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<=0.5`，则判断两份代码可疑，将重复率定义为`(1-最短编辑距离/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~i]`变成`b[1~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~i]`中删去`a[i]`可以使两个子串相同，则必有`a[1~i-1]`和`b[1~j]`相同，则`f[i][j] = f[i - 1][j] + 1`
    3. 如果在`a[1~i]`中插入`b[j]`可以使两个子串相同，则必有`a[1~i]`和`b[1~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~i-1]`和`b[1~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])`

```python
def check(a, b):
    a, b = ' ' + a, ' ' + b # 下标从1开始
    f = [[0 for _ in range(len(b) + 1)] for _ in range(len(a) + 1)]
    # 初始化边界
    # 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
    for i in range(0, len(a) + 1):
        f[i][0] = i
    # 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
    for i in range(0, len(b) + 1):
        f[0][i] = i
    # 基于状态转移方程，求出最短编辑距离
    for i in range(1, len(a)):
        for j in range(1, len(b)):
            f[i][j] = min(f[i - 1][j] + 1, f[i][j - 1] + 1)
            if a[i] == b[j]:
                f[i][j] = min(f[i - 1][j - 1], f[i][j])
            else:
                f[i][j] = min(f[i][j], f[i - 1][j - 1] + 1)
    # 结果就是将字符串a变成b的最短编辑距离
    return f[len(a) - 1][len(b) - 1]
```

四、验证算法

具体代码

1. 导入库

```python
import sys
import string
```

2. 读取并解析代码，将删减后的结果存在列表`str[]`中

```python
file1, file2 = sys.argv[1], sys.argv[2]
detail = []
with open(file1, 'r', encoding='utf-8') as f:
    res = f.readlines()
    detail.append(res)
with open(file2, 'r', encoding='utf-8') as f:
    res = f.readlines()
    detail.append(res)


str = []
for cppFile in detail:
    t = ''
    for i in range(len(cppFile)):
        line = cppFile[i].strip()
        if line == '' or line.startswith('#include') or
line.startswith('using') or line.startswith('//') or
line.startswith('printf') or line.startswith('return') or
line.startswith('cin') or line.startswith('cout') or
line.startswith('scanf') or line.startswith('ios::sync_with_stdio(false)')
or line.startswith('const') or line.startswith('#define'):
            continue
        else:
```

```
            for j in range(len(line)):
                if line[j: j + 2] == '//':
                    break
                if line[j] not in string.whitespace:
                    t += line[j]
    str.append(t)
```

3. 计算两份代码的最短编辑距离

```
def check(a, b):
    a, b = ' ' + a, ' ' + b
    f = [[0 for _ in range(len(b) + 1)] for _ in range(len(a) + 1)]
    for i in range(0, len(a) + 1):
        f[i][0] = i
    for i in range(0, len(b) + 1):
        f[0][i] = i
    for i in range(1, len(a)):
        for j in range(1, len(b)):
            f[i][j] = min(f[i - 1][j] + 1, f[i][j - 1] + 1)
            if a[i] == b[j]:
                f[i][j] = min(f[i - 1][j - 1], f[i][j])
            else:
                f[i][j] = min(f[i][j], f[i - 1][j - 1] + 1)
    return f[len(a) - 1][len(b) - 1]

editDist = check(str[0], str[1])
```

