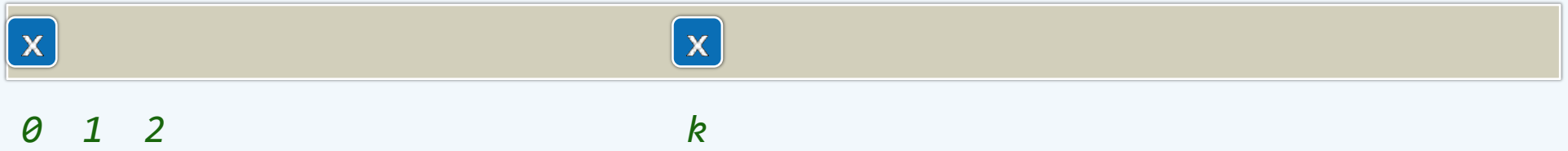
这样做能保存的信息量就小多了，不到原来的万分之一，但他们也只能接受这个结果。

整数集

$k \in S$? `bool test( int k );`

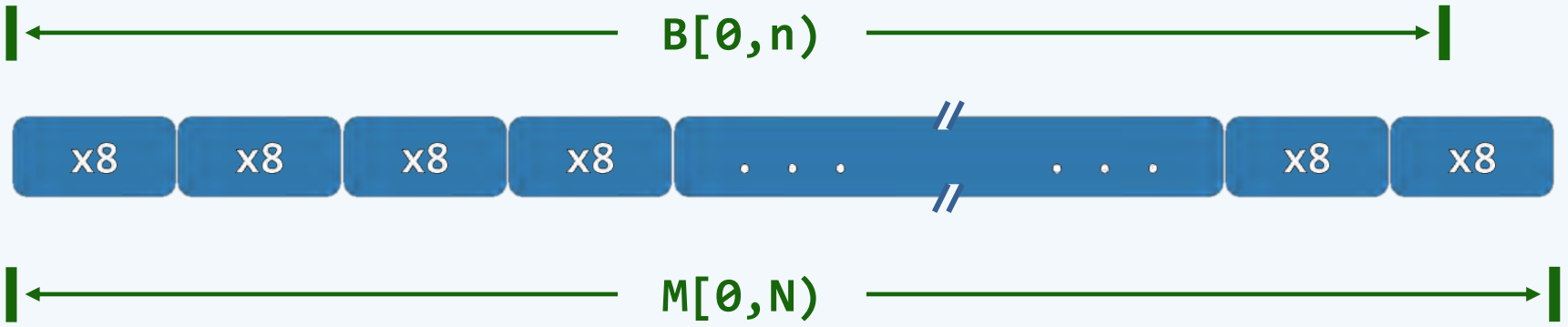
$S \cup \{k\}$ `void set( int k );`

$S \backslash \{k\}$ `void clear( int k );`



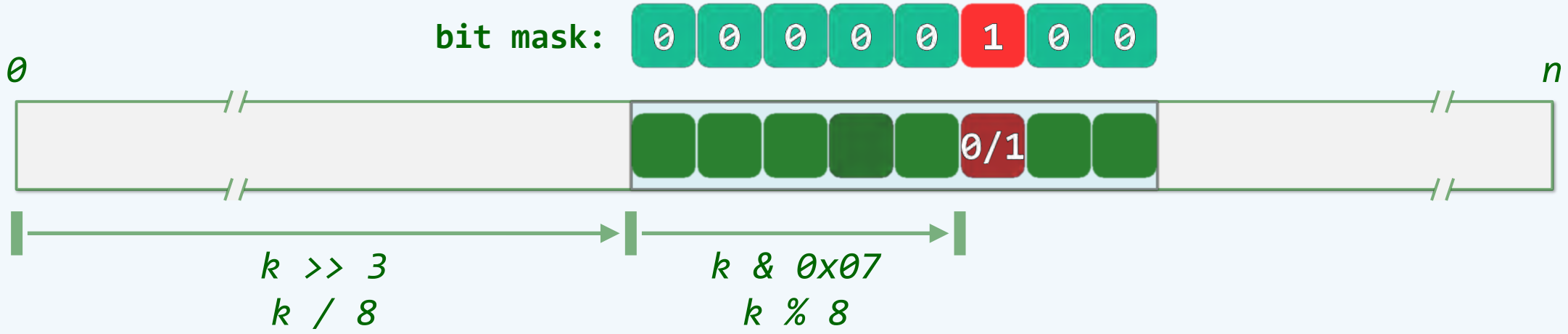
结构

```
❖class Bitmap {
  private:
     int N;
     char * M;
  public:
     Bitmap( int n = 8 ) { M = new char[ N = (n+7)/8 ]; memset( M, 0, N ); }
     ~Bitmap() { delete [] M; M = NULL; } //析构
     void set( int k ); void clear( int k ); bool test( int k ); //ADT
  };
```



实现

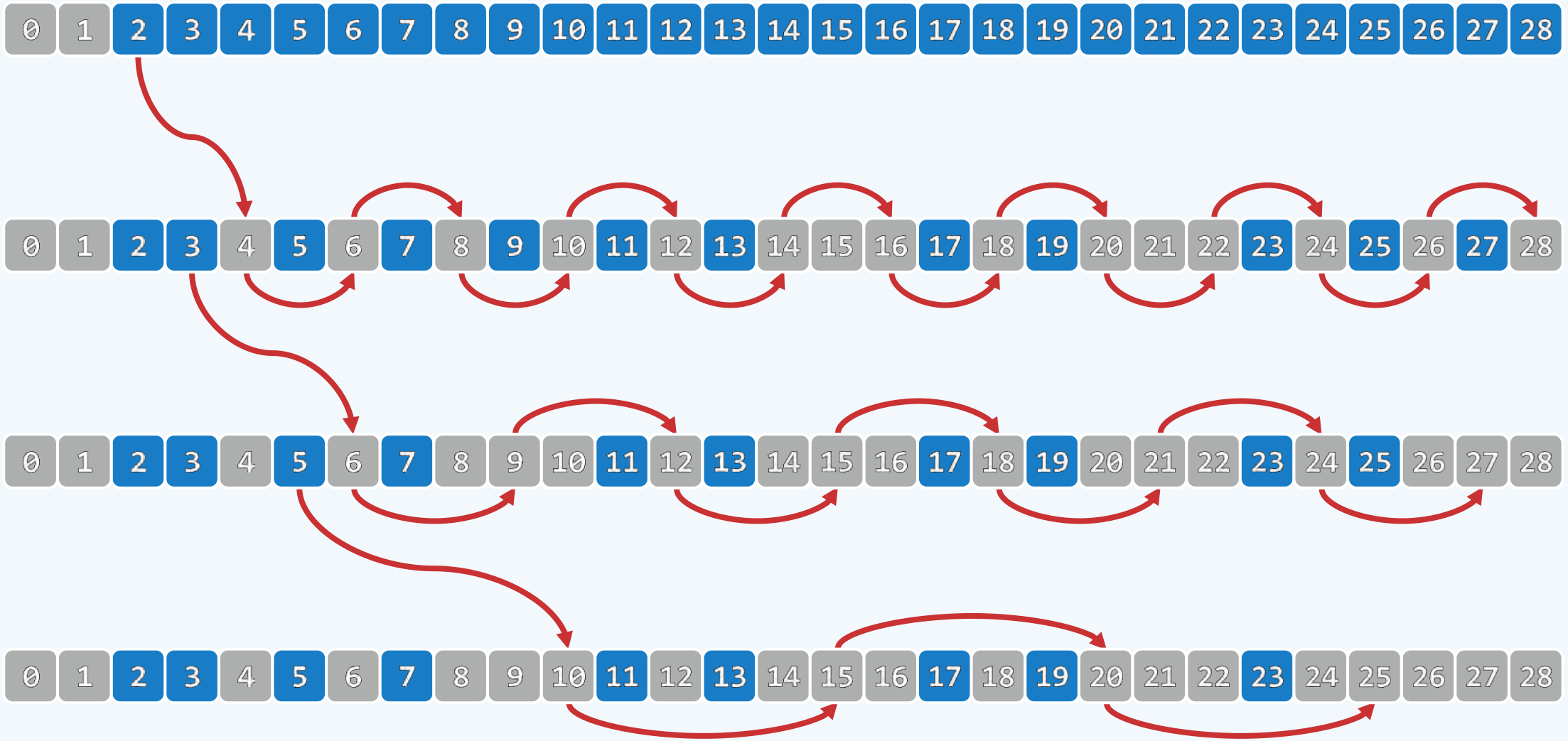
```
❖bool test( int k ) { return M[ k >> 3 ] &  ( 0x80 >> (k & 0x07) ); }
❖void set( int k )  { expand( k ); M[ k >> 3 ] |=  ( 0x80 >> (k & 0x07) ); }
❖void clear( int k ) { expand( k ); M[ k >> 3 ] &= ~( 0x80 >> (k & 0x07) ); }
```



Sieve Of Eratosthenes

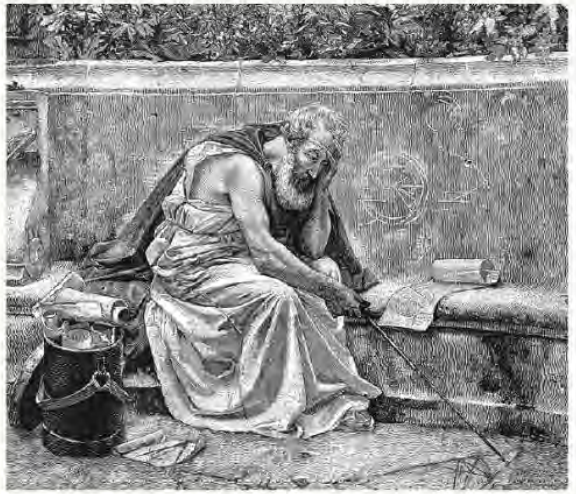
Those too big to pass through are our friends.

