

# 实验三报告

程序语言: python

姓名: 郑子睿

学号: 22920212204317

专业: 计算机科学与技术

内容: 代码查重

## 一、问题描述

给定两个程序, 判断他们的相似性

## 二、算法思想

本实验针对C++代码, 尤其是oj上提交的C++代码做查重工作

1. 将两个 .cpp 文件中的代码作为字符串读入;
2. 去除代码中的头文件、using 语句、注释、输入输出语句、const 语句、宏定义、读入优化和 return 语句, 以及空白字符, 将两份代码分别压缩成两个字符串;
3. 计算两个字符串的最短编辑距离;
4. 将 最短编辑距离/较短的字符串长度 作为判断依据
  1. 如果 该值  $\leq 0.5$ , 则判断两份代码可疑, 将重复率定义为  $(1 - \text{最短编辑距离} / \text{较短的字符串长度}) * 100\%$ 。输入重复率和两份代码;
  2. 否则, 判断这两份代码不可疑;
5. 将结果保存在一个 .txt 文件中。

## 三、描述算法

计算两个字符串的最短编辑距离

这个问题要用到动态规划算法

1. 状态表示: 定义  $f[i][j]$  是将子串  $a[1 \sim i]$  变成  $b[1 \sim j]$  的最短编辑距离
2. 状态转移方程
  1. 以两个子串的最后一个字符作为分隔点
  2. 如果在  $a[1 \sim i]$  中删去  $a[i]$  可以使两个子串相同, 则必有  $a[1 \sim i-1]$  和  $b[1 \sim j]$  相同, 则  $f[i][j] = f[i-1][j] + 1$
  3. 如果在  $a[1 \sim i]$  中插入  $b[j]$  可以使两个子串相同, 则必有  $a[1 \sim i]$  和  $b[1 \sim j-1]$  相同, 则  $f[i][j] = f[i][j-1] + 1$
  4. 如果修改  $a[i]$  可以使两个子串相同, 则必有  $a[1 \sim i-1]$  和  $b[1 \sim j-1]$  相同
    1. 如果  $a[i] == b[j]$ , 则不需要修改,  $f[i][j] = f[i-1][j-1]$
    2. 否则需要修改,  $f[i][j] = f[i-1][j-1] + 1$
5. 综上, 状态转移方程为  $f[i][j] = \min(f[i-1][j] + 1, f[i][j-1] + 1, f[i-1][j-1] + a[i] == b[j])$

```

1 def check(a, b):
2     a, b = ' ' + a, ' ' + b # 下标从1开始
3     f = [[0 for _ in range(len(b) + 1)] for _ in range(len(a) + 1)]
4     # 初始化边界
5     # 1. 如果字符串b的长度为0, 则要删除a中所有的字符, 编辑距离为子串a的长度
6     for i in range(0, len(a) + 1):
7         f[i][0] = i
8     # 2. 如果字符串a的长度为0, 则要插入b中所有的字符, 编辑距离为子串b的长度
9     for i in range(0, len(b) + 1):
10        f[0][i] = i
11    # 基于状态转移方程, 求出最短编辑距离
12    for i in range(1, len(a)):
13        for j in range(1, len(b)):
14            f[i][j] = min(f[i - 1][j] + 1, f[i][j - 1] + 1)
15            if a[i] == b[j]:
16                f[i][j] = min(f[i - 1][j - 1], f[i][j])
17            else:
18                f[i][j] = min(f[i][j], f[i - 1][j - 1] + 1)
19    # 结果就是将字符串a变成b的最短编辑距离
20    return f[len(a) - 1][len(b) - 1]

```

## 四、验证算法

### 具体代码

#### 1. 导入库

```

1 import sys
2 import string

```

#### 2. 读取并解析代码, 将删减后的结果存在列表 str[] 中

```

1 file1, file2 = sys.argv[1], sys.argv[2]
2 detail = []
3 with open(file1, 'r', encoding='utf-8') as f:
4     res = f.readlines()
5     detail.append(res)
6 with open(file2, 'r', encoding='utf-8') as f:
7     res = f.readlines()
8     detail.append(res)
9
10
11 str = []
12 for cppFile in detail:
13     t = ''
14     for i in range(len(cppFile)):
15         line = cppFile[i].strip()
16         if line == '' or line.startswith('#include') or
17 line.startswith('using') or line.startswith('//') or
18 line.startswith('printf') or line.startswith('return') or
19 line.startswith('cin') or line.startswith('cout') or
20 line.startswith('scanf') or line.startswith('ios::sync_with_stdio(false)')
21 or line.startswith('const') or line.startswith('#define'):
22             continue
23         else:

