基于单调队列的Wand算法优化[文本相关度排序算法]

一. 简述:

在搜索问题中，我们经常需要对于给定的一个大小为n个关键词的询问(Query)文本以及数据库中的多个（m个）文本相似度匹配（假设最长文本关键词个数与n同级），按照相似度倒序找到与Query文本前k相似的文本并推荐给用户。使用常规的算法，很容易获得O(m*n*log(n))的时间复杂度。对于这项操作，可以考虑使用Wand算法进行优化。

*传统Wand算法的最坏时间复杂度为O(**m*n**^2)，但是通常情况下，Wand的剪枝优化可以使程序获得更低的时间复杂度（剪枝效果待测）。

二. 算法内容:

1. 问题简化:
   - 对于文本的相似度计算，我们采用关键词加权法进行计算，即将查询文本和数据文本看做一些关键词与该关键词s在该文本中出现数量v的映射集合，每个集合的权值是v，键值(判断集合间相等的值)是s。
     例：Text="a b c a b c aa dd d"
     则，集合Set={a:2,aa:1,b:2,c:2,d:1,dd:1}
     最终求出的k个文本，是该文本的set与询问文本的set对应键值的权值乘积和最高的k个文本
     （聪明的你可以在脑中构思该简化如何在C++/Python等语言中实现）
2. 一些定义:
   - n：查询文本关键词个数
   - m：数据文本个数
   - key[i]：Query文本中的第i个关键词（显然，每个关键词也有一个id）
   - key_id：是一个数组，其中key_id[i]表示由所有包含关键词key[i]的数据文本的id构成的队列，key_id[i]中元素总是单调递增的
   - key_v：是一个数组，其中每个元素是一个队列，key_v[i][j]表示关键词key[i]在id为key_id[i][j]的数据文本出现的次数
   - nowv：当前的k个答案中最小的答案的权值
   - maxv[i]：表示当前key_v[i]中最大的元素
3. 算法过程:

该算法理解之后其实非常简单，首先保证key_q按照key_q[i].front()的大小升序排序随后用下标i遍历key_q，并记录maxv的前缀和，当前缀和大于nowv时停止，和当前位置的队列第一个元素found_id=key_q[i].front()，随后，更新key_q中的所有队列，推出内部id<=found_id的所有元素，由于key_q[i]是单调的，所以这个操作很容易实现，并更新key_v、maxv等数组，使得其维护的值遵循定义，并返回你找到的可能可以替换前k优值的文本的id——found_id，将该id返回进行详细计算后确定是否能成为前k大，随后将更新后的第k大文本的权值继续传入算法从头执行，直到找完所有的id。

4. 优化

该算法有一些行之有效的细节优化。

根据上述算法，在最差情况下，会得到O(n^2*m)的时间复杂度——一共需要执行算法m次，每次需要遍历数组key_v，更新时需要对key_v的每个队列重新求得maxv，共计O(n^2*m)；虽然剪枝可对其进行执行层面上的优化，但是理论复杂度依旧如此。

考虑如何快速的更新maxv，对于所有进行更新（pop_front()）操作的队列，我们都有可能需要更新maxv，一种简单而行之有效的办法是用线段树（或同等数据结构）储存每个原始队列的单点值，并提供维护区间最大值的功能，每次更新时，只需要求出一个新的最大值即可，这样做时间复杂度O(n*logn*m)。进一步，考虑对于每一个队列，其中可能作为maxv的位置（贡献点）是确定的一些点，并且这些点的v在队列中是单调递减的，所以可用单调队列进行算法优化，单调队列存储这些可能的贡献点，每次只需要删除其中id不满足条件的点，剩下队列头部一定是待求的最大值，至此，该算法被优化为O(n*m)。

三. 代码实现：