Team Project Report

我们组选择的任务是第二个任务：实现软件抄袭检测系统。

该系统有两个功能：文件抄袭检测以及工程抄袭检测。文件抄袭检测可以输入两个文件的路径，得出两个文件之间有抄袭现象的可能性。而工程抄袭检测是建立在文件抄袭检测的基础上，得出两个工程之间有抄袭现象的可能性。

整个系统的结构见 src/README.md。以下重点解释文件抄袭检测以及工程抄袭检测的实现原理。下面抄袭检测的原理建立在一个基本假设之上：抄袭者生成的程序和原来的程序完成的是同一个任务。

文件抄袭检测的重点在于将代码字符串压缩成 fingerprint 特征字符串，具体的实现在 strategies_file.cpp 中。这种压缩方法是将代码当中的具体成分进行替换。替换过程中删去了空白字符，而且将具体变量和函数名称替换成同样的字符，从而避免因变量替换而导致的检测失败;与此同时保留可以体现程序结构的操作符和关键字，从而得到一个对程序的结构概括化的字符串。这些操作在 PFile::make_fingerprint()中得以实现。

在对比两个文件时，就对相应的压缩后的特征字符串进行比较。比较的方法是随机抽样，在一个字符串中随机抽取长度为 L 的字符串，并在另一个字符串中进行匹配，得到一个匹配成功的频率。将这个频率开 L 次方作为存在抄袭现象的概率。于此同时考虑 A 匹配 B 和 B 匹配 A 的差异，如果 A 匹配 B 得到的结果与 B 匹配 A 得到的结果有差异，则乘上一个随差异增大而减小的修正值，最终得到表征两个文件存在抄袭现象可能性的 0 到 1 的数。以上的实现可在 string_compare 中找到。

工程抄袭检测是在文件抄袭检测的基础上，对两个工程的文件两两进行比较，然后以可能性为权值进行匹配，以最优匹配的答案作为工程抄袭的可能性的考量。具体实现见 detector_project.cpp。

以上是对两个检测系统具体实现的解释。下面是对检测系统的具体测试过程。

为测试文件检测，我们制造了以下几组测试样例：

第一组：完全不抄袭（实现同一个功能类）

第二组：抄袭一个函数的实现

第三组：抄袭全部

在第二组和第三组中分别添加三个分组：什么都不做，替换变量，更改注释。

测试结果如下：

	程序 1	程序 2	程序 3
不抄袭 1	17%	20%	23%
不抄袭 2	28%	25%	21%
不抄袭 3	13%	10%	20%
部分抄袭	63%	52%	70%
部分抄袭+替换变量	61%	55%	65%
部分抄袭+更改注释	55%	51%	62%
全部抄袭	100%	100%	100%
全部抄袭+替换变量	100%	100%	100%
全部抄袭+更改注释	93	95%	97%

为测试工程检测，我们制造了以下几组测试样例：

第一组：完全不抄袭

第二组：部分文件抄袭（有一部分文件进行完全抄袭）

第三组：合并部分文件（将独立的文件合并写在同一个文件中）

第四组：全部文件部分抄袭（全部文件进行部分的抄袭）

第五组：抄袭全部

测试结果如下：

	工程 1	工程 2	工程 3
不抄袭	10%	5%	13%
部分文件抄袭	78%	70%	75%
合并文件	95%	92%	93%
文件部分抄袭	83%	82%	85%
全部抄袭	100%	100%	100%

综上所述，此监测系统可以在误差范围内将抄袭的代码与非抄袭的代码区分开来。