4. 计算可疑指标，输出结果

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

结果展示

1. 筛查两份思路和代码实现基本相同的代码

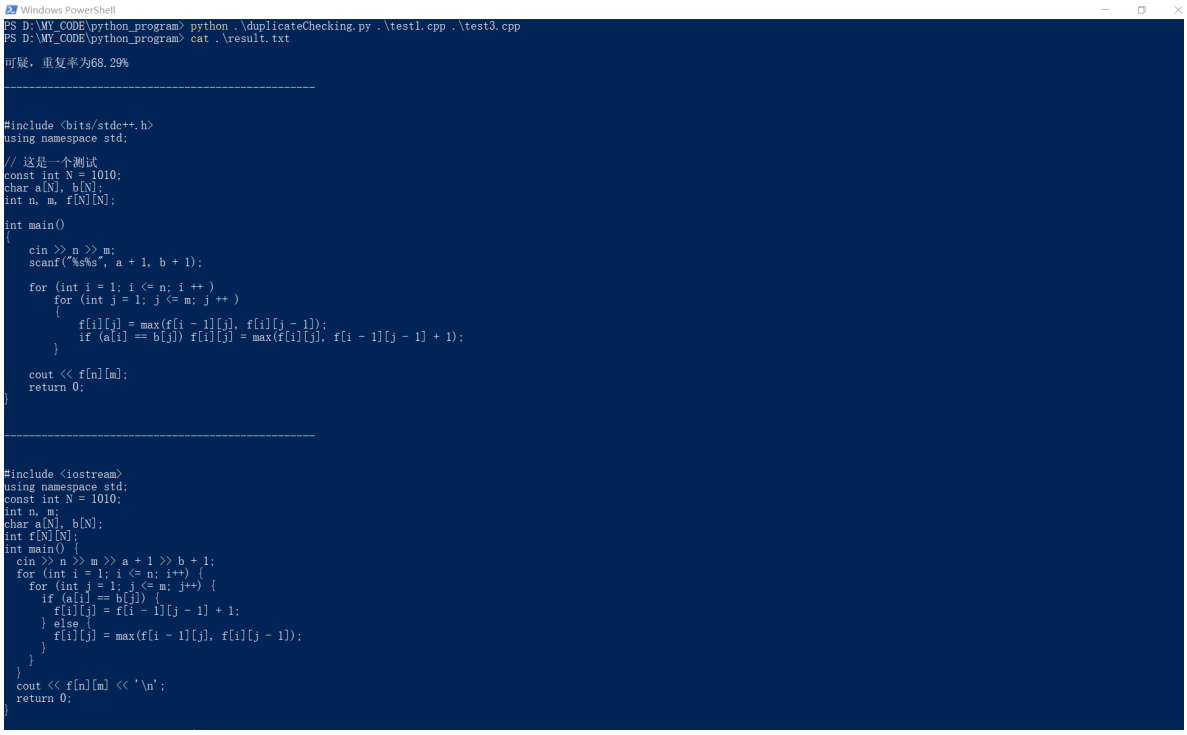
测试代码

```
// test1.cpp，最长公共子序列算法的cpp实现
#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;
}
// test2.cpp，最长公共子序列算法的cpp实现，对代码作为修改
#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;
}
```

