姓名： 黄彬寓

学号： MF20330030

总述

该代码查重工作基于论文“Winnowing: Local Algorithms for Document Fingerprinting”的思想完成。具体实现步骤如下。

1 step1: 预处理

根据输入指令进行预处理，主要函数:prehandle。在这里，我们根据指令来进行判断，不合法的会报错输出”error: invalid input.” 或”error: non-existent filename.” 信息并直接退出。若合法，则进行下一步。

2 step2: 生成汇编源代码文件

通过 system 函数，生成两个 C++ 文件的汇编源代码文件 code1.s 和 code2.s，因为后续通过这两个文件进行比较，所以认为这两个文件是比较重要的，而区别于临时文件，在和 codesim 同目录下进行了保留。

3 step3: 生成文本字符串

根据 step2 得到的.s 文件生成文本字符串，主要函数:code_to_string。去除了空格、回行等几乎无影响字符后将文件内容保存在了内部类型为 string 的字符串中。

4 step4: 生成文本的哈希值序列

根据 step3 得到的文本字符串，在经过测试设置了 guarantee threshold t = 7, noise threshold k = 3,Karp-Rabin String Matching 哈希算法中的 base = 7 后，生成哈希值序列，主要函数 string_to_hashValue。

其中 hash 算法为： $H'(c_2 \dots c_{k+1}) = ((H'(c_1 \dots c_k) - c_1 * b^k) + c_{k+1}) * b$

5 step5: 使用 winnowing 算法对哈希值过滤得到文本指纹

根据 step4 得到的哈希值序列，对其使用 winnowing 进行过滤得到本文指纹，主要函数:filterByWINNOWING。具体算法采用的是单调递增的双端队列来实现。这个算法是这样实现的：

1. 我们每次将滑动窗口 W 后移一个单位，如果双端队列 q 的 front 恰为 W 丢掉的那一个数，那么它需要被 deque；2. 让这轮进来的那个数下标入队尾，但在此之前需要确保大于等于它的所有数字先从队尾被 deque，使得 q 满足安全性:q 始终为单调递增双端队列；3. 如果上一轮写入指纹 F 的数字下

标和 q 的队首相同，证明这轮 W 中的最小数仍是上一轮 W 的最小数，则不做操作；4. 如果上一轮写入指纹 F 的数字下标和 q 的队首不同 (必然是小于)，那么将 q 的队首下标对应的数字加入指纹 F 尾部，并更新上一轮写入指纹 F。如图：

```
deque<int> queNum; // one queue in which data are increasing monotonicly.
queNum.push_back(0);
int minPos = 0;
for (int i = 1; i < w; i++) {
    while (!queNum.empty() && hashtable[queNum.back()] >= hashtable[i])
        queNum.pop_back();
    queNum.push_back(i);
}
vector<int> results;
results.push_back(hashtable[queNum.front()]);
int lastStorePos = queNum.front();
for (int i = w; i < hashtable.size(); i++) {
    if (queNum.front() == i - w)
        queNum.pop_front();
    while (!queNum.empty() && hashtable[queNum.back()] >= hashtable[i])
        queNum.pop_back();
    queNum.push_back(i);
    if (lastStorePos == queNum.front())
        continue;
    results.push_back(hashtable[queNum.front()]);
    lastStorePos = queNum.front();
}
return results;
```

6 step6: 比较两个文本的指纹并生成相似度

根据 step5 得到的文本指纹进行相似度比较，主要函数：compSimilar1_EditDistance，compSimilar2_LinearizableCompare。这里是本工作的输出模块，所以首先会判断一下原先指令中是否有-v 或–verbose，若有则先输出路径和文件以及每个文件对应的指纹。

随后，我用了两个算法，算法 1：指纹的数据结构是 vector<int>，故分别排序，随后进行从头到尾的线性比较，统计相同指纹数量 S，S 除以两个文本指纹数量的平均值为相似度；

算法 2：不排序，直接采用基于动态规划的编辑距离算法，把距离值 D 看作差异值，故相似度为 1-距离值 D 除以两个文本指纹数量的平均值。

```
for (int i = 1; i <= win1.size(); i++) {
    for (int j = 1; j <= win2.size(); j++) {
        minDist[i][j] = min(minDist[i][j - 1], minDist[i - 1][j]) + 1;
        if (win1[i - 1] == win2[j - 1])
            minDist[i][j] = min(minDist[i - 1][j - 1], minDist[i][j]);
        else
            minDist[i][j] = min(minDist[i - 1][j - 1] + 1, minDist[i][j]);
    }
}
int diff = minDist[win1.size()][win2.size()];
return (double)1 - (((double)diff) * 2 / (win1.size() + win2.size()));
```

最后，相似度取这两个算法得到的相似度的平均值，并进行输出。