9.词典

（xb2）位图：典型应用

邓俊辉

deng@tsinghua.edu.cn

小集合 + 大数据

- 老问题：int A[n]的元素均取自[0, m)，如何剔除其中的重复者？
- 仿照Vector::deduplicate()改进版：先排序，再扫描 —— O(nlogn + n) —— 毫无压力
- 新特点：数据量虽大，但重复度极高 —— 想想我们电脑里的MP3，不难理解
- 比如，2^24 = m << n = 10^10

  亦即，10,000,000,000个24位无符号整数
- 如果采用内部排序算法，你至少需要 4 * n = 40GB 内存

  ——否则，频繁的I/O将导致整体效率的低下
- 那么，m << n的条件，又应如何加以利用？

小集合 + 大数据

❖ Bitmap B(m); //O(m)

```
for (int i = 0; i < n; i++) B.set( A[i] ); //O(n)
for (int k = 0; k < m; j++) if ( B.test( k ) ) /* ... */; //O(m)
```

❖ 总体运行时间 = O(n + m) = O(n)

❖ 空间 = O(m)　　就上例而言，降至：m/8 = 2^21 = 2MB << 40GB

即便m = 2^32，也不过：2^29 = 0.5GB

❖ 拓展： 搜索引擎的使用规律亦是如此，词表规模不大，但重复度极高

——如何从中剔除重复的索引词？

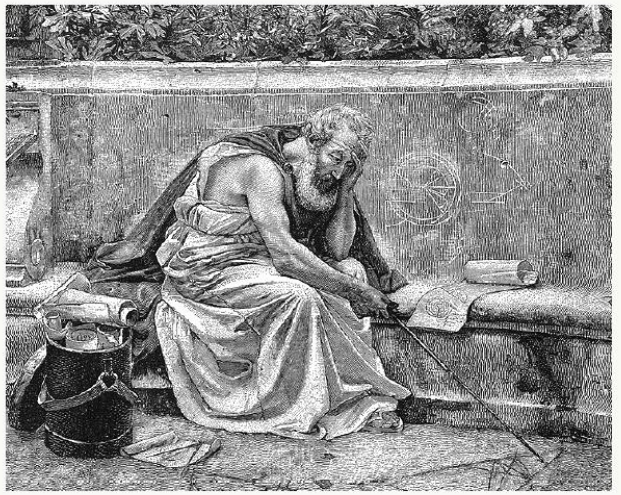
❖ 关键在于，如何将查询词表转换为某一集合——留作习题

筛法

❖ 如何计算出[0, n)之间的所有素数？

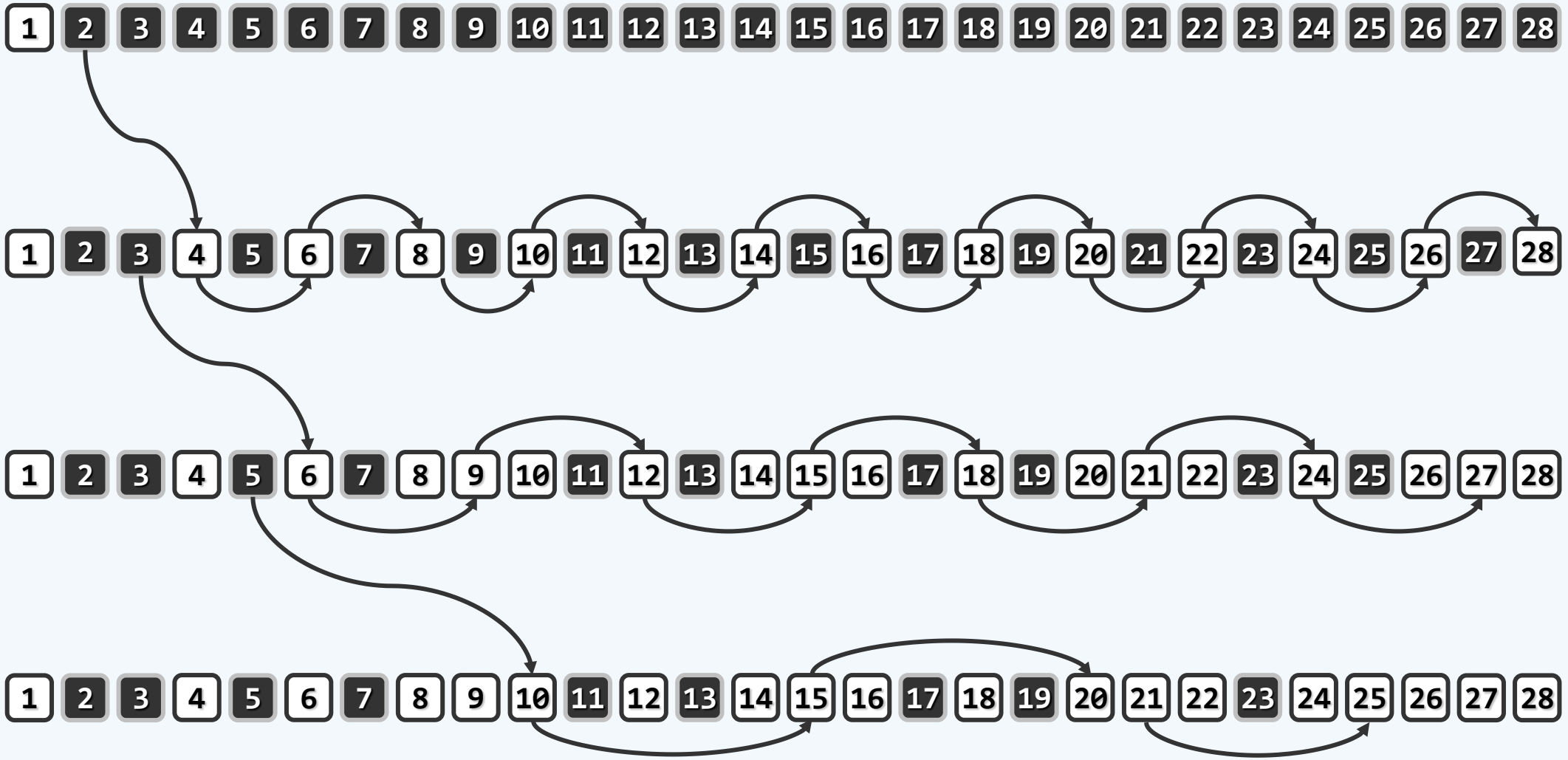
❖ 与其说是计算，不如说是筛选

❖
```
void Eratosthenes( int n, char * file ) {
    Bitmap B( n ); //创建位图
    B.set( 0 ); B.set( 1 ); //0和1都不是素数
    for ( int i = 2; i < n; i++ ) //反复地，从下一
        if ( ! B.test( i ) ) //可认定的素数i起
            for ( int j = 2 * i; j < n; j += i ) //以i为间隔
                B.set( j ); //将下一个数标记为合数
    B.dump( file ); //将筛选出的素数转储至文件，以便日后使用
}
```