思路


0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28

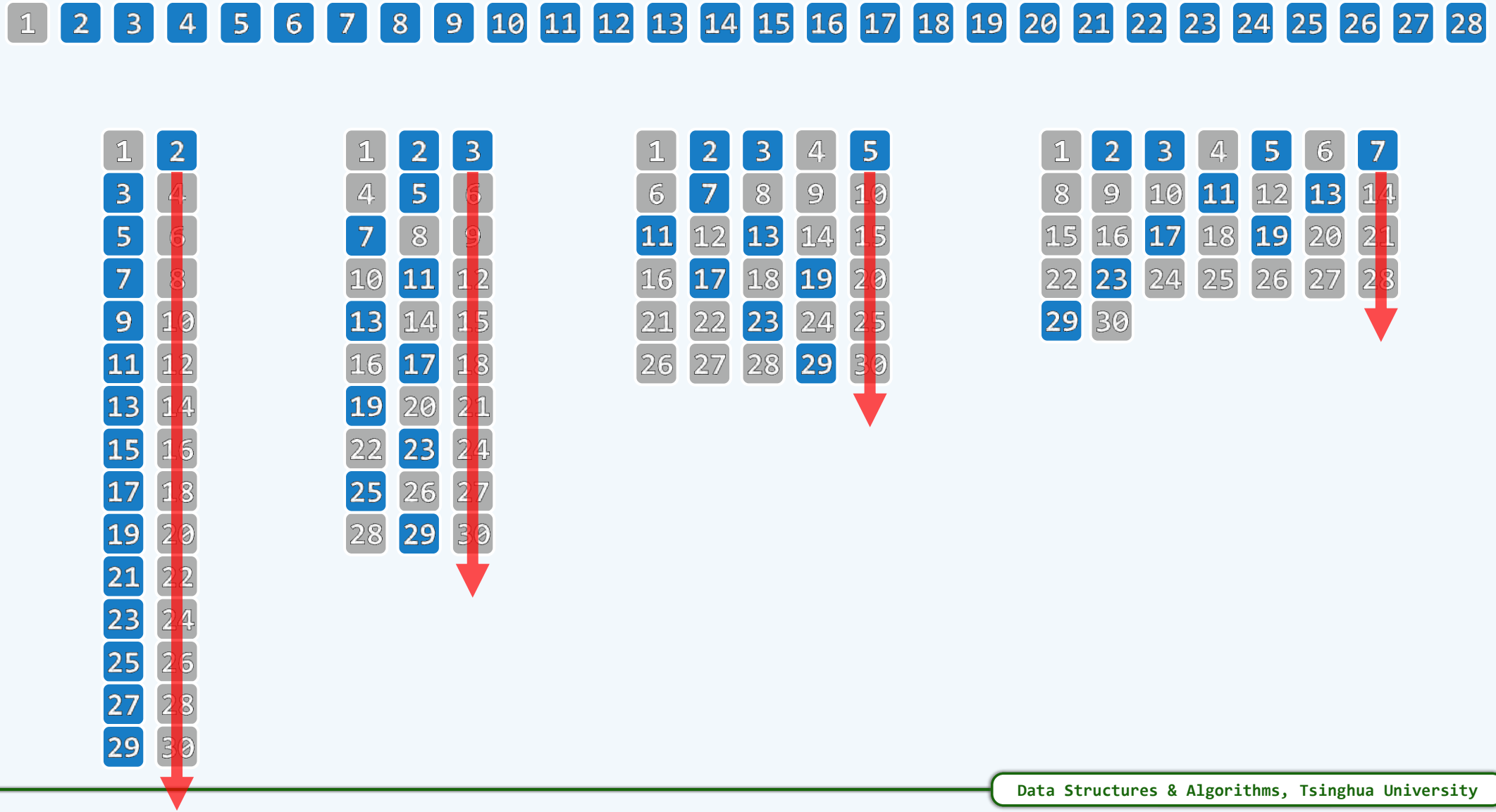
实现

```
❖void Eratosthenes( int n, char * file ) {
    Bitmap B( n );
    B.set( 0 ); B.set( 1 );
    for ( int i = 2; i < n; i++ )
       if ( ! B.test( i ) )
          for ( int j = 2*i; j < n; j += i )
             B.set( j );
    B.dump( file );
  }
```



Eratosthenes
(276 ~ 194 B.C.)

效果


1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30

Bucketsort

Don't put all your eggs in one basket.

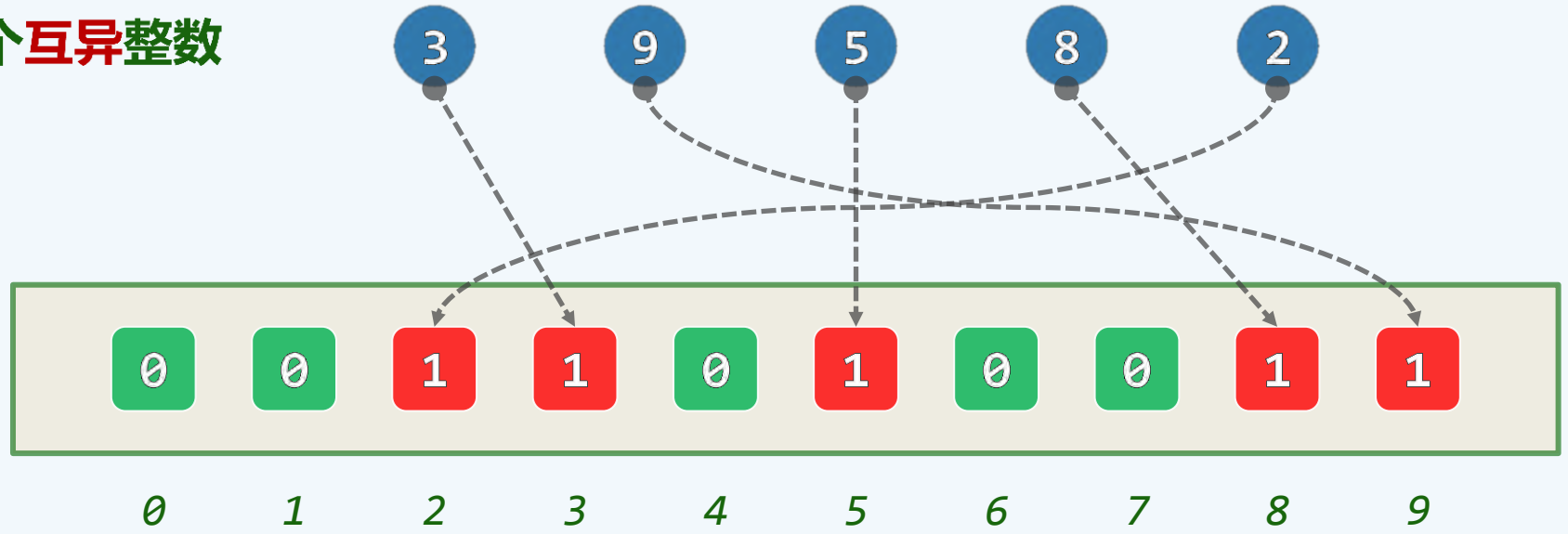
简单情况

❖ 对[0,m)内的n (<m) 个互异整数

借助散列表$\mathcal{H}$[]做排序

❖ 空间 = O(m)

时间 = O(n) !



❖ initialization: for i = 0 to m - 1, let $\mathcal{H}$[i] = 0 //O(m) --> O(1)

distribution: for each key in the input, let $\mathcal{H}$[key] = 1 //O(n)

enumeration: for i = 0 to m - 1, output i if $\mathcal{H}$[i] = 1 //O(m) --> O(n)

