大文本找相同行的解决思路

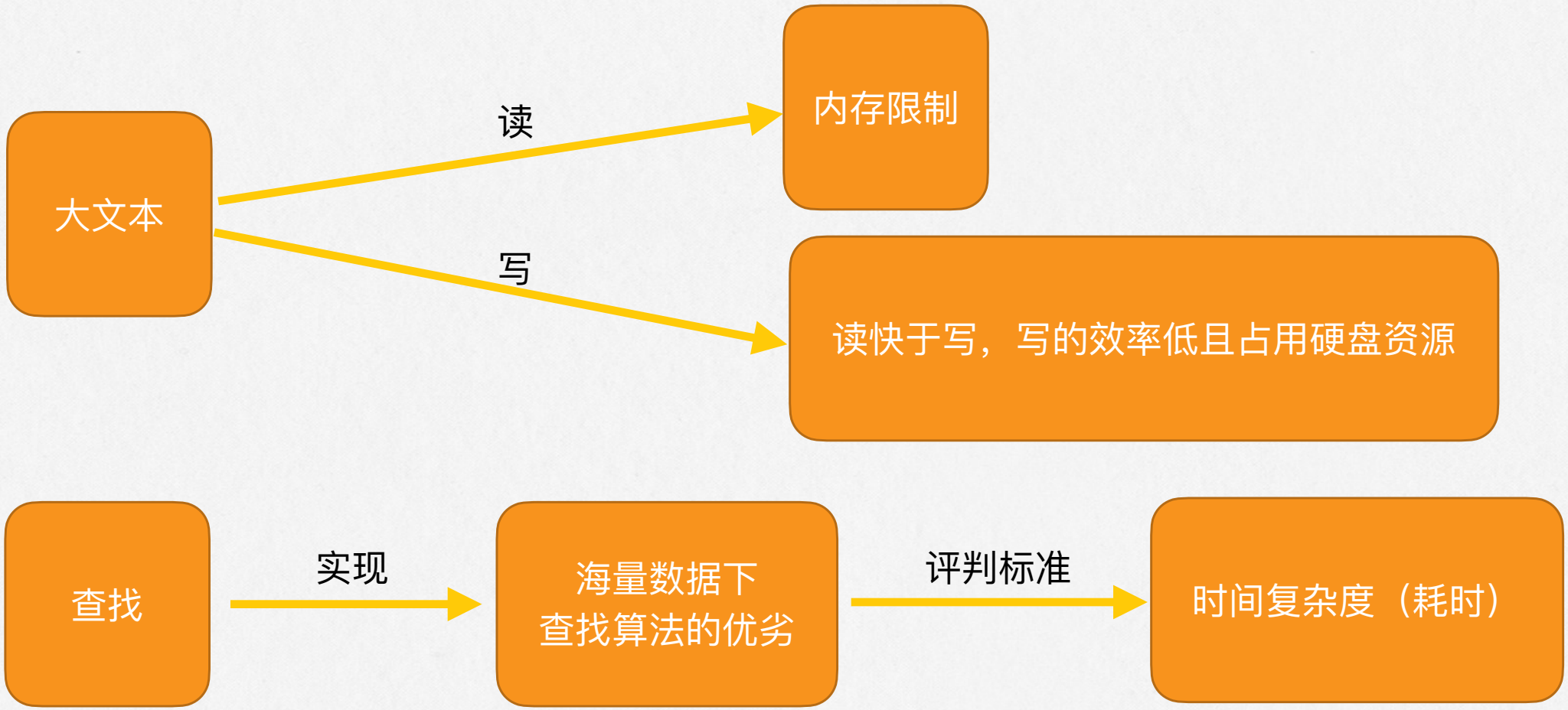
2020.5.28

目录

难点分析

题目：超大文件如何找相同行?

要思考的因素和难点：内存，运行时间，硬盘容量



方案设计

方案一.