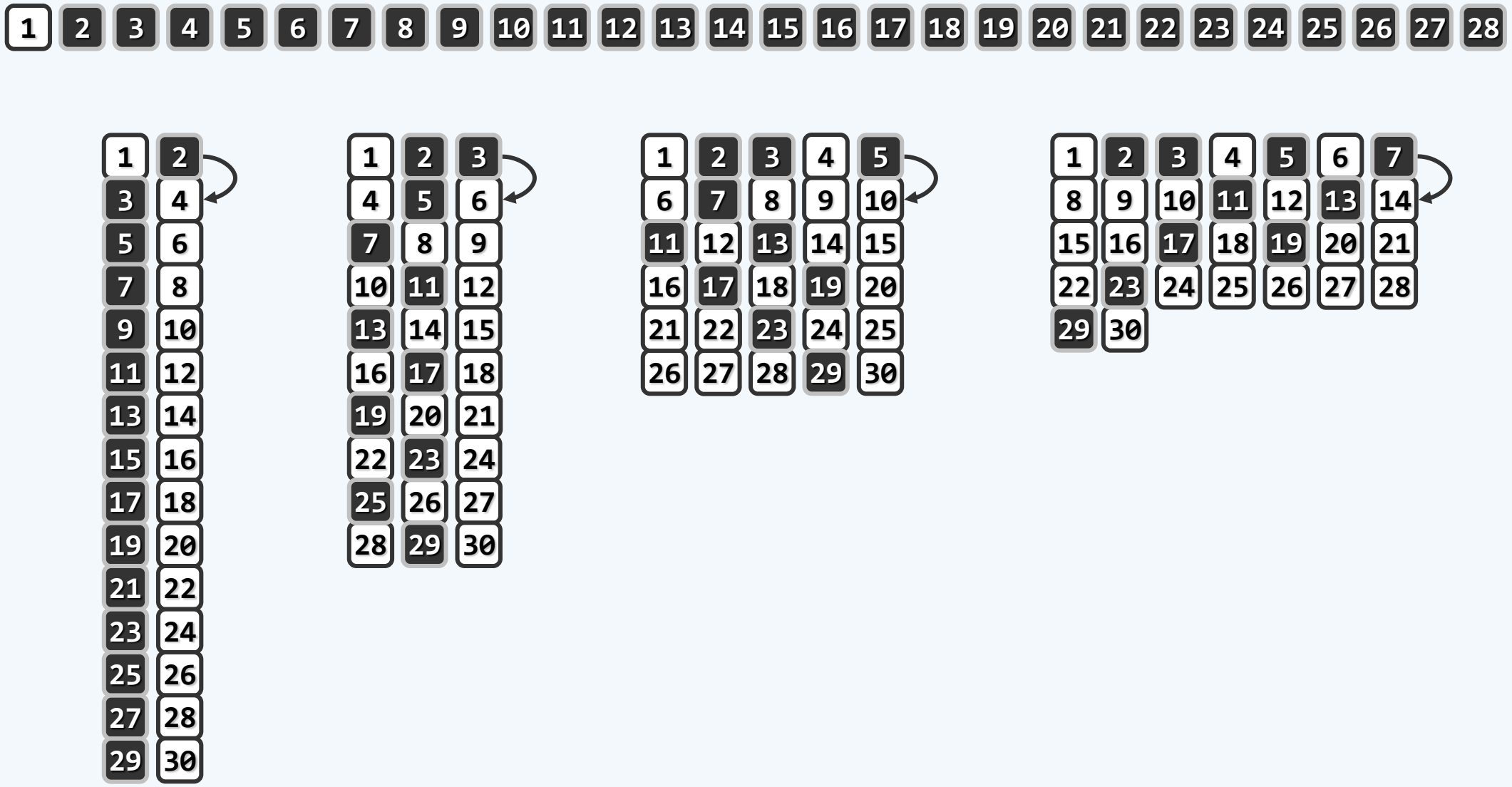


Eratosthenes
(276 ~ 194 B.C.)

筛法：实例


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
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

筛法：实例


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

筛法：效率

❖ 不计内循环，外循环自身每次仅一次加法、两次判断，累计$O(n)$

❖ 内循环每趟迭代$O(n/i)$步，由素数定理至多$n/\ln(n)$趟，累计耗时不过

$$n/2 + n/3 + n/5 + n/7 + n/11 + \ldots$$

$$< n/2 + n/3 + n/4 + n/5 + n/6 + \ldots + n/(n/\ln(n))$$

$$= O(n * [\ln(n/\ln(n)) - 1])$$

$$= O(n\ln(n) - n\ln(\ln(n)))$$

$$= O(n\log(n))$$

筛法：改进

- 循环起点 `i + i` ，可进一步改作 `i * i`

  ——为什么？

- 如此，每次内循环的长度，由

  $O(\ n/i\ )$

  降至

  $O(\ \max(\ 1,\ n/i\ -\ i\ )\ )$

- 效率有所提高！

  ——从渐进的角度看，是否实质改进？

- 基于以上，如何实现`primeNLT(int low)`？