一般情况

- 若允许重复

  （可能m << n）

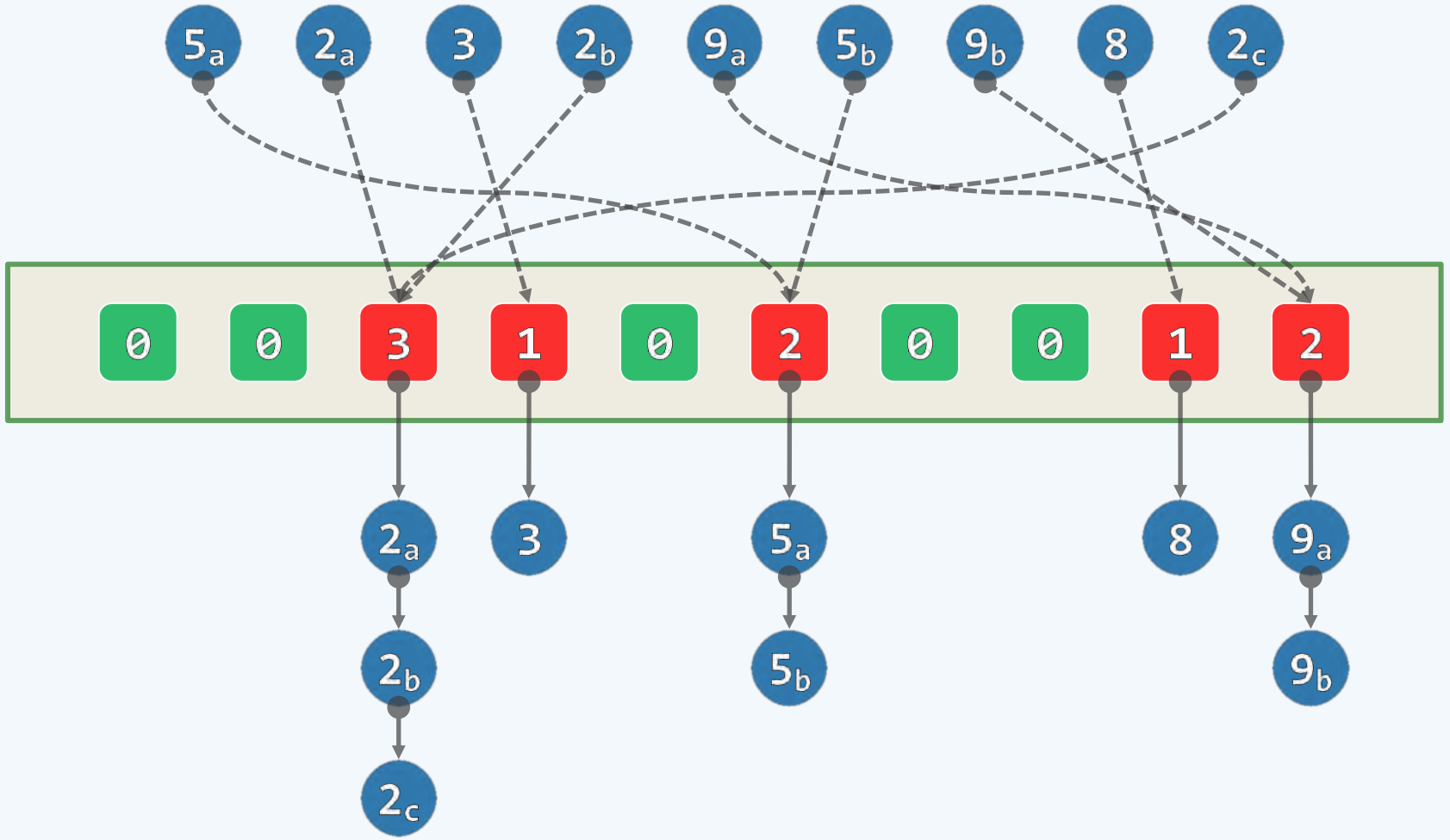
- 依然使用散列表

  - 每一组同义词

  - 各成一个链表

- 空间 = O(m + n)

  时间 = O(n) !



O(1)-Time Initialization

我从那无比圣洁的河水那里

走了回来，仿佛再生了一般

正如新的树用新的枝叶更新

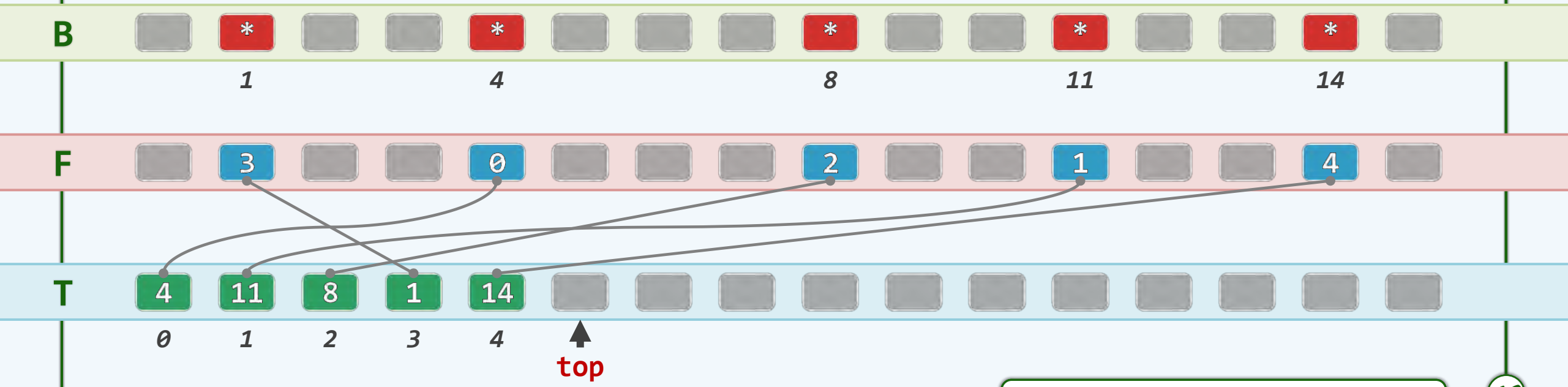
一身洁净，准备就绪，就飞往星辰

校验环

❖ //[J. Hopcroft, 1974] 将B[m]拆分为一对等长的Rank型向量...

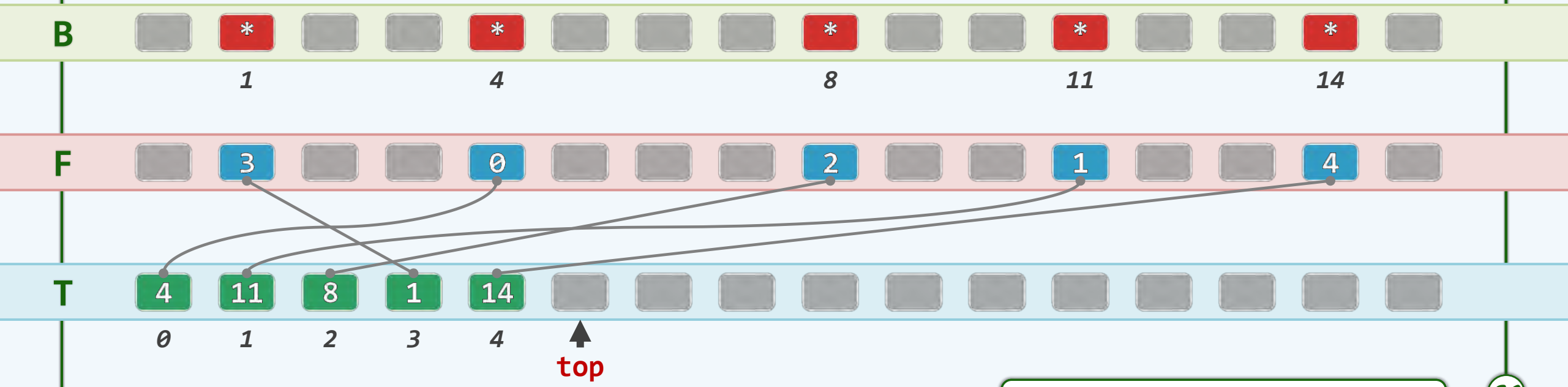
```
Rank F[ m ]; //From

Rank T[ m ]; Rank top = 0; //To及其栈顶指示
```



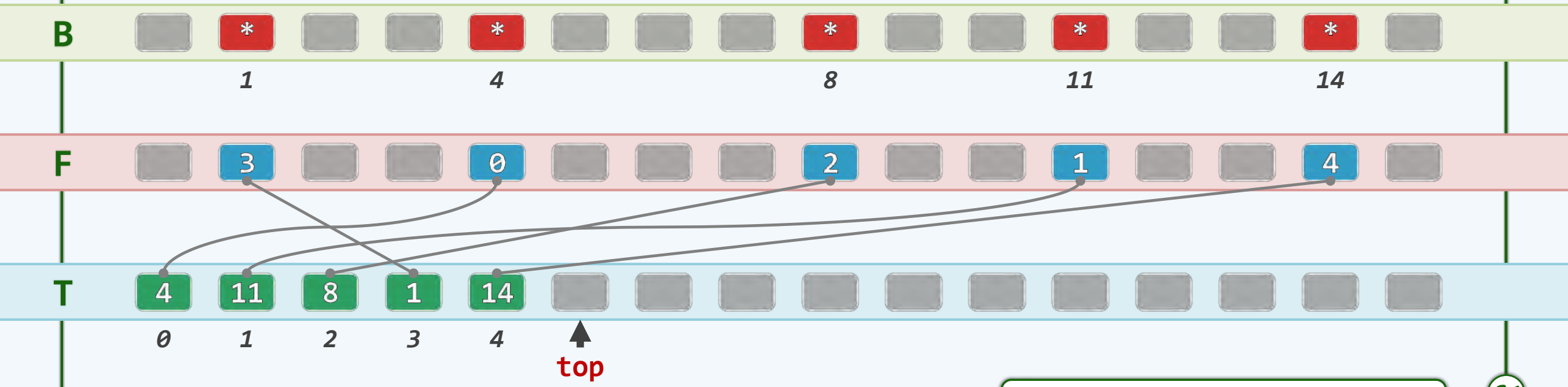
判断

```
❖bool Bitmap::test( Rank k ) {
    return ( 0 <= F[ k ] ) && ( F[ k ] < top )
        && ( k == T[ F[ k ] ] );
  }
```