```

```

19         for j in range(len(line)):
20             if line[j: j + 2] == '//':
21                 break
22             if line[j] not in string.whitespace:
23                 t += line[j]
24         str.append(t)

```

### 3. 计算两份代码的最短编辑距离

```

1 def check(a, b):
2     a, b = ' ' + a, ' ' + b
3     f = [[0 for _ in range(len(b) + 1)] for _ in range(len(a) + 1)]
4     for i in range(0, len(a) + 1):
5         f[i][0] = i
6     for i in range(0, len(b) + 1):
7         f[0][i] = i
8     for i in range(1, len(a)):
9         for j in range(1, len(b)):
10            f[i][j] = min(f[i - 1][j] + 1, f[i][j - 1] + 1)
11            if a[i] == b[j]:
12                f[i][j] = min(f[i - 1][j - 1], f[i][j])
13            else:
14                f[i][j] = min(f[i][j], f[i - 1][j - 1] + 1)
15     return f[len(a) - 1][len(b) - 1]
16
17 editDist = check(str[0], str[1])

```

### 4. 计算可疑指标, 输出结果

```

1 editDist = check(str[0], str[1])
2 duplicateCheckRate = editDist / min(len(str[0]), len(str[1]))
3 if duplicateCheckRate <= 0.5:
4     print(f'\n可疑, 重复率为{(1 - duplicateCheckRate) * 100:.2f}%\n')
5     for i in range(len(detail)):
6         print('-'* 50)
7         print('\n')
8         for j in range(len(detail[i])):
9             print(detail[i][j], end = "")
10        print('\n')
11 else:
12     print(f'\n不可疑')

```

## 结果展示

### 1. 筛查两份思路 and 代码实现基本相同的代码

#### 测试代码

```

1 // test1.cpp, 最长公共子序列算法的cpp实现
2 #include <bits/stdc++.h>
3 using namespace std;
4
5 // 这是一个测试
6 const int N = 1010;
7 char a[N], b[N];
8 int n, m, f[N][N];

```

```

9
10 int main()
11 {
12     cin >> n >> m;
13     scanf("%s%s", a + 1, b + 1);
14
15     for (int i = 1; i <= n; i ++ )
16         for (int j = 1; j <= m; j ++ )
17         {
18             f[i][j] = max(f[i - 1][j], f[i][j - 1]);
19             if (a[i] == b[j]) f[i][j] = max(f[i][j], f[i - 1][j - 1] + 1);
20         }
21
22     cout << f[n][m];
23     return 0;
24 }
25 // test2.cpp, 最长公共子序列算法的cpp实现, 对代码作为修改
26 #include <iostream>
27 using namespace std;
28 const int N = 1010;
29 int n, m;
30 char a[N], b[N];
31 int f[N][N];
32 int main() {
33     cin >> n >> m >> a + 1 >> b + 1;
34     for (int i = 1; i <= n; i++) {
35         for (int j = 1; j <= m; j++) {
36             if (a[i] == b[j]) {
37                 f[i][j] = f[i - 1][j - 1] + 1;
38             } else {
39                 f[i][j] = max(f[i - 1][j], f[i][j - 1]);
40             }
41         }
42     }
43     cout << f[n][m] << '\n';
44     return 0;
45 }

```

测试结果

可疑, 重复率为68.28%

```
Windows PowerShell
PS D:\MY_CODE\python_program> python .\duplicateChecking.py .\test1.cpp .\test3.cpp
PS D:\MY_CODE\python_program> cat .\result.txt

可疑, 重复率为68.29%

-----

#include <bits/stdc++.h>
using namespace std;

// 这是一个测试
const int N = 1010;
char a[N], b[N];
int n, m, f[N][N];

int main()
{
    cin >> n >> m;
    scanf("%s%s", a + 1, b + 1);

    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= m; j++)
        {
            f[i][j] = max(f[i - 1][j], f[i][j - 1]);
            if (a[i] == b[j]) f[i][j] = max(f[i][j], f[i - 1][j - 1] + 1);
        }

    cout << f[n][m];
    return 0;
}

-----

#include <iostream>
using namespace std;
const int N = 1010;
int n, m;
char a[N], b[N];
int f[N][N];
int main() {
    cin >> n >> m >> a + 1 >> b + 1;
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= m; j++) {
            if (a[i] == b[j]) {
                f[i][j] = f[i - 1][j - 1] + 1;
            } else {
                f[i][j] = max(f[i - 1][j], f[i][j - 1]);
            }
        }
    }
    cout << f[n][m] << '\n';
    return 0;
}
```