1.载入全部文件

2.遍历文件，并把相同行的查找结果写入结果文件

优点：实现方便

缺点：内存需求大，只适合小文件，时间复杂度为O(n^2)

方案二.

1.按行载入文件（改进方案一）

2.遍历文件，并把相同行的查找结果写入结果文件

优点：内存占用小

缺点：需要额外的硬盘空间，查找的时间复杂度为O(n^2)，大文本不适用

方案三.

1.按行载入文件

2.散列（改进方案二）并把散列结果写入x个文件中（n越大散列查找越快，属于空间换时间）

3.散列结果文件进行遍历

4.查询过程删除文件的相同行（文件重写），减少下次遍历的数量，并把相同行的查找结果写入结果文件

优点：内存占用小

缺点：需要额外的硬盘空间，写文本耗时严重 ，查找速度慢，假设遇到数据行都相同的极端情况，增加了散列成本，散列函数的设计很重要

方案设计

方案四.

1.按行载入文件

2.散列并把散列结果写入x个文件中（n越大散列查找越快，属于空间换时间）

3.散列结果文件进行遍历

4.记录并跳过查找过的行（改进方案三），减少下次查找的数量，并把相同行的查找结果写入结果文件，

优点：内存占用小，查找速度较快

缺点：需要额外的硬盘空间，写文本耗时严重 ,假设遇到数据行都相同的极端情况，增加了散列成本，散列函数的设计很重要

方案五.

1.按行载入文件

2.散列并把散列结果先缓存，后一次性写入散列文件中（改进）（n越大散列查找越快，属于空间换时间）

3.散列结果文件进行遍历

4.记录并跳过查找过的行，减少下次查找的数量，并把相同行的查找结果暂存，后续一次性写入结果文件（改进方案四）

优点：内存占用小，查找速度较快，比方案四写文件耗时少

缺点：需要额外的硬盘空间，写文本耗时严重 ,假设遇到数据行都相同的极端情况，增加了散列成本，散列函数的设计很重要

方案设计

方案六.

1.按行载入文件

2.散列并把散列结果为m的散列结果暂存在内存（改进方案五）

3.排序散列集合后进行遍历查找（改进方案五）

4.并把相同行的查找结果暂存，后续一次性写入结果文件

5.重复1 2 3 4，直到散列结果都遍历完，这个过程可以加入多线程并发执行

优点：不需要额外的硬盘空间，，节约了写文件的时间，多线程加速，查找速度快

缺点：内存占用大，散列函数的设计很重要

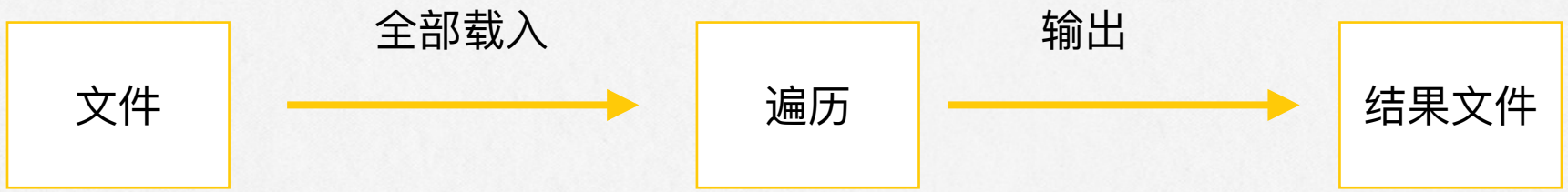
方案一

1.载入全部文件

2.遍历文件，并把相同行的查找结果写入结果文件

优点：实现方便

缺点：内存需求大，只适合小文件，时间复杂度为O(n^2)



方案二

1.按行载入文件（改进方案一）

2.遍历文件，并把相同行的查找结果写入结果文件

优点：实现方便

缺点：内存需求大，只适合小文件，时间复杂度为O(n^2)