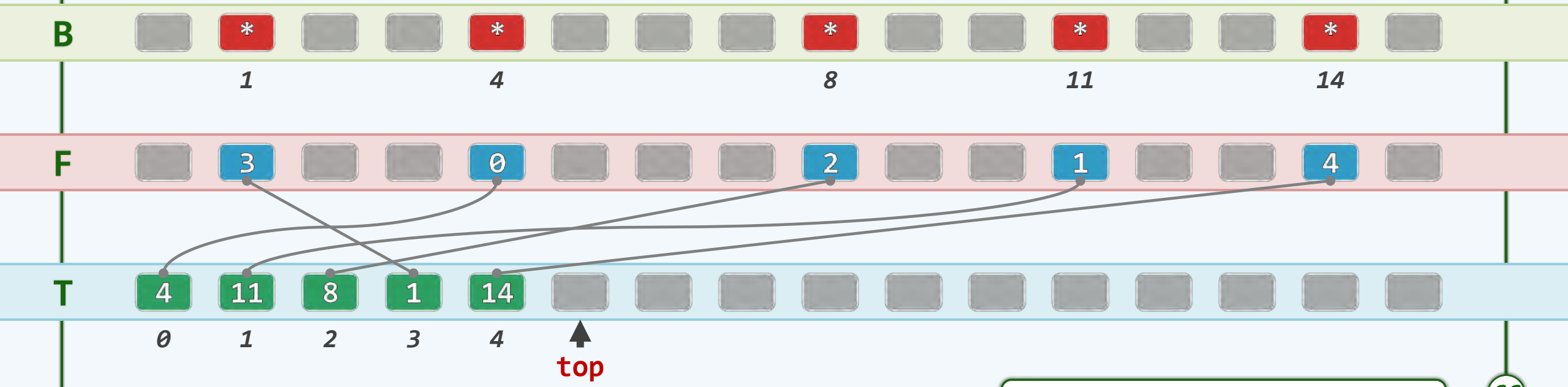
复位

❖ void Bitmap::reset() { top = 0; }



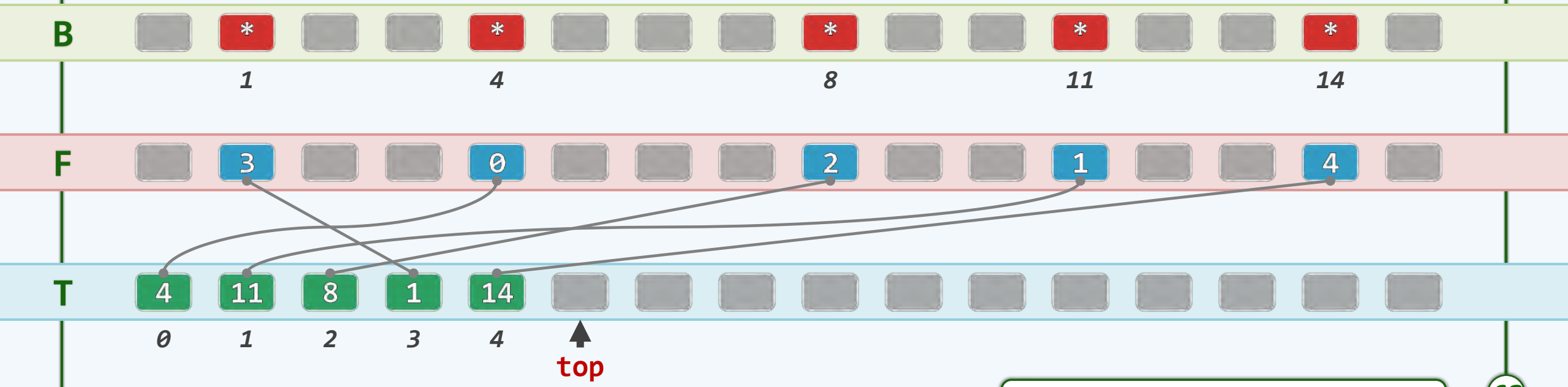
插入

```
❖void Bitmap::set( Rank k ) {
    if ( ! test ( k ) ) { T[ top ] = k; F[ k ] = top++; }
  }
```



删除

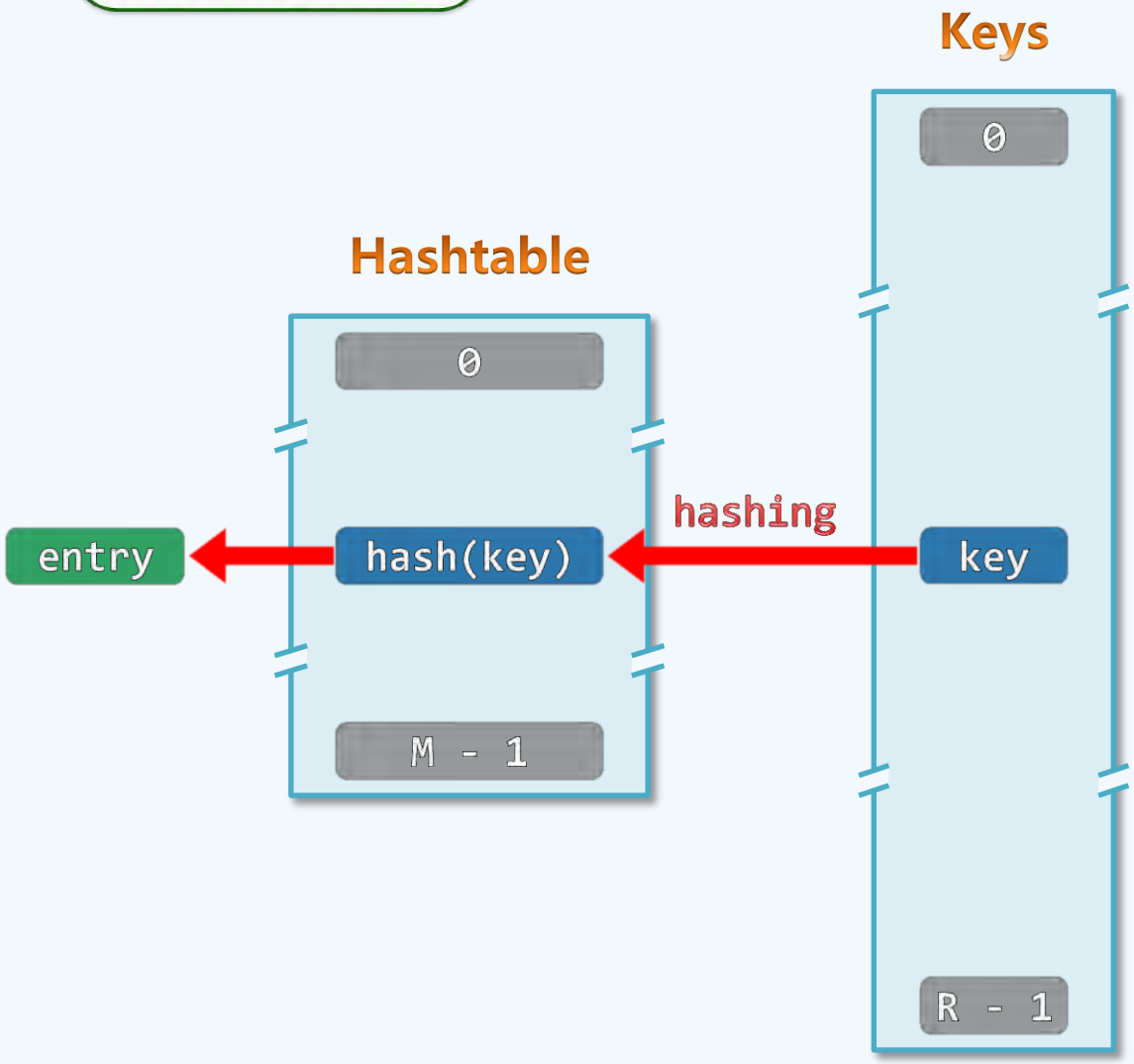
```
❖void Bitmap::clear( Rank k ) {
    if ( test ( k ) && ( --top ) )
     { F[ T[ top ] ] = F[ k ]; T[ F[ k ] ] = T[ top ]; }
  }
```