## 2. 筛查两份不同的代码

### 测试代码

```
1 // test3.cpp 归并排序的递归实现
2 #include <bits/stdc++.h>
3 using namespace std;
4 const int N=100008;
5 int a[N],tmp[N];
6
7 void merge_sort(int a[], int l, int r){
8     if(l>=r) return;
9     int mid=l+r>>1;
10    merge_sort(a,l,mid), merge_sort(a,mid+1,r);
11
12    int k=0, i=l, j=mid+1;
13    while(i<=mid&& j<=r){
14        if(a[i]<=a[j]) tmp[k++]=a[i++];
15        else tmp[k++]=a[j++];
16    }
17
18    while(i<=mid) tmp[k++]=a[i++];
19    while(j<=r) tmp[k++]=a[j++];
20
21    for(int i=l,j=0;i<=r;i++,j++){
22        a[i]=tmp[j];
23    }
24
25 }
26
27 int main(){
28     int n;
29     scanf("%d",&n);
30     for(int i=0;i<n;i++){
31         scanf("%d",&a[i]);
32     }
33     merge_sort(a,0,n-1);
```

```

34     for(int i=0;i<n;i++){
35         printf("%d ",a[i]);
36     }
37     return 0;
38 }
39 // test4.cpp 归并排序的非递归实现
40 // 非递归写法二
41 #include <bits/stdc++.h>
42 using namespace std;
43
44 using ll = long long;
45 using pii = pair<int, int>;
46 const int N = 1e5 + 10;
47 int a[N], t[N];
48
49 void merge(int l, int r)
50 {
51     if (l >= r) return;
52     int mid = l + r >> 1;
53     int i = l, j = mid + 1, k = 0;
54     int len = r - l + 1;
55     if (log2(len) != floor(log2(len)))
56         mid = l - 1 + pow(2, floor(log2(len))), j = mid + 1;
57
58     while (i <= mid && j <= r)
59     {
60         if (a[i] <= a[j]) t[k++] = a[i++];
61         else t[k++] = a[j++];
62     }
63     while (i <= mid) t[k++] = a[i++];
64     while (j <= r) t[k++] = a[j++];
65     for (int i = l, j = 0; i <= r; i++, j++) a[i] = t[j];
66 }
67
68 int main()
69 {
70     ios::sync_with_stdio(false), cin.tie(nullptr), cout.tie(nullptr);
71
72     int n;
73     cin >> n;
74     for (int i = 1; i <= n; i++) cin >> a[i];
75     for (int s = 1; ; s *= 2)
76     {
77         for (int i = 1; i <= n; i += s)
78         {
79             if (i + s - 1 > n)
80             {
81                 merge(i, n);
82                 break;
83             }
84             else merge(i, i + s - 1);
85         }
86         if (s > n) break;
87     }
88     for (int i = 1; i <= n; i++) cout << a[i] << ' ';
89     return 0;
90 }

```

测试结果，重复率低于50%，不可疑

```
PS D:\桌面\22920212204317_郑子睿_第3次报告\代码> python .\duplicateChecking.py .\test3.cpp .\test4.cpp
PS D:\桌面\22920212204317_郑子睿_第3次报告\代码> cat .\result.txt

不可疑
PS D:\桌面\22920212204317_郑子睿_第3次报告\代码>
```

## 五、结论

---

本实验通过去芜存菁，将代码的核心部分压缩成两个字符串，运用最短编辑距离算法计算重复率作为判断依据从而得到结果。

通过本实验，我认识到了Python这门语言的便捷性，提高了对Python代码的熟练度。

当然，本实验仅仅能解决最基础的查重问题，目前市场上有斯坦福大学研发的Moss查重等优秀的产品，它们所用的技术手段是我所不知道的。但是我觉得可以从汇编语言等角度更彻底地解决代码查重问题。本实验的算法将会继续改进。