大数据Top K算法是面试中常见的问题之一

其考察的内容主要是对数据结构的理解和应用能力

应用场景：

搜索引擎会通过日志文件把用户每次检索串都记录下来，每个查询串长度为1-255字节。假设目前有一亿个记录，这些查询串的重复度比较高，去重后可能不超过3千万个。请你统计最热门的10个查询，要求使用内存不超过2G。

第一步：

Query统计 统计每个Query出现的次数

1. 直接排序法

首先对所有的Query都进行排序，然后遍历排好的Query，然后再统计每个Query出现的次数。显然内排序是无法装下这么多Query的，

这里可以采用外排序，常用的外排序是归并排序，时间复杂度是O(NlogN)

排序完成后统计每个Query出现的次数，遍历的时间复杂度是O(N)，因此该算法总体时间复杂度是O(NlogN)

1. HashTable 法

由于记录的重复度比较高，去重后可以完全放入到内存中，那么可以通过HashTable来统计每个记录出现的次数，HashTable的查询速度非常快，几乎是O(1)的时间复杂度。

因此这种方法的最终时间复杂度是O(N)的

第二步：找出top 10

这个思路很简单，可以通过堆来实现。

二、中位数算法

中位数算法也是面试常问的内容之一，其中中位数算法本质是选择第k大的数的经典问题。

1. 基本思路

* 归并排序，将数据排序后，选取第k个数，时间复杂度O（nlogn）
* 基数排序，数据排序后，选取第k个数，时间复杂度O(n)
* 维护一个k个元素的最大堆，存储当前遇到的最小的k个数，时间复杂度O(nlogk)，缺点：k=n/2时，等价为O(nlogn)
* 部分选择排序：把最小的放在第1位，知道第k位为止，时间复杂O(kn)，实现简单
* 部分快速排序（快速选择法），每次划分后判断第k个数再左右哪个部分，然后递归对应的部分，平均时间复杂度O(n)，最坏时间复杂度O(n^2)
* BFPRT算法，修改快速排序选择的主元选取规则，使用中位数的中位数作为主元。

BFRT算法主元选取算法：将数组的n个元素划分为n/5组，每组5个元素，且至多只有一个组由

三、实时排行榜问题

游戏中存在各种各样的排行榜，比如玩家的等级排名、分数排名。一个典型的游戏排行榜包括以下各个常见功能：

1. 记录每个玩家的分数，2、能更新玩家分数，3、能查询每个玩家的分数和名次，4、能够按名次查询排名前N名的玩家，5，能够查询排在指定玩家后M名的玩家。

其中，分数具有一定的范围，在1-10000之间，玩家总数N>=1千万

请设计一个种数据结构，能够高效的实现上述功能。

思路1、统计每个分数上的人数，用一个数组记录每个分数的玩家的总数，不妨设为score[]

这样在更新分数的时候，只需要对分数score[old]--，而score[new]++即可。查询排名时，只需计算大于等于x的分数的总人数。

这样分数更新的时间复杂度位O(1)，而查询排名的时间复杂度为O(10000)

这种方式的缺点是，无法查询在指定排名前的所有玩家。

思路2：

由思路1的改进版，用数组+链表的方式记录每个分数上具有的玩家总数，即将相同分数的玩家组成一个链表。当一个玩家的分数改变时，将这个玩家从旧分数的链表中取出，然后追加到新分数的链表中。这种方式。同时也记录每个分数上的玩家数目。

查询排名的时间复杂度不变，为O(10000).

更新分数的时间复杂度为O(n/10000).

其缺点是，可能某个分数上的玩家数目特别集中，将导致更新分数的速度很慢。

思路3：

思路2的改进版，从HashMap得出灵感，数组+红黑树，即将相同分数的玩家组成一个红黑树，这样在查找一个玩家的时候，能很快找到并删除，然后插入到新的分数中的时间也很快。更新的时间复杂度为O(log(n/10000))，最坏也是O(logn)。

思路4：

借鉴redis的zset的数据结构，即跳表。可以用redis实现实时排行榜

跳表的本质是多层链表，redis中跳表节点结构为

typedef struct zskiplistNode {

robj \*obj;

double score;//分数

struct zskiplistNode \*backward;//前向节点，只有第一层链表是双向链表

struct zskiplistLevel {

struct zskiplistNode \*forward;//每层对应的后继节点

unsigned int span;//后继节点的跨度，用于计算元素的排名

} level[];

} zskiplistNode;