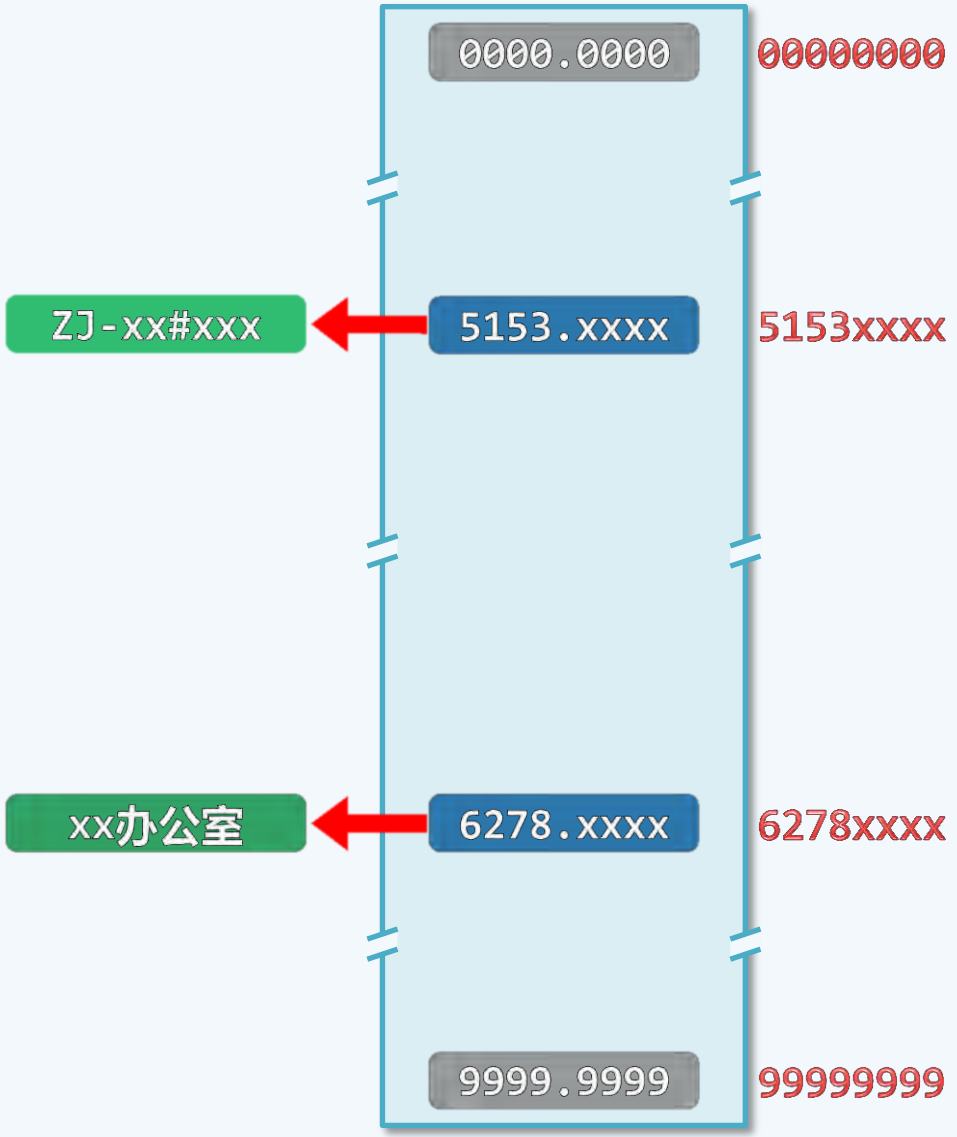


Hashing

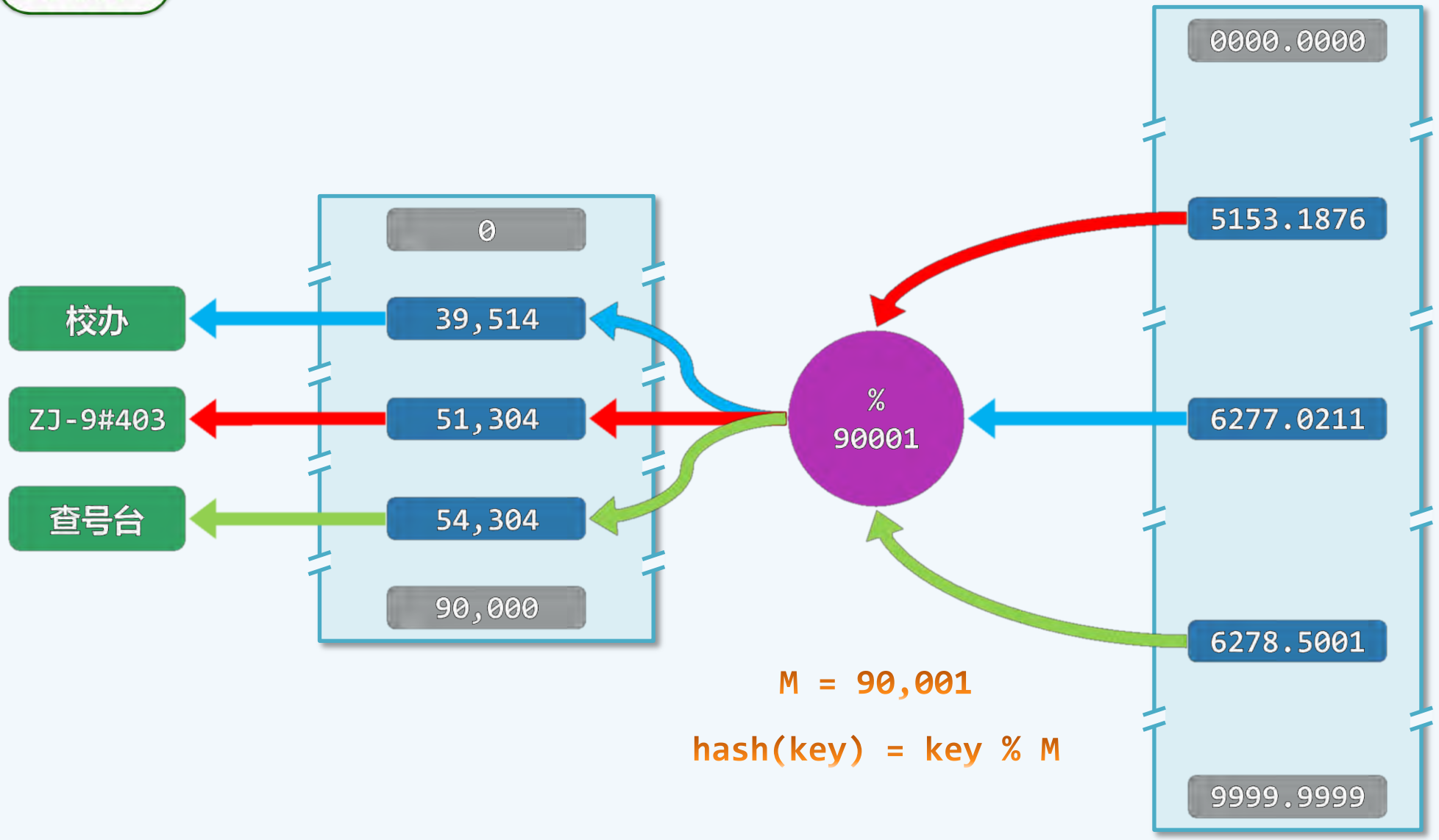
宝玉道：“已经完了，怎么又作揖？”袭人笑道：“这是他来给你拜寿。今儿也是他的生日，你也该给他拜寿。”宝玉听了，喜的忙作下揖去，说：“原来今儿也是姐姐的芳诞。”

原理与方法


Keys
0
Hashtable
0
hashing
entry
hash(key)
key
M - 1
R - 1

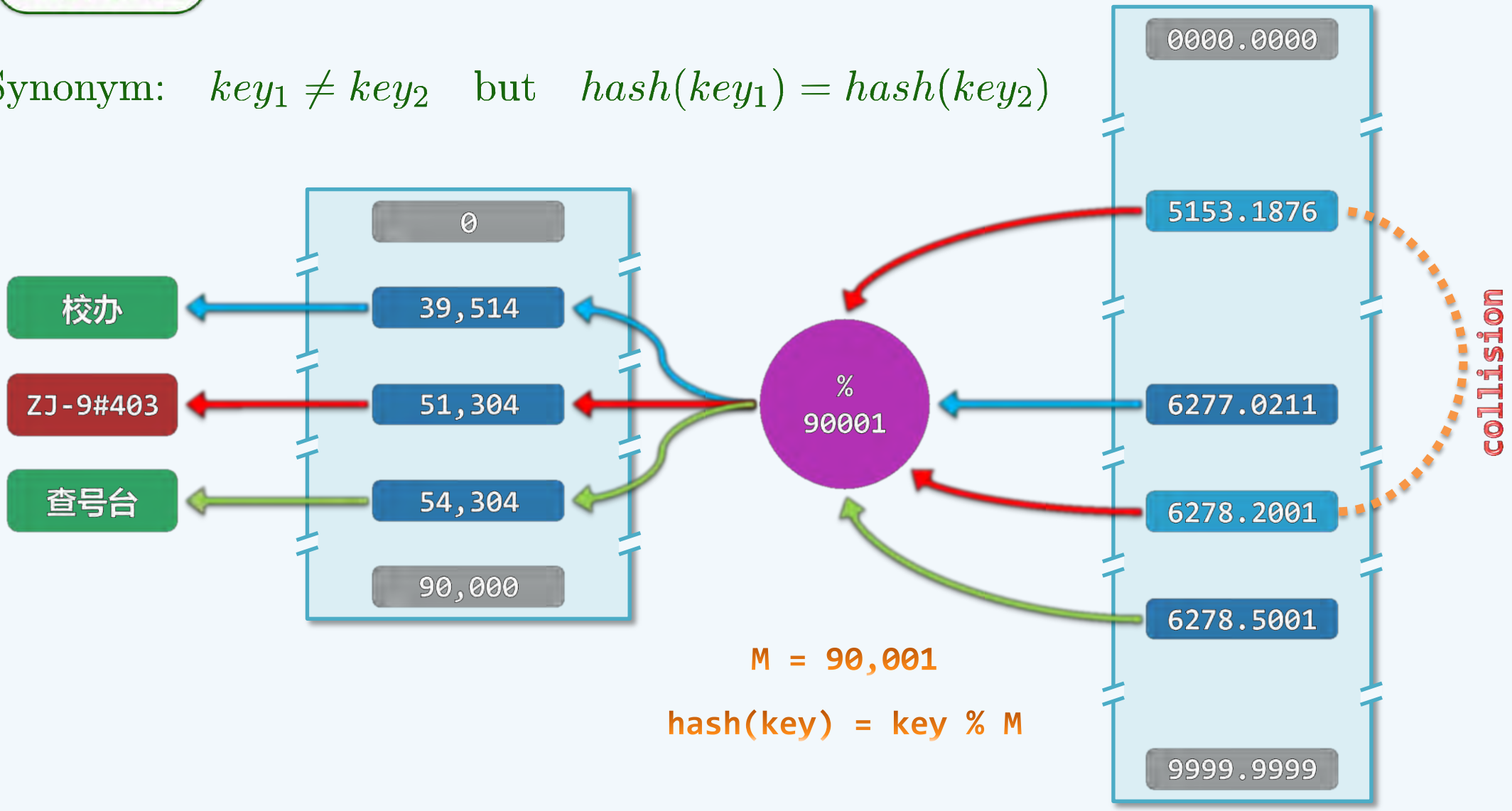

0000.0000
00000000
ZJ-xx#xxx
5153.xxxx
5153xxxx
xx办公室
6278.xxxx
6278xxxx
9999.9999
99999999

