

# Database Project

## Part1

### Levenshtein Distance

据说这个是某个🇷🇺数学家整出的算法（就是 *Levenshtein* 我不知道怎么读的词的数学家），惊呼毛子牛皮。

#### 1. 算法思路

##### 1. 概念介绍：

- 这个算法就是在计算两个字串之间，由一个转换成另一个所需的最少编辑操作次数
- 然后这个大概就是论文查重网站的基础算法吧（我猜的，么得考证过）

##### 2. 算法思路

$$\text{lev}_{a,b}(i, j) = \begin{cases} \max(i, j) & \text{if } \min(i, j) = 0, \\ \min \begin{cases} \text{lev}_{a,b}(i-1, j) + 1 \\ \text{lev}_{a,b}(i, j-1) + 1 \\ \text{lev}_{a,b}(i-1, j-1) + 1_{(a_i \neq b_j)} \end{cases} & \text{otherwise.} \end{cases}$$

##### 3. 这个算法的基础是像下图一样更改字符串的方法（djv快点想怎么说）

|         |              |
|---------|--------------|
| replace | innsert      |
| delete  | you are here |

##### 4. 大概过程：

- 初始化一个 table 矩阵 (M,N)，M 和 N 分别是两个输入字符串的长度
- 矩阵可以从左上角到右下角进行填充，每个水平或垂直跳转分别对应于一个插入或一个删除通过定义每个操作的成本为1，如果两个字符串不匹配，则对角跳转的代价为1，否则为0，简单来说就是：
  - 如果 [i][j] 位置的两个字符串相等，则从 [i][j] 位置左加1，上加1，左上加0，然后从这三个数中取出最小的值填充到 [i][j]
  - 如果 [i][j] 位置的两个字符串不相等，则从 [i][j] 位置左、左上、上三个位置的值中取最小值，这个最小值加1（或者说这三个值都加1然后取最小值），然后填充到 [i][j]
- 按照上面规则 Table 矩阵 (M,N) 填充完毕后，最终矩阵右下角的数字就是两个字符串的 Levenshtein Distance 值

5. 好我编不下去了自行参照 [YouTube](#) 老哥教学视屏

## 2. 过程中的种种问题

- text 转换问题

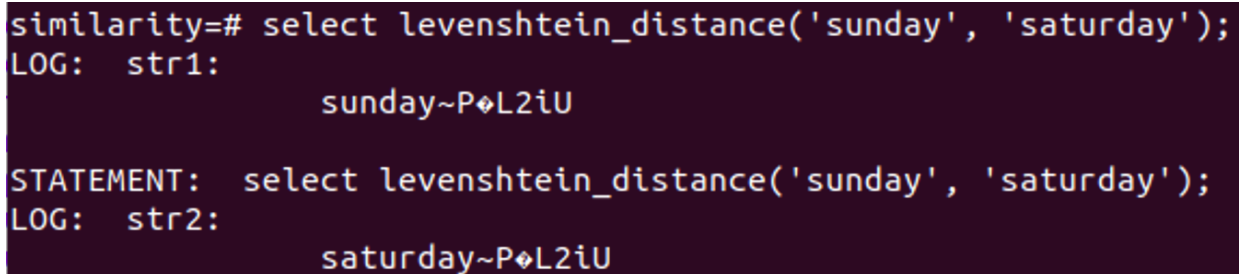
研究了好久这个 text 是个啥

```
typedef struct varlena text;
struct varlena{
char    vl_len_[4];    /* Do not touch this field directly! */
char    vl_dat[1];
};
```

看了源码后，我寻思 vl\_dat 应该就是string了， 于是不假思索的写了如下代码转换

```
text * str_01 = PG_GETARG_DATUM(0);
text *txt_02 = PG_GETARG_DATUM(1);
int32 result=1;
char *str1 = str_01->vl_dat;
char *str2 = txt_02->vl_dat;
int len_1 = strlen(str1);
int len_2 = strlen(str2);
```

然后运行后发现如图情况



```
similarity=# select levenshtein_distance('sunday', 'saturday');
LOG:  str1:
        sunday~P♦L2iU

STATEMENT:  select levenshtein_distance('sunday', 'saturday');
LOG:  str2:
        saturday~P♦L2iU
```