方案三

1.按行载入文件
2.散列（改进方案二）并把散列结果写入x个文件中（n越大散列查找越快，属于空间换时间）
3.散列结果文件进行遍历
4.查询过程删除文件的相同行（文件重写），减少下次遍历的数量，并把相同行的查找结果写入结果文件

优点：内存占用小，查找速度较快
缺点：需要额外的硬盘空间，写文本耗时严重 ,假设遇到数据行都相同的极端情况，增加了散列成本，散列函数的设计很重要

方案四

1.按行载入文件
2.散列并把散列结果写入x个文件中（n越大散列查找越快，属于空间换时间）
3.散列结果文件进行遍历
4.记录并跳过查找过的行（改进方案三），减少下次查找的数量，并把相同行的查找结果写入结果文件，

优点：内存占用小，查找速度较快

缺点：需要额外的硬盘空间，写文本耗时严重 ,假设遇到数据行都相同的极端情况，增加了散列成本，散列函数的设计很重要

方案五

1.按行载入文件
2.散列并把散列结果先缓存，后一次性写入散列文件中（n越大散列查找越快，属于空间换时间）
3.散列结果文件进行遍历
4.记录并跳过查找过的行，减少下次查找的数量，把写操作合并暂存，后续一次性写入结果文件（改进方案四）

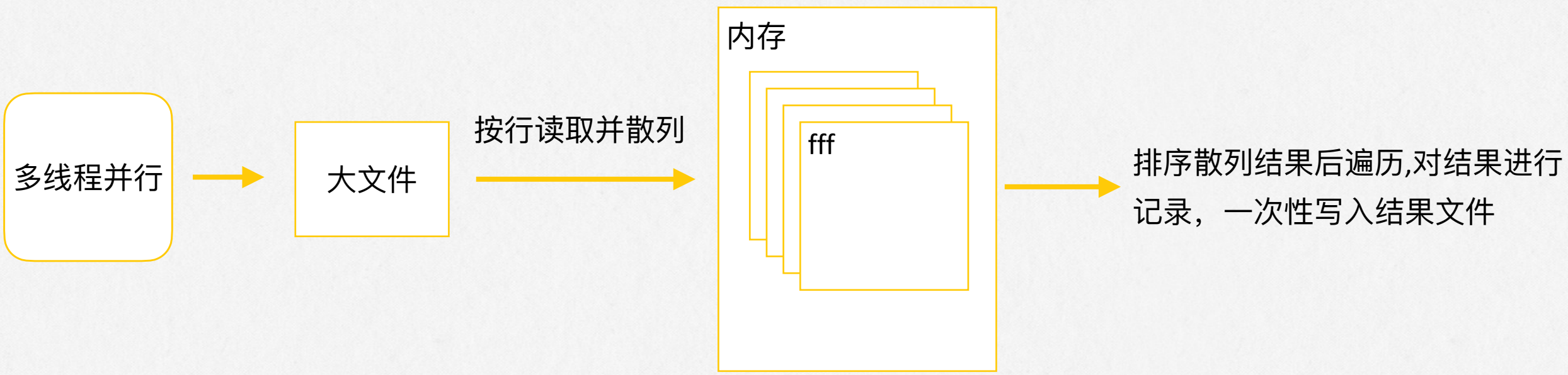
优点：内存占用小，查找速度较快

缺点：需要额外的硬盘空间，写文本耗时严重 ,假设遇到数据行都相同的极端情况，增加了散列成本，散列函数的设计很重要

方案六(最终方案)

1.按行载入文件
2.散列并把散列结果为m的散列结果暂存在内存（改进方案五）
3.排序散列集合后进行遍历查找（改进方案五）
4.并把相同行的查找结果暂存，后续一次性写入结果文件
5.重复1 2 3 4，直到散列结果都遍历完，这个过程可以加入多线程并发执行