实例

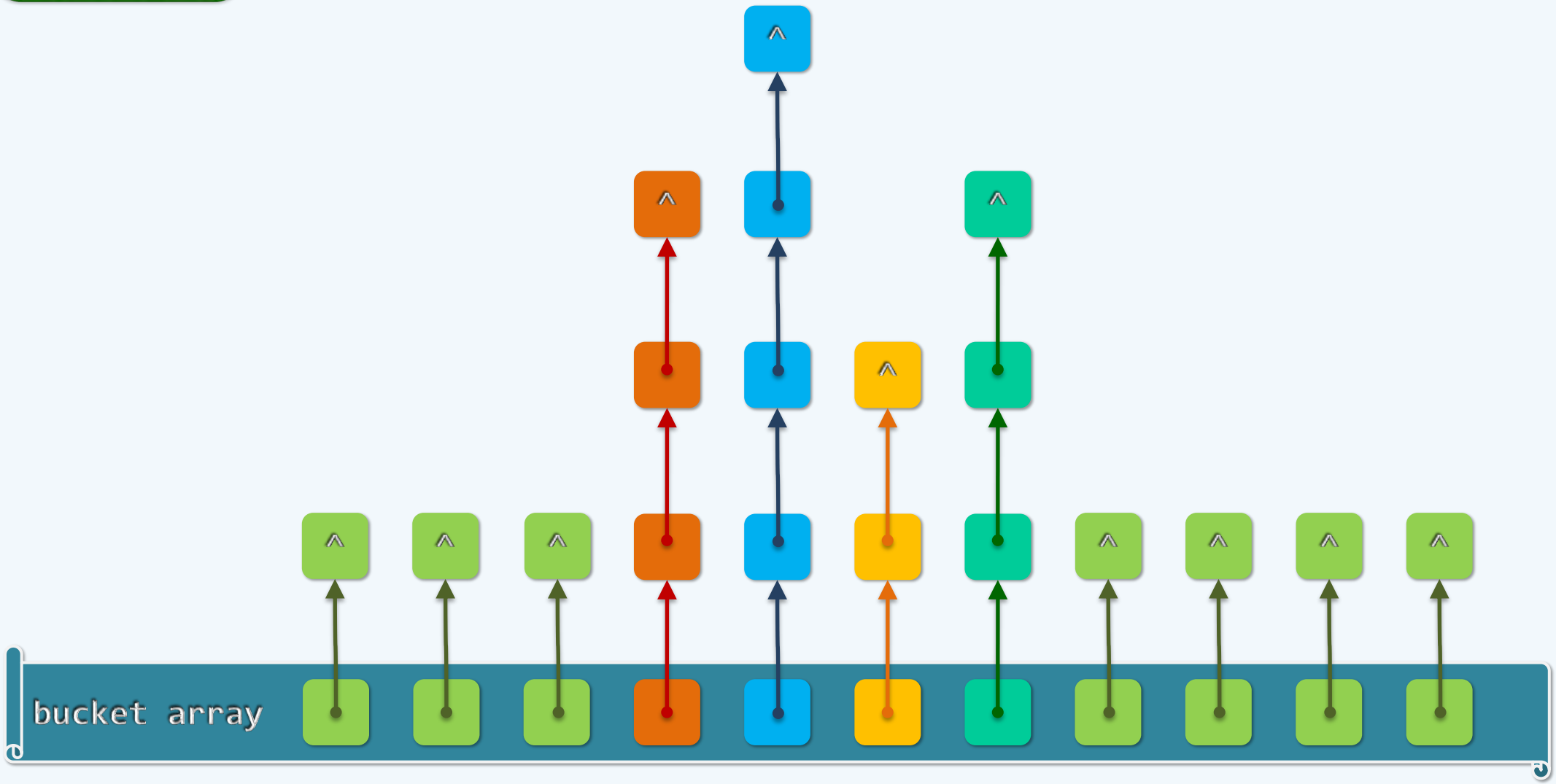

0000.0000
5153.1876
6277.0211
6278.5001
9999.9999
%
90001
0
39,514
51,304
54,304
90,000
校办
ZJ-9#403
查号台
M = 90,001
hash(key) = key % M

同义词

Synonym: $key_1 \neq key_2$ but $hash(key_1) = hash(key_2)$



排解冲突



Karp-Rabin Algorithm

我有些明白了：如果把要指明的恒星与周围恒星的相对位置信息发送出去，接收者把它与星图进行对照，就确定了这颗恒星的位置。

散列压缩

❖ 基本构思：通过对比经压缩之后的**指纹**，确定匹配位置

❖ 关键技巧：通过**散列**，将指纹压缩至存储器支持的范围

比如，采用模余函数：hash(key) = key % 97

❖ P = 8 2 8 1 8 //hash(82818) = 77

T = 2 7 1 8 2 8 1 8 2 8 4 5 9 0 4 5 2 3 5 3 6

2 7 1 8 2 //22

7 1 8 2 8 //48

1 8 2 8 1 //45

8 2 8 1 8 //77

散列冲突

❖ 注意：hash()值相等，并非匹配的充分条件... //好在必要

因此，通过hash()筛选之后，还须经过严格的比对，方可最终确定是否匹配...

❖ P = 1 8 2 8 4 //hash(18284) = 48

T = 2 7 1 8 2 8 1 8 2 8 4 5 9 0 4 5 2 3 5 3 6

2 7 1 8 2 //22

7 1 8 2 8 //48

. . .

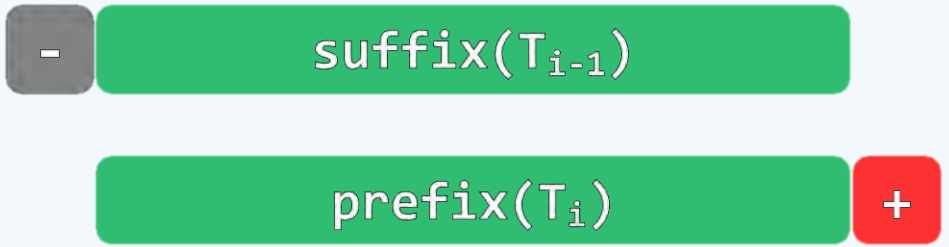
1 8 2 8 4 //48

❖ 既然是散列压缩，指纹冲突就在所难免——好在，适当选取散列函数，极大降低冲突的概率

快速指纹计算

- hash()的计算，似乎每次均需$O(|P|)$时间

  有可能加速吗？



- 回忆一下，进制转换算法...

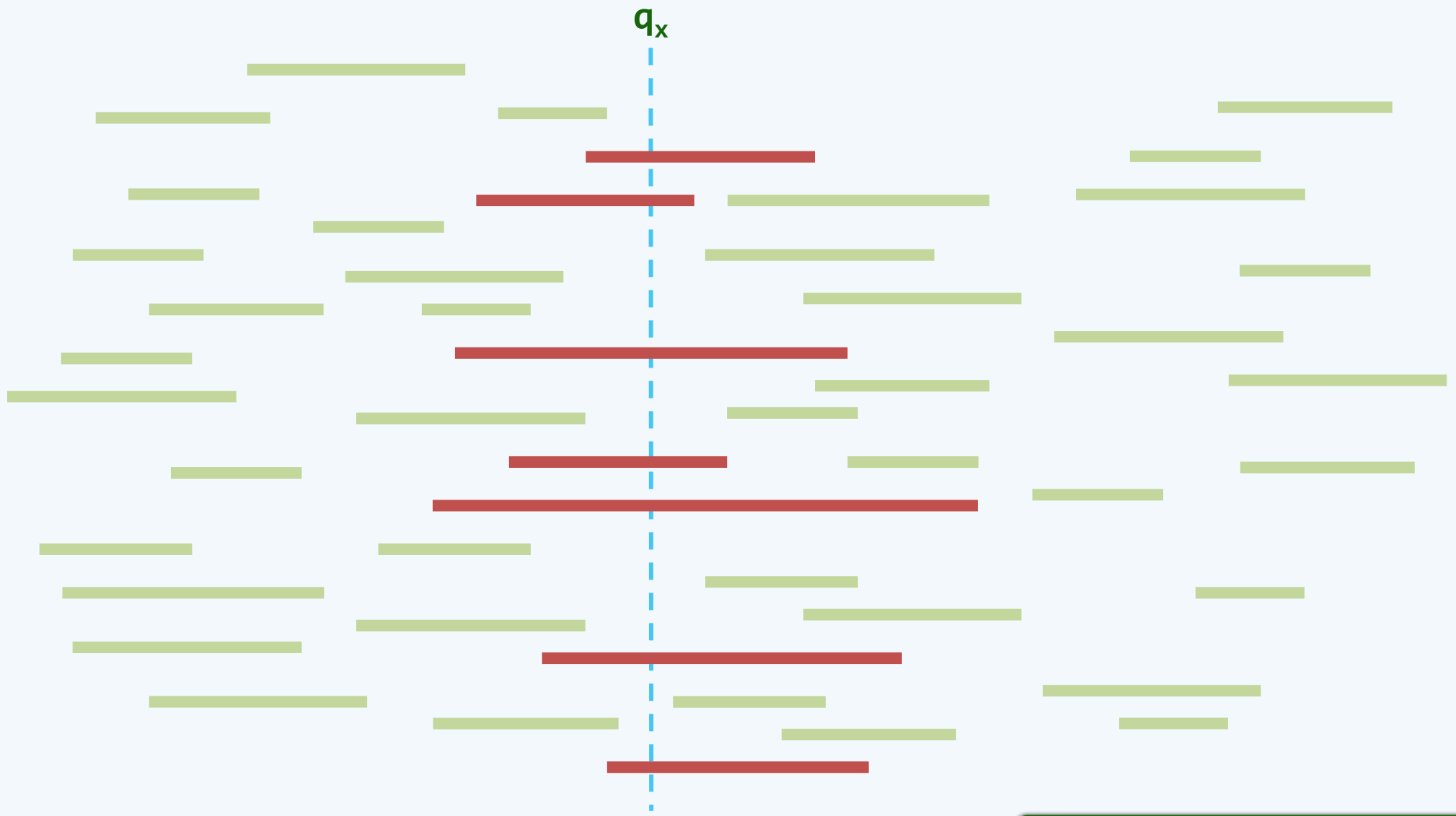
- 观察
  - 相邻的两次**散列**之间，存在着某种相关性
  - 相邻的两个**指纹**之间，也有着某种相关性

- 利用上述性质，即可在$O(1)$时间内，由上一指纹得到下一指纹...