测试结果

可疑，重复率为68.28%



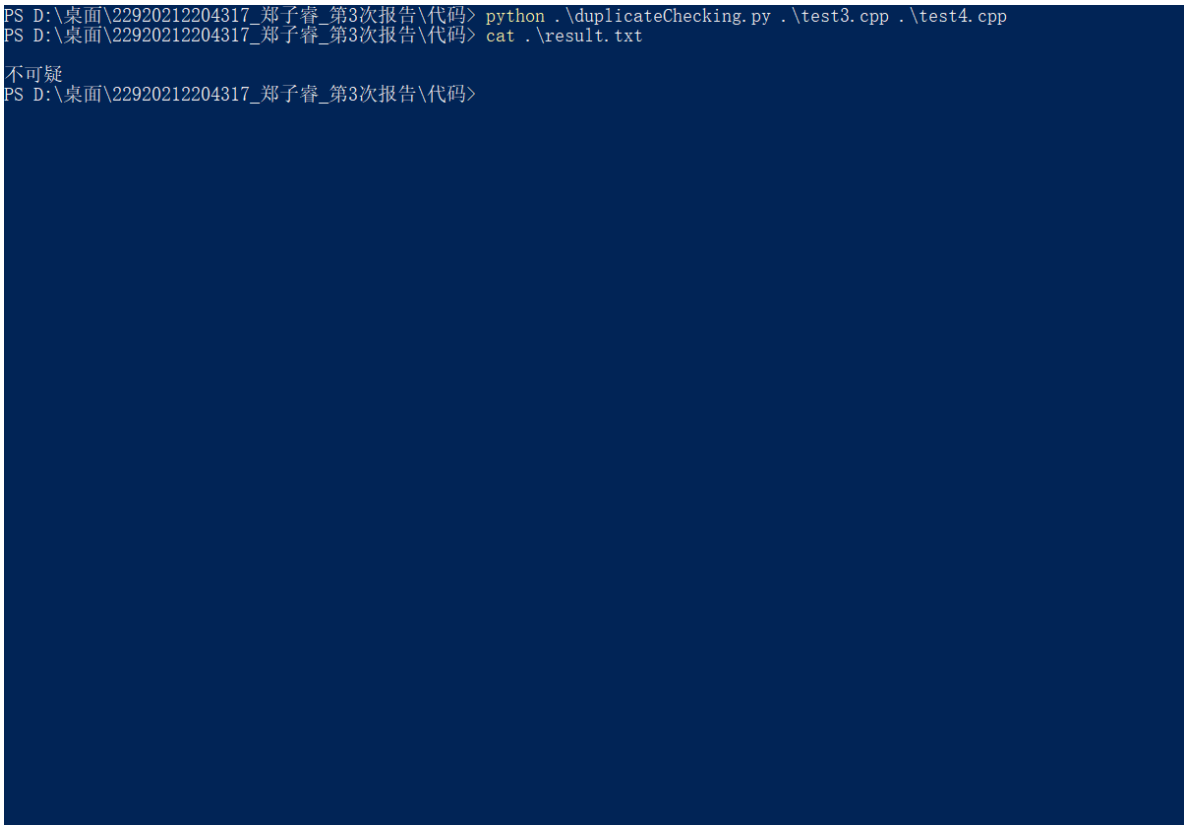
2. 筛查两份不同的代码

测试代码

```cpp
// test3.cpp 归并排序的递归实现
#include <bits/stdc++.h>
using namespace std;
const int N=100008;
int a[N],tmp[N];

void merge_sort(int a[], int l, int r){
    if(l>=r) return;
    int mid=l+r>>1;
    merge_sort(a,l,mid), merge_sort(a,mid+1,r);

    int k=0, i=l, j=mid+1;
    while(i<=mid&&j<=r){
        if(a[i]<=a[j]) tmp[k++]=a[i++];
        else tmp[k++]=a[j++];
    }

    while(i<=mid) tmp[k++]=a[i++];
    while(j<=r) tmp[k++]=a[j++];

    for(int i=l,j=0;i<=r;i++,j++){
        a[i]=tmp[j];
    }

}

int main(){
    int n;
    scanf("%d",&n);
    for(int i=0;i<n;i++){
        scanf("%d",&a[i]);
    }
    merge_sort(a,0,n-1);
```

```
    for(int i=0;i<n;i++){
        printf("%d " ,a[i]);
    }
    return 0;
}
// test4.cpp 归并排序的非递归实现
// 非递归写法二
#include <bits/stdc++.h>
using namespace std;

using ll = long long;
using pii = pair<int, int>;
const int N = 1e5 + 10;
int a[N], t[N];

void merge(int l, int r)
{
    if (l >= r) return;
    int mid = l + r >> 1;
    int i = l, j = mid + 1, k = 0;
    int len = r - l + 1;
    if (log2(len) != floor(log2(len)))
        mid = l - 1 + pow(2, floor(log2(len))), j = mid + 1;

    while (i <= mid && j <= r)
    {
        if (a[i] <= a[j]) t[k ++ ] = a[i ++ ];
        else t[k ++ ] = a[j ++ ];
    }
    while (i <= mid) t[k ++ ] = a[i ++ ];
    while (j <= r) t[k ++ ] = a[j ++ ];
    for (int i = l, j = 0; i <= r; i ++ , j ++ ) a[i] = t[j];
}

int main()
{
    ios::sync_with_stdio(false), cin.tie(nullptr), cout.tie(nullptr);

    int n;
    cin >> n;
    for (int i = 1; i <= n; i ++ ) cin >> a[i];
    for (int s = 1; ; s *= 2)
    {
        for (int i = 1; i <= n; i += s)
        {
            if (i + s - 1 > n)
            {
                merge(i, n);
                break;
            }
            else merge(i, i + s - 1);
        }
        if (s > n) break;
    }
    for (int i = 1; i <= n; i ++ ) cout << a[i] << ' ';
    return 0;
}
```

测试结果，重复率低于50%，不可疑



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。