

高级程序设计II 大作业实验报告

项目地址: [GitHub](#)

代码贡献

- 农钧翔 2019201407:
 - 实现迭代数据流算法 & 构建Flatten图 (Task2)
 - 生成关联规则及置信度 (Task3)
 - 库函数对接与格式化输出 (Task4)
- 于倬浩 2019201409:
 - 构建Transaction图 & 规范化指令 (Task2)
 - 实现Apriori算法生成项集 (Task3)
 - 图形化界面的实现 & 与动态链接库的输出对接 (Task5)
- 汪元森 2019201420:
 - 初始化数据流算法 (Task2)
 - 实现calcu_support计算项集的支持度 (Task3)
 - 缺陷检测算法的实现 (Task4)

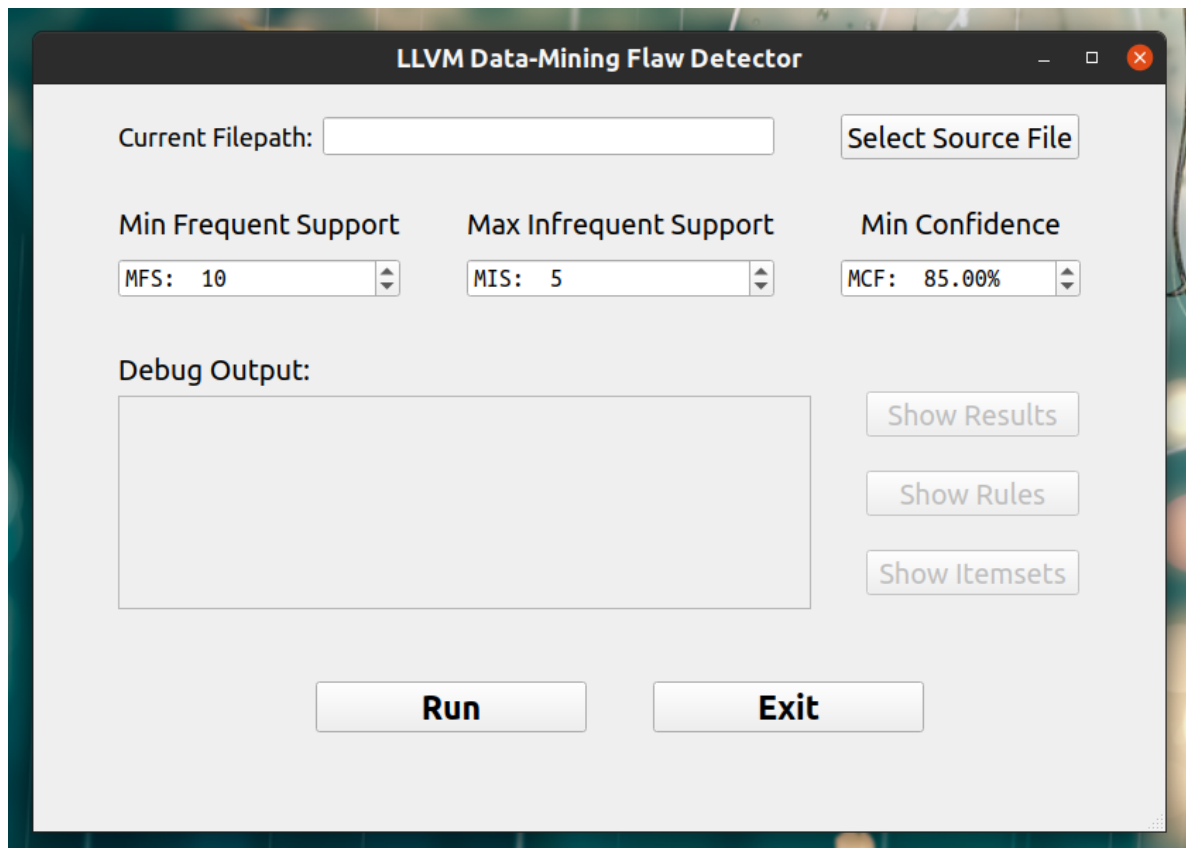
实验环境

操作系统: Ubuntu 20.04 LTS

```
1 $ clang --version
2 clang version 10.0.0
3 Target: x86_64-pc-linux-gnu
4 Thread model: posix
5 InstalledDir: /usr/bin
6 $ qmake --version
7 QMake version 3.1
8 Using Qt version 5.12.8 in /usr/lib/x86_64-linux-gnu
```