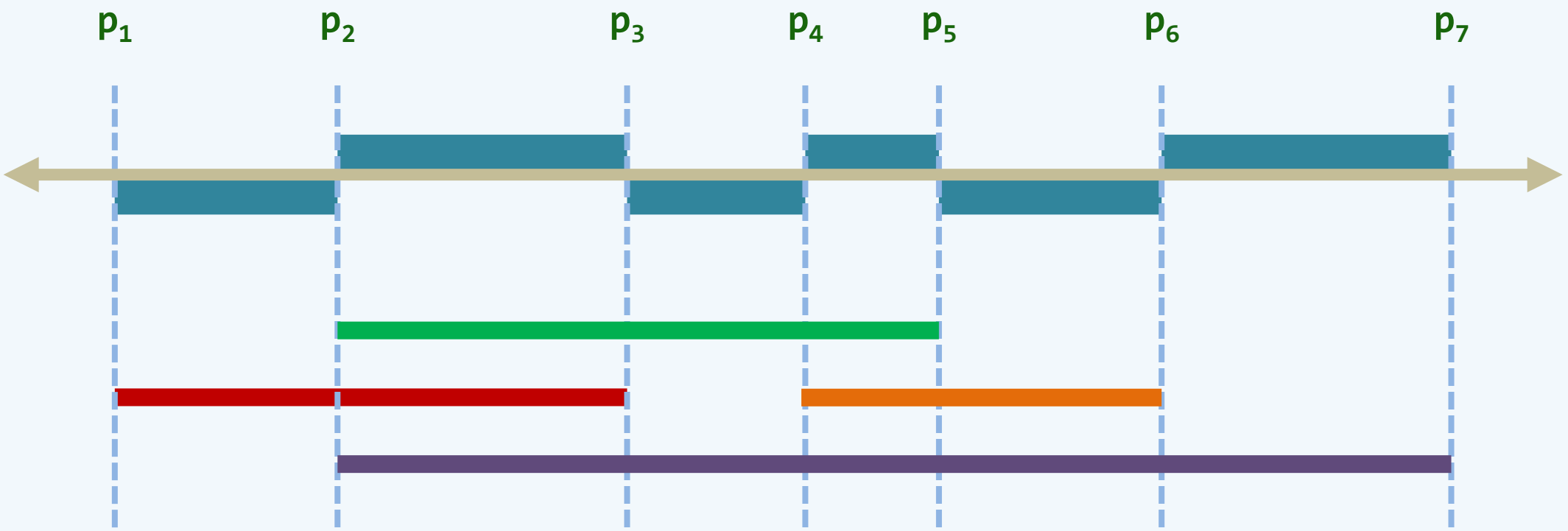
Stabbing Query

Your instinct, rather than precision stabbing, is more about just random bludgeoning.

