崔子寒
MG20330012
zihancui@smail.nju.edu.cn

Assignment 1: Codesim
ISER, 20

2020 年 12 月 2 日

1 开发环境和运行方法

开发环境	
操作系统	Ubuntu 18.04.4 LTS (GNU/Linux 4.15.0-111-generic x86_64)
编译器	g++ (Ubuntu 7.5.0-3ubuntu1 18.04) 7.5.0
`llvm`	10.0.0
`clang`	clang version 10.0.0-4ubuntu1
编译运行	
编译方法	在项目目录下执行 `make` 指令进行编译
运行	./codesim <input file1> <input file2> [-v] [-h]

2 解析 Abstract Syntax Tree(AST)

2.1 `Clang` 介绍

`Clang` 是一个适用于 C,C++,Objective-C 以及 Objective-C++ 的编译器前端。通常和编译器后端 `LLVM` 一起使用。在程序的编译过程中，通常分为前端和后端两个阶段，前端又叫做分析阶段，在这一阶段会完成语法分析、语义分析和中间代码生成，这部分和机器无关。而编译器的后端则只需要前端输出的中间表示，结合具体的机器进行优化，生成可执行的目标程序。

`Clang` 提供了各种用于语法分析和语义分析的工具，但是直接使用这些工具是不合适的，我们需要在程序中使用这些功能。为此，`Clang` 提供了 `libclang` 库。`libclang` 是 `Clang` 的高层次的 C 语言编程接口，尽管在 Python 和 Go 等语言中都可以通过 Cython 或者 Cgo 来使用 libclang，但是还是原生的 C++ 更加稳定。因此我使用 C++ 来进行开发。

2.2 使用 `libclang` 解析 AST

AST 是代码编译过程中产生的一种代码中间表示，可以反映出语法的层次结构。在 AST 中，每一个节点都是一个语法符号，一个节点的到它的所有子节点对应一个语法的推导。因为在 AST 中，我们略去了变量、函数等在程序源代码中的名字，只关注程序的语法结构，所以对 AST 进行相似度检测可以检测出更换变量名称等抄袭方法。

`libclang` 提供了 `clang_visitChildren` 函数，可以帮助我们遍历源程序中对语法单元，并构建语法树。`clang_visitChildren` 要求我们提供一个 `visit` 函数，在遍历源程序对过程中，当遇到了一个语法单元，就会调用 `visit`。`visit` 的参数有当前的语法单元，当前语法单元的父节点，以及一个自定义的数据类型。我们可以通过在 `visit` 中将节点之间的父子关系储存到自定义的数据类型中，构建出 AST。构建出的 AST 可以用下面的这种数据结构表示。从根结点触发进行前序遍历，就可以得到一个 `vector<string>`，也就是一个字符串序列，每一个元素都是一个语法单元的名称。通过对两个字符串序列进行相似度检测，就可以判断出源代码中是否存在抄袭。

```
class ASTNode {
private:
    vector<ASTNode*> children;
    int depth;
}
```

3 Greedy String tilling 相似度算法

Greedy String tilling 相似度算法可以检测两个序列型数据的相似度,相比于最长公共子串,Greedy String tilling 可以更好的检测顺序更换的情况。Greedy String tilling 通过寻找序列中不重叠的若干个匹配长度之和的最大值来得到两个序列的相似度。

Algorithm 1 Greedy String tilling

Input: vector<string> v_1, v_2

Output: similarity of v_1, v_2

$tiles := \{\}$
repeat
  $matches := \{\}$
  maxMatchLength := MIN_MATCH_LENGHT
  for $i := 0; i < length(v_1); i++$ **do**
    for $j := 0; j < length(v_2); j++$ **do**
      $k := 0$
      while $v_1[i+k] = v_2[j+k] \wedge unmarked(v_1[i+k]) \wedge unmarked(v_2[j+k])$ **do**
        $k := k + 1$
      end while
      if $k =$maxMatchLength **then**
        $matches := matches \cup match(i, j, k)$
      else
        $matches := \{match(i, j, k)\}$
        maxMatchLength := k
      end if
    end for
  end for
  for $match(i, j, k)$ *in* $matches$ **do**
    $mark(v_1, i, i+k), mark(v_2, j, j+k)$
    $tiles := tiles \cup \{match(i, j, k)\}$
  end for
until maxMatchLength < MIN_MATCH_LENGHT
return $\frac{\sum_{tile \in tiles} length(tile)}{length(v_1)+length(v_2)}$

注意最后一步，需要求所有 tile 之和时，需要过滤掉有重叠的 tile，有重叠的 tile 只计算最长的。