简单介绍

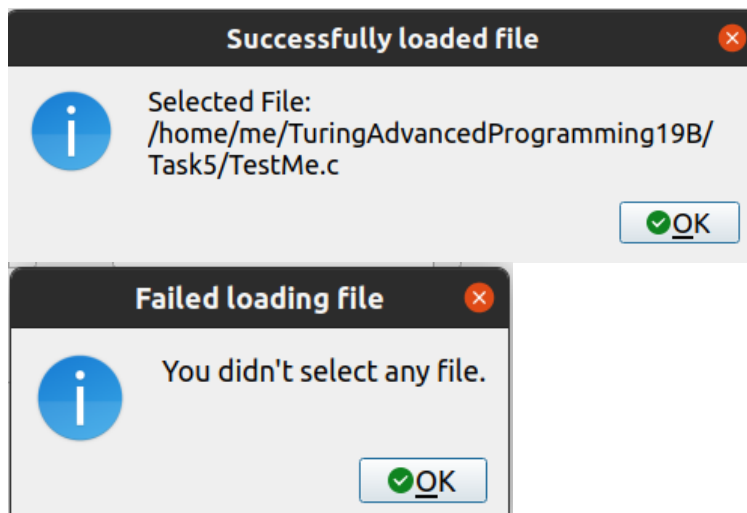
本次UI的实现采用了QT5，实现了一个简单的多窗口程序，主窗口初始状况下如下图所示：



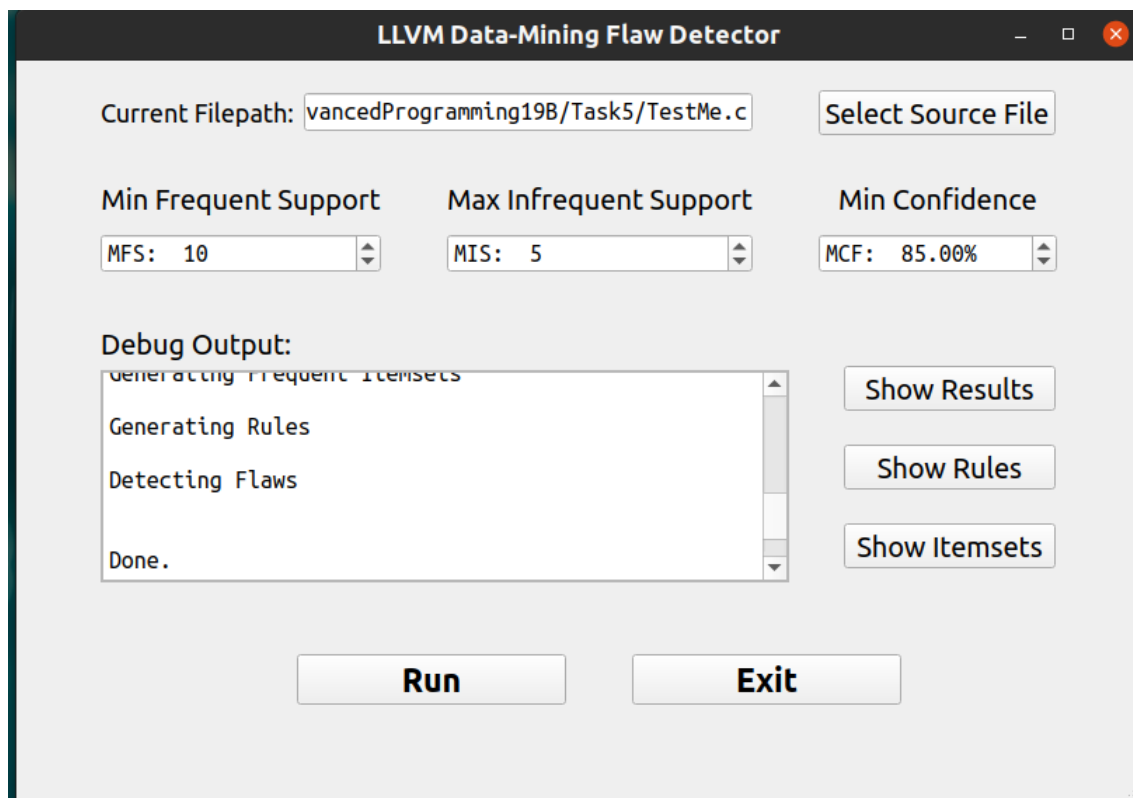
支持指定需要检测的源文件位置，指定MFS、MIS、MCF的值。

对于三个指定值的输入，选用了spinbox，以便控制输入值的上下界、步长等。

选择文件的PushButton通过调用QFileDialog，调用系统选择文件的窗口，较为美观，且限定用户只能选择C/C++源代码文件。文件选择结束后，调用QMessageBox给出反馈，同时将文件目录更新至主窗口的lineEdit中。