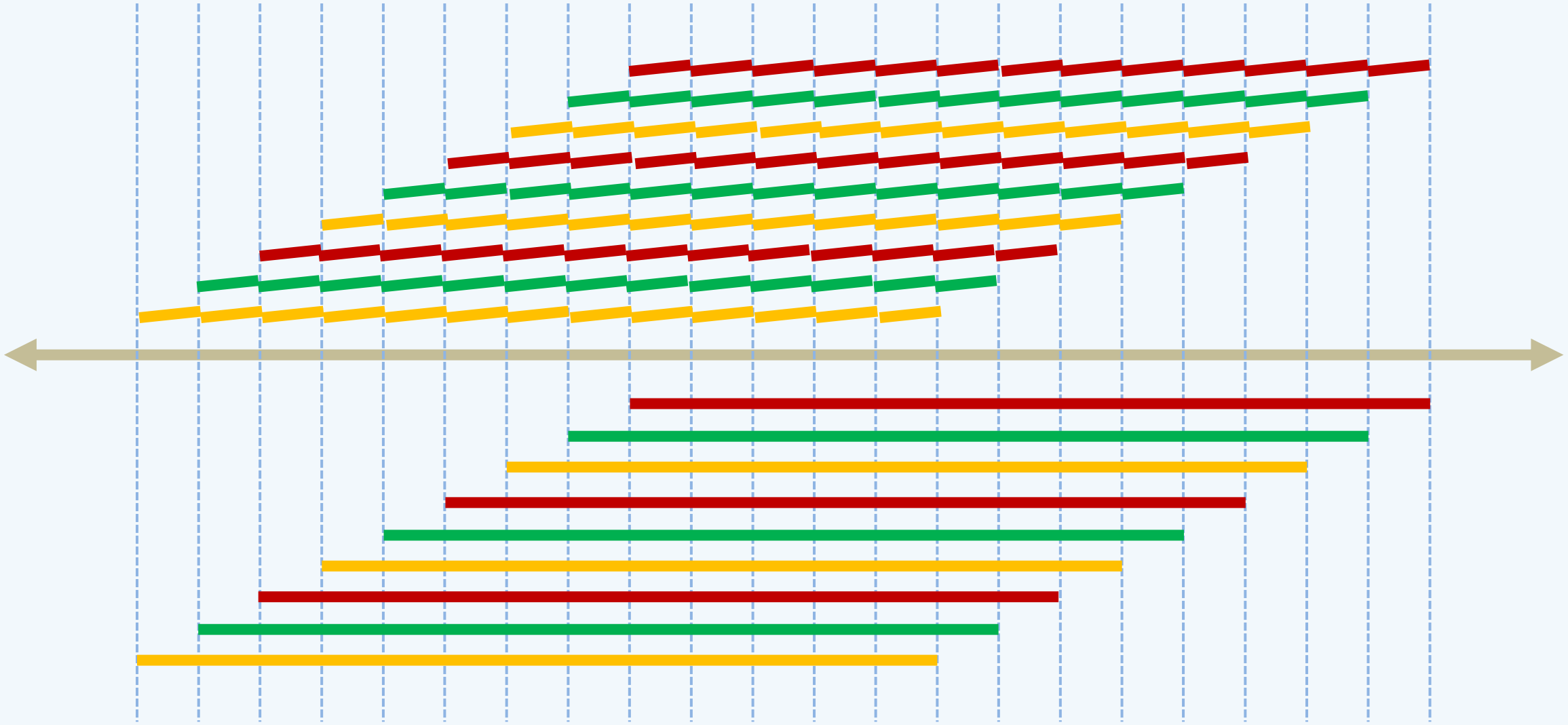
穿刺查询



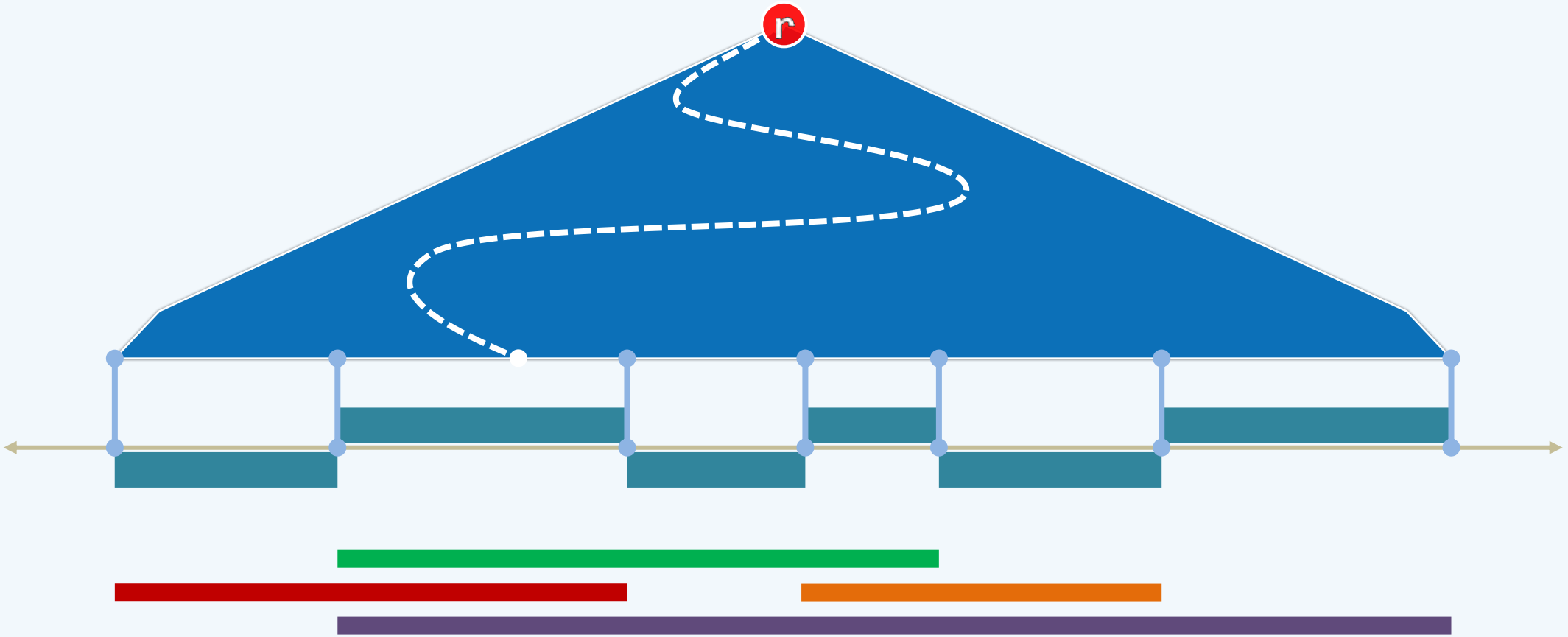
离散化



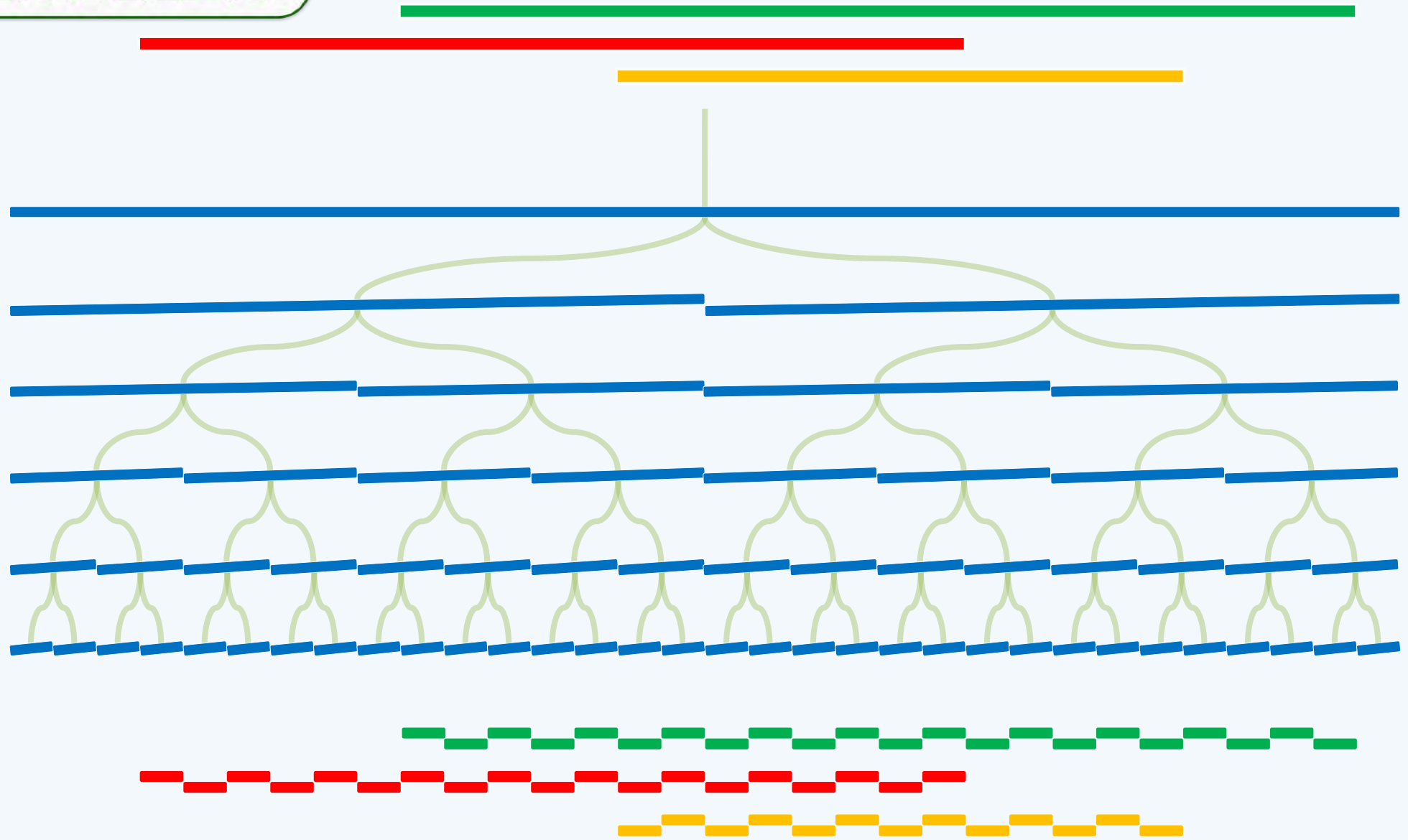
重复



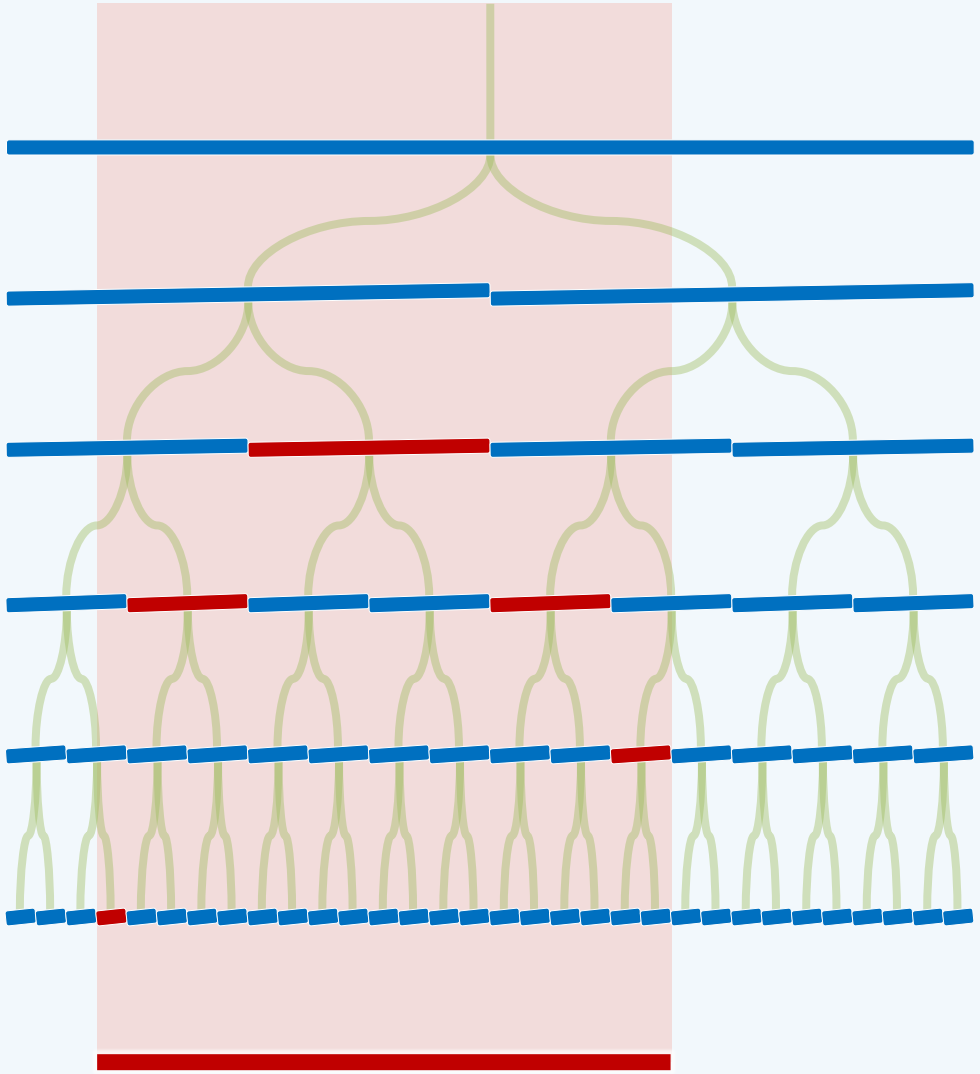
Segment Tree

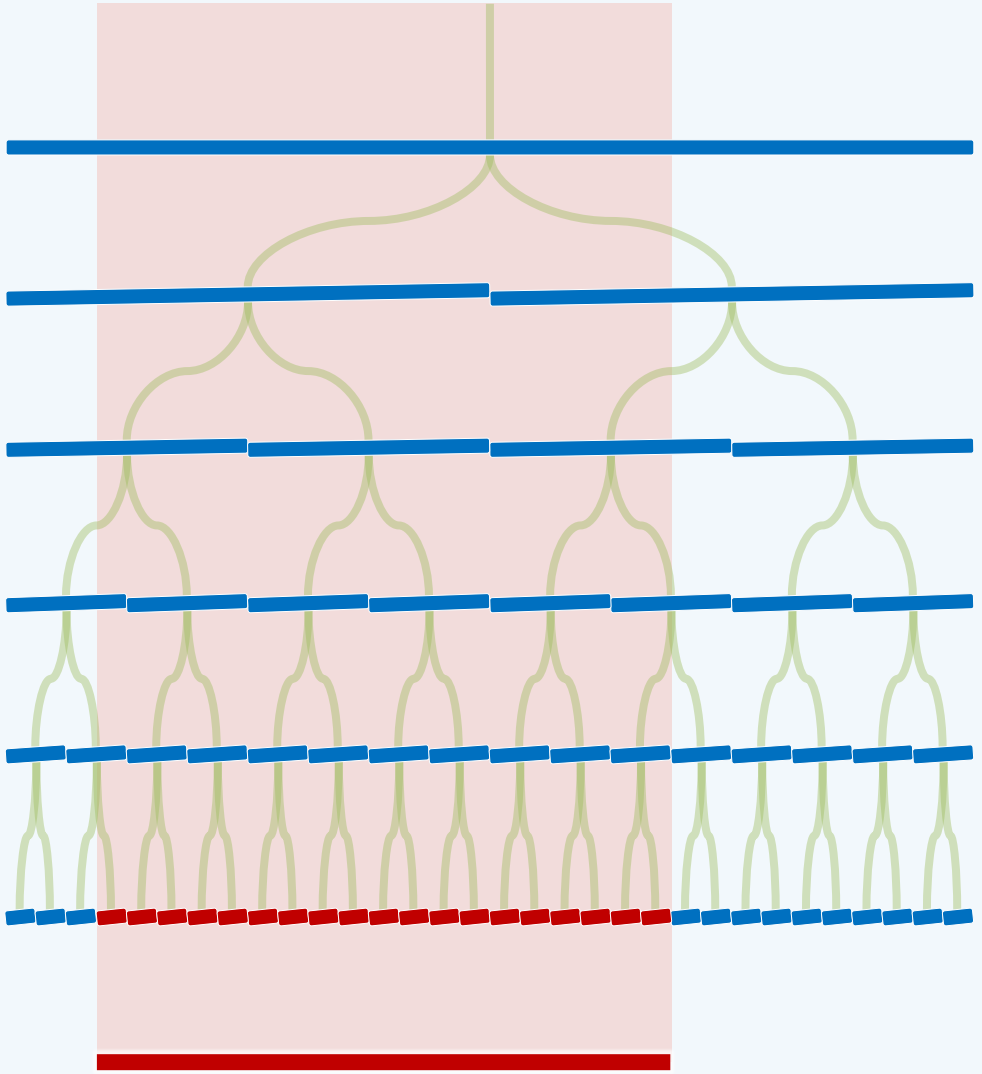


重复，还是重复



合并





插入 + 查询