**LeetCode以外的算法问题补充**

1. **大数据TopK问题**

**1、 1亿个数字中找出最大或最小的前100个数字（考虑判重和内存空间限制）？**

思路： 考虑将数字分散存储在不同文件中， 然后依次读取各个文件， 例如1~10000存到1.txt、 2~20000存到2.tx、以此类推；

        如果不判重，可以将前100个数字做成最大堆或最小堆的数据结构（找最大的Top100用最小堆， 找最小的Top100用最大堆）， 然后依次遍历所有数字， 符合条件时替换根节点后并重新构建堆。堆的缺点是允许有重复数字！！！

        如果要判重，则创建一个100个空间的空数组， 遍历1亿个数字依次插入值（按照升序）， 使用二分查找算法判断新值在数组中的位置并插入， 该数组最多容纳100个值。 当有101个值时先判重， 如果重复继续向后遍历， 如果值大于数组最小值则插入到指定位置，数组第一个元素移出数组， 因为数组是连续的，所以可以用内存拷贝方式赋值，只有新插入的值要单独赋值到对应的下标(原理类似于android.util.SparseArray： 这是Android开发中用来代替HashMap的，好处是：其内部实现了压缩算法，可以进行矩阵压缩，大大减少了存储空间，节约内存。此外它的查找算法是二分法，提高了查找的效率。

)。   因内存拷贝可认为不占用时间， 该思路的总会时间复杂度是O(1亿\*log100）， log100是二分查找的复杂度。

（内存拷贝方式赋值：memcpy会执行数据复制，而赋值语句只是简单将一个指针的值传给另一个指针，指针对应的存储空间的内容并不会改变）

**2、 1亿个搜索关键词找出次数最多的前100个搜索词（考虑内存空间限制）？**

思路： 核心是“分治”、“归并”和哈希，  第一次遍历将关键词散列到不同的文件中（散列算法是性能关键，哈希（理想情况下时间复杂度O(1)）函数的性能直接影响散列的结果， 尽量避免“数据倾斜”）， 同一个关键词一定会散列到同一个文件， 理想情况是所有关键词均匀散列到不同的文件中（即分治思想，将大文件分解为小问题）。

       读取每个文件并记录各关键词的次数， 做个冒泡排序， 从每个文件中排序出前100的关键词；

       取第一个文件的记录结果， 和第二个文件做“合并”， 即200个结果中排序出前100个关键词， 然后依次合并第三个、第四个。。。。第N个文件(将子结果合并为总结果)。

（数据倾斜：”数据倾斜“是大数据里的一个概念，就是数据集中到几个文件、处理器， 没均匀分散到各个处理单元。说白了就是忙的忙死，闲的闲死。）

**3、10亿int型数，统计只出现一次的数，内存为1GB的机器，时间复杂度要求O(n)**

一个int型占4字节，10亿就是40亿字节（很明显就是4GB），也就是如果完全读入内存需要占用4GB，而题目只给1GB内存，显然不可能将所有数据读入内存。

1. **位图法**：用一个bit位来标识一个int整数。
2. **分治法**：分批处理这10亿的数。

1、位图法（Bitmap）

位图法是基于int型数的表示范围这个概念的，用一个bit位来标识一个int整数，若该位为1，则说明该数出现；若该位为0，则说明该数没有出现。一个int整型数占4字节（Byte），也就是32位（bit）。那么把所有int整型数字表示出来需要2^32 bit位的空间，为了方便，我们可以把这些信息每8bit分割保存为byte数组，换算成字节单位也就是2^32 bit/8 = 2^29 Byte，大约等于512MB

那么接下来我们只需要申请一个int数组长度为 int tmp[N/32+1]即可存储完这些数据，其中N代表要进行查找的总数（这里也就是2^32），tmp中的每个元素在内存在占32位可以对应表示十进制数0~31,所以可得到BitMap表:

tmp[0]:可表示0~31

tmp[1]:可表示32~63

tmp[2]可表示64~95 。。。

数组值为1对应的数即为只出现一次的数

或者拓展：采用2-BitMap来解决，即为每个整数分配2bit，用不同的0、1组合来标识特殊意思，如00表示此整数没有出现过，01表示出现一次，11表示出现过多次，就可以找出重复的整数了，其需要的内存空间是正常BitMap的2倍

2、分治法

分治法目前看到的解决方案有**哈希分桶（Hash Buckets）**和**归并排序**两种方案。

哈希分桶的思想是先遍历一遍，按照hash分N桶（比如1000桶），映射到不同的文件中。这样平均每个文件就10MB，然后分别处理这1000个文件，找出没有重复的即可。

**4、名人问题**

有n个人他们之间认识与否用邻接矩阵表示(1表示认识，0表示不认识)，并A认识B并不意味着B认识A，也就意味着是个有向图

如果一个人是名人，他必须满足两个条件，一个是他不认识任何人，另一个是所有人必须都认识他。

我们从中可以分析出，名人必定最多只有1个

int n;

int adj[Max][Max];

int findFamous()

{

int person[Max]; // person存放还未判断是不是名人的数组

for(int i=0;i<n;i++)

{

person[i]=i;

}

int count=n;

while(count>1)

{

if(adj[0][1]) //第一个人认识第二个人,说明第一个人不是名人

person[0]=person[--count]; //从 person中删除第一个人

else

person[1]=person[--count]; //第二个人认识第一个人,说明第二个人不是名人

//从 person中删除第二个人

}

//剩下只有person[0]才有可能是名人，再进行完全判断一下！！

for(int i=1; i<n; i++)

{

if(person[0]!=i && (adj[person[0]][i]==1 || adj[i][person[0]]==0))

return -1;

}

return person[0]+1;

}