19.数据结构-位图

1.概念

位图
本质是一个数组，被查找的数字当做下标，值为boolean类型，true表示被查找的数存在，false表示不存在，大多数语言的boolean类型都是1个字节，这对于位图来说很浪费，所以需要就间接的利用某一数据类型做位运算来表示位图的变化情况
例如：判断num这个数是否存在，array[num]取出来，为 true就存在，false表示不存在

```
public class BitMap { // Java中char类型占16bit，也即是2个字节
  private char[] bytes; 位图代码
  private int nbits;    不直接使用boolean类型
                        而是简介的使用char类型做位运算来表示一个数
  public BitMap(int nbits) {
    this.nbits = nbits;
    this.bytes = new char[nbits/16+1];
  }

  public void set(int k) {
    if (k > nbits) return;
    int byteIndex = k / 16;
    int bitIndex = k % 16;
    bytes[byteIndex] |= (1 << bitIndex);
  }

  public boolean get(int k) {
    if (k > nbits) return false;
    int byteIndex = k / 16;
    int bitIndex = k % 16;
    return (bytes[byteIndex] & (1 << bitIndex)) != 0;
  }
}
```

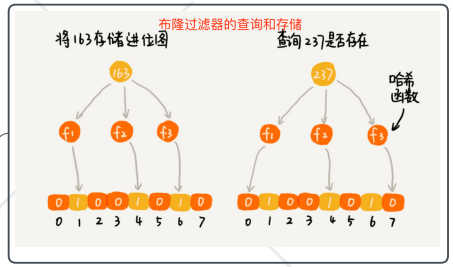
2.布隆过滤器

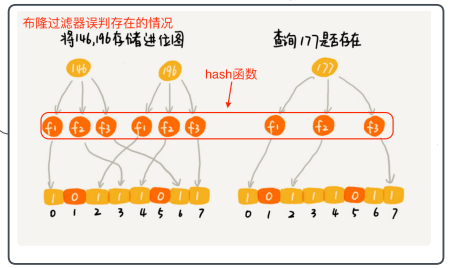
概念：
可以理解位图的改进版本，由于位图表示一个数就是数组的一个单位，这样如果数据夸度很大，存在很多稀疏数据，位图中就会有很多空闲的位置，布隆过滤器就是解决这个问题

原理：
使用 K 个哈希函数，对同一个数字进行求哈希值，那会得到 K 个不同的哈希值，这 K 个数字作为位图中的下标，将对应的 BitMap[X1]，BitMap[X2]，BitMap[X3]，...，BitMap[XK]都设置成 true，也就是说，我们用 K 个二进制位，来表示一个数字的存在

适用场景：
如果某个数字经过布隆过滤器判断不存在，那说明这个数字真的不存在，不会发生误判；
如果某个数字经过布隆过滤器判断存在，这个时候才会有可能误判，有可能并不存在

所以，布隆过滤器适合业务上对误判有一定的容忍度，例如误判为存在，对网页抓取来说并不会影响很大





问题：数据爬取过程中，同一个网页链接有可能被包含在多个页面中，这就会导致爬虫在爬取的过程中，重复爬取相同的网页

思路：
可以将已经爬取过的url存储起来，对于要爬取的网页可以先从列表中查询是否存在，如果存在就是不再爬取，如果不存在继续爬取

涉及到的操作：
添加一个url、查询一个url

要求：
面对大量的数据可以快速的查找和添加
假设url平均长度64字节，10亿个url

符合要求的数据结构：
散列表、红黑树、跳表

散列表：
　大约需要64亿字节(约60GB)，由于散列冲突的原因，不能装载因子设置的太大，使用链表法时存储指针数据也会消耗内存，顾使用的内存量会远远大于60GB，采用分治思想也可以解决问题，
　可以优化的地方：链表不是连续存储的所以不能一下加载到cpu中，链表中的url对比的过程也比较耗时，内存方面如果要求很高就需要考虑另外的方案

使用布隆过滤器：
CPU 只需要将网页链接从内存中读取一次，进行多次哈希计算，理论上讲这组操作是 CPU 密集型的。而在散列表的处理方式中，需要读取散列值相同（散列冲突）的多个网页链接，分别跟待判重的网页链接，进行字符串匹配。这个操作涉及很多内存数据的读取，所以是内存密集型的