然后发现事情没有那么简单，他竟然是字符串后面加东西的？这时我果断放弃了这个危险的想法，转用 text\_to\_cstring()（这个函数是在看text结构的时候看到的，于是我想我为啥一定要自己写一个新的呢？）

```
text * str_01 = PG_GETARG_DATUM(0);
text *txt_02 = PG_GETARG_DATUM(1);
int32 result=1;
char *str1 = text_to_cstring(str_01);
char *str2 = text_to_cstring(txt_02);
int len_1 = strlen(str1);
int len_2 = strlen(str2);
```

## 3. 代码解释

- 代码总览

```

int min(int a, int b, int c){
    int reslut = a;
    if ( a > b){
        reslut = b;
        if (b > c)
            reslut = c;
    }
    else if(a > c)
        reslut = c;
    return reslut;
}

char TOLOWER(char c){
    if(c >= 'A' && c <= 'Z')
        c += 32;
    return c;
}

Datum levenshtein_distance(PG_FUNCTION_ARGS){
    text * str_01 = PG_GETARG_DATUM(0);
    text *txt_02 = PG_GETARG_DATUM(1);
    int32 result=1;
    // text to char c
    char *str1 = text_to_cstring(str_01);
    char *str2 = text_to_cstring(txt_02);
    int len_1 = strlen(str1);
    int len_2 = strlen(str2);
    int table[256][256];
    int i, j;
    elog(LOG, "str1:\n\t%s\n", str1);
    elog(LOG, "str2:\n\t%s\n", str2);
    elog(LOG, "len1:\n\t%d\n", len_1);
    elog(LOG, "len2:\n\t%d\n", len_2);
    for(i = 1; i <= len_1; ++i)
        table[i][0] = i;
    for(i = 1; i <= len_2; ++i)
        table[0][i] = i;
    for(j = 1; j <= len_2; ++j )
        for(i = 1; i <= len_1; ++i){
            if(TOLOWER(str1[i-1]) == TOLOWER(str2[j-1]))
                table[i][j] = table[i-1][j-1];
            else
                table[i][j] = min(table[i-1][j-1], table[i-1][j], table[i][j-1])
        }
    result = table[len_1][len_2];
    PG_RETURN_INT32(result);
}

```

- int min(int a, int b, int c) :

用来计算三者中最小值；

- `char TOLOWER(char c)` :  
用来讲大写字母转换成小写，即实现了大小写不敏感的需求。  
之后的 Jaccard Index 也要用
- `Datum levenshtein_distance(PG_FUNCTION_ARGS)` :  
讲道理不知道 `Datum` 是个啥，但是不影响我写这个函数，然后这个函数已经在上面介绍过啦，这里就划划水了

#### 4. 结果：

- `select levenshtein_distance('sunday', 'sunday');`

```

similarity=# select levenshtein_distance('sunday', 'sunday');
LOG:  str1:
          sunday

STATEMENT:  select levenshtein_distance('sunday', 'sunday');
LOG:  str2:
          sunday

STATEMENT:  select levenshtein_distance('sunday', 'sunday');
LOG:  len1:
          6

STATEMENT:  select levenshtein_distance('sunday', 'sunday');
LOG:  len2:
          6

STATEMENT:  select levenshtein_distance('sunday', 'sunday');
          levenshtein_distance
-----
          0
(1 row)

```

- `select levenshtein_distance('sunday', 'Monday'); levenshtein_distance`

```

similarity=# select levenshtein_distance('sunday', 'Monday'); levenshtein_distance
LOG:  str1:
           sunday

STATEMENT:  select levenshtein_distance('sunday', 'Monday');
LOG:  str2:
           Monday

STATEMENT:  select levenshtein_distance('sunday', 'Monday');
LOG:  len1:
           6

STATEMENT:  select levenshtein_distance('sunday', 'Monday');
LOG:  len2:
           6

STATEMENT:  select levenshtein_distance('sunday', 'Monday');
           levenshtein_distance
-----
                2
(1 row)

```

- select levenshtein\_distance('sunday', 'saturday');

```

similarity=# select levenshtein_distance('sunday', 'saturday');
LOG:  str1:
           sunday

STATEMENT:  select levenshtein_distance('sunday', 'saturday');
LOG:  str2:
           saturday

STATEMENT:  select levenshtein_distance('sunday', 'saturday');
LOG:  len1:
           6

STATEMENT:  select levenshtein_distance('sunday', 'saturday');
LOG:  len2:
           8

STATEMENT:  select levenshtein_distance('sunday', 'saturday');
           levenshtein_distance
-----
                3
(1 row)

```