优点：不需要额外的硬盘空间，，节约了写文件的时间，多线程加速，查找速度快
缺点：内存占用大，散列函数的设计很重要



散列函数的设计

设计目标：

1.能让相同的值在一个散列集合内

2.均匀散列

3.让每个散列集合内的数量较少，这样能降低查找时间

4.散列函数本身不能耗时过长

最终实现

```
# 能得到[0,65536]间的数字
def hash(line):
  return int(md5(line[0:4].encode("utf-8")).hexdigest()[0:4], 16)
```

方案比对和总结

	方案一	方案二	方案三	方案四	方案五	方案六
流程	1.载入全部文本 2.遍历查找 3.把重复过过的行进行记录，减少行数比较	1.按行读入文件 2.遍历查找 3.把重复过过的行进行记录，减少行数比较	1.按行读入文件 2.散列并把散列结果写入文件 3.遍历查找散列文件 4.把重复过过的行进行删除，减少遍历次数	1.按行读入文件 2.散列并把散列结果写入文件 3.遍历查找散列文件 4.记录并跳过重复过的行，减少行数比较	1.按行读入文件 2.散列并把散列结果写入文件 3.遍历查找散列文件 4.记录并跳过查找过的行，减少行数比较 5.把写操作合并，改为一次写入	1.按行读入文件 2.散列并把散列结果为n的内容暂存内存 3.排序散列结果 4.遍历查找散列结果，因为经过排序，可以先进行前一位的比较，减少总体比较次数 5.重复1 2 3直到所有的散列结果都处理完（开启多线程处理）
内存	内存消耗=文件大小	内存消耗少	内存消耗少	同方案三	同方案三	内存消耗大
硬盘	不需要额外空间	不需要额外空间	需要额外磁盘空间	同方案三	同方案三	不需要额外磁盘空间
复杂度	时间复杂度 O(n^2) 空间复杂度 O(n)	时间复杂度 O(n^2) 空间复杂度 O(n)	时间复杂度 O(n^2) 空间复杂度 O(1)	时间复杂度 O(n^2) 空间复杂度 O(n)	同方案四	时间复杂度最好 O(nlog n) 最差 O(n^2) 空间复杂度 O(1)
优点	1.实现方便 2.硬盘消耗少	1.实现方便 2.硬盘消耗少	1.数据通过散列进行了分组，查找速度较快 2.内存消耗少	1.减少了写文件操作，比方案三的查找速度更快	1.合并了写文件的操作，降低了散列耗时和查找耗时	1.数据通过散列进行了分组，查找速度快 2.没有硬盘的写入操作，总耗时短
缺点	1.受内存限制，不适用于大文件查找 2.对于大文本查找，复杂度太高	1.对于大文本查找，复杂度太高	1.写文件的io耗时严重 2.需要额外磁盘空间 3.散列函数的设计很重要	同方案三	同方案三	1.内存占用多 2.散列函数的设计很重要

运行结果

	方案一	方案二	方案三	方案四	方案五	方案六
500k	0.01 s	1.14 s	散列耗时 0.6 s 查找耗时 2.6 s	散列耗时 0.6 s 查找耗时 0.4 s	散列耗时 0.2 s 查找耗时 0.3 s	0.32 s
5.9M	1.9 s	165 s	散列耗时 20 s 查找耗时 186 s	散列耗时 20 s 查找耗时 2.2 s	散列耗时 0.3 s 查找耗时 2.3 s	0.34 s
50M	201 s	-	-	散列耗时 185 s 查找耗时 63.6 s	散列耗时 1 s 查找耗时 61.1 s	0.65 s
10G	-	-	-	-	-	71 s
60G	-	-	-	-	-	436 s

结论：方案五耗时最短，效果最好，适合处理超大文本

代码仓库：git@github.com:1036875207/BigTextFindLine.git

https://github.com/1036875207/BigTextFindLine

成长收获

1.了解了大数据背后的查找逻辑

2.通过这次的程序设计和实现，学习了散列函数的设计，大文件内容的查找方法，锻炼了思维能力

感谢观看