


近似连接

计64 翁家翌 2016011446

效果

OJ Submission ID: 9703

ID	Homework	Upload Timestamp	Status	Memory(GB)	Time(s)	Comment	Ops
9703(Marked)	exp2-final	2019/05/04 14:28:15	Correct.	0.235	1.815	JC init with query	

实现

初始化

根据观察，filename1和filename2是一样的，并且joinED和joinJC的时候这四个filename也是一样的，因此只需要一次初始化即可。代码中为 `global_init` 函数。

对于ED而言，使用Partition-based method，底层使用uint32的hash实现，之后存储 `part_vec[字符串长度][0-ED_LIMIT]` 的倒排表，其中ED_LIMIT=3由FAQ中给出。此外我还顺便维护了个抽象hash，根据观察发现输入只有空格、0-9、a-z一共37个字符，可以用一个uint64存储每个种类的字符在这个字符串中一共出现奇数次还是偶数次，记作 `state`。

对于JC而言，和第一次作业处理方法十分类似，此处不细表。

joinED

有如下几个步骤：

1. length filter: 只查询 `part_vec[target_len][*]` 的倒排表，其中 $target_len \in [len - ED_LIMIT, len + ED_LIMIT]$
2. partition filter: 分段，使用multi-match-aware method，如果有一样的就到下一步
3. prefix / suffix filter: 前缀后缀判断，因为中间已经有一段一样了，没必要重新DP。

在前缀后缀判断的时候有一步很巧妙的地方，默认ED_LIMIT=3，需要分四段。可以发现满足ED距离 ≤ 3 的字符串一定会满足一个性质：一定存在一个相同的partition，从前往后记作第i个分段 ($i \in \{0, 1, 2, 3\}$)，满足它左边的需要不超过i的编辑距离，右边的不超过ED_LIMIT-i的编辑距离。

有了这个性质，就能保证总距离能够通过前缀后缀快速地计算了。

joinJC

1. 计算下界：在给定字符串的单词中统计，而不是用全局最小值
2. 求idf，按照频次从小到大排序
3. 第一次筛选：相当于第一次作业的shortlist中查找符合条件的
4. 第二次筛选：相当于第一次作业的longlist中二分查找

总之和第一次作业代码很像。值得一提的是，可以不用调用最后的query_JC，因为在所有的shortlist和longlist中查询完毕之后可以直接计算结果。

优化

倒排表连续分配

这个挺重要的，我之前偷懒在Trie中的节点直接开vector，21s，改成在Trie中插一个token，然后凭着这个token去事先开好的倒排表里面访问，7s

读文件只读一次

由于所有的输入文件都是同一个，因此原本需要4次读写都可以变成1次。

结果计算可以利用偏序关系

还是由于filename1==filename2，如果存在result<a,b>，那么必定存在result<b,a>，因此本来需要由b查询a和由a查询b的两次询问可以合并成一次来做，然后直接在result里面一次性加上<a,b>和<b,a>，实测确实能快一倍。并且插入和查询可以交替进行，而不是像常规那样先全部插入、后全部查询。

Trie该插什么进去

以前一直都是字符串，但是可以把字符串hash之后的结果转换成字符串（只含0-9），然后再插入到Trie中。这样既减少空间，又减少时间。

字符串hash之后还能进行二次hash，我最后用的是 `hash %= 99989` 然后再把hash值插入到Trie中。（虽然可能会造成hash冲突

我还试过用std::map替换Trie，但是反而变慢了。

Update：我发现我写上面第二段话的时候是傻逼，直接开个数组就完了。

靠二次hash的trick，还能把ED中的二分直接给省略掉，但是速度提升并不是很明显