选定源文件，指定参数后，单击"Run"即可运行监测。运行过程中，所有设定参数的按钮均被禁用，运行结束后重新启用这些按钮。



动态链接库MyPass.so会将调试信息输出到stderr中，并实时更新至"Debug Output"中。对于结果和项集的传递，MyPass.so将这些信息输出至stdout，使用不同的token区分输出类型，MyPass在成功执行结束后，在stdout中返回一个token "\$\$SUCCESS"，UI在读取到这个token后，才会启用右侧的三个按钮。可以保证在编译错误/运行错误/MyPass运行结束前，阻止用户点击右侧的按钮。同时在文件更新后，亦会禁用展示结果的按钮。

在实现上，我使用了两个不同的槽函数，对接动态链接库的两个输出流，需要注意的是，由于输出流自带缓冲区，在每一次输出后必须清空缓冲区，否则会导致部分字符串丢失。

单击"Show ... "，会打开一个新窗口，展示项集数据/生成的规则/缺陷检测的结果。对于项集和规则，使用字符串列表的形式展示，每行一个规则或项集，使用等宽字体展示每条规则。对于缺陷检测结果，以只读纯文本的形式展示，灵活性更好。在实现上，我在MyPass.so的标准输出中，加入了不同的token，表示不同类

型的数据，槽函数读取时，便可较为容易的将输出的纯文本数据分类处理。下面同时展示了三个数据窗口的内容：

The screenshot displays the LLVM Data-Mining Flaw Detector interface, which is divided into several windows. The main window on the left contains configuration options for the analysis. The 'Current Filepath' is set to 'vancedProgramming19B/Task5/TestMe.c'. The 'Min Frequent Support' (MFS) is 10, 'Max Infrequent Support' (MIS) is 5, and 'Min Confidence' (MCF) is 85.66%. The 'Debug Output' window shows 'Generating Rules' and 'Detecting Flaws'. The 'Show Results' window displays two errors: Error #1, which is a missing 'free(i8*)' instruction, and Error #2, which is an unexpected instruction. The 'Show Itemsets' window shows frequent itemsets, including 'free(i8*)' and 'i8* = malloc(i32)', and infrequent itemsets, including 'free(i8*)' and 'free(i8*)'. The 'Show Rules' window displays positive association rules, including 'free(i8*)' -> 'i8* = malloc(i32)' and 'i8* = malloc(i32)' -> 'free(i8*)', and a negative association rule, 'free(i8*)' -> 'free(i8*)'.

LLVM Data-Mining Flaw Detector

Current Filepath: vancedProgramming19B/Task5/TestMe.c

Min Frequent Support: MFS: 10
Max Infrequent Support: MIS: 5
Min Confidence: MCF: 85.66%

Debug Output:
Generating Rules
Detecting Flaws

Show Results

Results

Error #1:
" %5 = call i8* @malloc(i32 %4), !dbg !19 ", TestMe.c:62
except an instruction but MISSING: free(i8*)

Error #2:
with the(se) following UNEXPECTED instruction(s):
" call void @free(i8* %12), !dbg !26 ", TestMe.c:93
" call void @free(i8* %9), !dbg !23 ", TestMe.c:92

Show Itemsets

Frequent Itemsets

"free(i8*)", "i8* = malloc(i32)", Support = 12

Infrequent Itemsets

"free(i8*)", "free(i8*)", Support = 1

Show Rules

Positive Association Rules

{ "free(i8*)" } -> { "i8* = malloc(i32)" } Confidence = 100.0%
{ "i8* = malloc(i32)" } -> { "free(i8*)" } Confidence = 92.3%

Negative Association Rule

Rule: { "free(i8*)" } -> { "free(i8*)" } Confidence = 91.7%