数据库专题训练——第一次小作业

MaskRay

2015 年 4 月 6 日

1 任务

实现近似串查询，近似串有两种度量方式：Levenshtein edit distance 和 Jaccard index。需要实现 `SimSeacher` 类的 `createIndex`、`searchED`、`searchJaccard` 三个方法。

代码中实现了 brute-force、tournament sort、MergeSkip[**LLL08**] 和 DivideSkip[**LLL08**] 四种查询算法。

2 BruteForce

枚举所有字符串，和查询字符串一一进行 Levenshtein edit distance 或 Jaccard index 的计算，若满足阈值条件则添加到存放结果的向量。

3 Tournament sort

采用 filter-and-verification framework，代码中实现了索引、Filter 和 Verification 三个部分。

3.1 索引

索引的数据结构是若干倒排索引，另外有一个 hash table 为 Q-gram 到倒排索引的映射。

执行 `createIndex` 构建索引时，对于每个输入的字串，使用类似 Rabin-Karp string search 算法的方式，获取当前长为 q 的窗口中的 Q-gram，计算出散列值，找到对应的倒排索引，并在该索引末端加入当前行号。如果输入字串有重复的 Q-gram，那么当前行号可能会在倒排索引中插入多次。然后窗口向右移动一格，把之前的散列值通过 rolling hash 计算出下一时刻的值。

输入文件读入完毕后，对于 hash table 中的每个倒排索引 (必然是非空的)，在末端加入作为哨兵元素的无穷大。

3.2 Filter

对于一个查询，使用类似构建索引时的 Rabin-Karp string search algorithm，对于每个长为 q 的窗口中的 Q-gram，找到对应的倒排索引，把指向首元素的指针放入列表 L 中。设查询字串长度位 n，那么 L 中将会有 $n-q+1$ 个指针 (可能重复)。

3.2.1 建立 binary heap

列表 L 的元素是指向倒排索引的指针，以指针指向的值为关键字使用 Floyd's algorithm 在 $\Theta(|L|)$ 时间内建立 binary heap L。

3.2.2 Tournament sort 进行 N-way merge

1. 堆顶指针指向的值为 old，若 old 为无穷大则返回
2. 计算堆顶元素指向的值 old 的出现次数，若大于等于阈值 *overlap* 则添加到候选集中
3. 把堆顶元素指针向前移动一格 (即指向对应的倒排索引的下一项)，若下一项仍等于原来指向的值则继续移动
4. 上述操作后堆顶指针指向的值变大了，对它进行下滤操作
5. 若新的堆顶指向的值等于 old 则跳转到步骤 3

之后对候选集的每个元素进行检验，是否满足 Jaccard index 或 Levenshtein 编辑距离的阈值要求，输出筛选后的。

3.3 Verification

3.3.1 Levenshtein edit distance

对于 `searchED` 操作，需要对每个候选字串和查询字串计算编辑距离。

两个字符串的 Levenshtein edit distance 可以使用 $\Theta(nm)$ 的 Needleman–Wunsch algorithm 计算。

注意到代码中使用到编辑距离的地方都有阈值 *th* 限制，如果编辑距离超过阈值，那么它的实际值无关紧要。因此我们可以只计算动态规划矩阵中对角线带状区域的状态。

另外当其中某个字符串的长度小于等于 128 时，还可以采用 bit vector 的算法 [**edit03**] 加速到 $\Theta(n)$。

3.3.2 Jaccard index

采用 scan count 的方式。先用 rolling hash 计算查询串所有 Q-gram 的标号，在计数容器中增加一。然后对于候选串的所有 Q-gram，若计数容器中的值大于零则减去零并加到答案中。再便利候选串的所有 Q-gram，把减去的值再加回来。

4 MergeSkip

和 tournament sort 基本结构相同，对 Filter 部分做了优化，即优化了统计每个字串的出现次数，过程如下：

1. 堆顶指针指向的值为 old，若 old 为无穷大则返回
2. 计算堆顶元素指向的值 old 的出现次数，若大于等于阈值 *overlap* 则添加到候选集中
3. 把堆顶元素指针向前移动一格 (即指向对应的倒排索引的下一项)，若下一项仍等于原来指向的值则继续移动

4. 上述操作后堆顶指针指向的值变大了，对它进行下滤操作
5. 若新的堆顶指向的值等于 old 则跳转到步骤 3
6. 从堆中弹出 $overlap - 1$ 个元素，使用二分检索把这些倒排索引指针移动至大于当前堆顶指向的值，再重新插入堆中

5 DivideSkip

结合了 MergeSkip 和 MergeOut。

1. 使用启发函数把倒排索引划分为长索引和短索引两类，长索引有 $nlong < n$ 个，其中 n 为索引个数
2. 对短索引采用 MergeSkip 算法，对于短索引中所有出现次数不少于 $overlap - nlong$ 的元素，在所有长索引中二分检索
3. 将出现次数不少于 $overlap$ 的元素添加到候选集中
4. 把堆顶元素指针向前移动一格 (即指向对应的倒排索引的下一项)，若下一项仍等于原来指向的值则继续移动
5. 上述操作后堆顶指针指向的值变大了，对它进行下滤操作
6. 若新的堆顶指向的值等于 old 则跳转到步骤 4
7. 从堆中弹出 $overlap - nlong - 1$ 个元素，使用二分检索把这些倒排索引指针移动至大于当前堆顶指向的值，再重新插入堆中

6 其他

Tournament 类使用了 curiously recurring template pattern。