## The algorithm:

Solution1

The approach is to expand function and renew the HashMap by adding new callee function and deleting caller function. Then use the same solution of Part A to search the bug again.

Step1: Traverse the callgraph to store **set of callee functions for each caller function in HashMap ‘callerCallee’,** while storing set of caller functions for each paired or single function in HashMap ‘location’, which is also used in Part A that the size of caller functions set is just the support for each paired function or single function.

Step2: **Create HashMap ‘callerCalleeExtend’ to copy the HashMap ‘callerCallee’. Replace the set of callee functions with set of expanded callee functions, which is generated by recursion function called getCalleeExtend().**

Step3: **In recursion function, the return value is a list to store expanded callee functions. Besides, there is a parameter to control recursion level for recursion function itself, so that the recursion function will return the list of expanded callee functions for each caller function until the recursion function is stop at limited level.**

Step4: **Update HashMap ‘location’ according to HashMap ‘callerCallee’. Then print bugs as the solution of partA.**

Solution2

The approach is to traverse the bug output of part A to expand neighbor functions, which are called together with the bug in the same caller function. When expanding function, we will try to search the other element of pipair until the element is found.

Step1: Traverse the callgraph to store **set of callee functions for each caller function in HashMap ‘callerCallee’,** while storing set of caller functions for each paired or single function in HashMap ‘location’, which is also used in Part A that the size of caller functions set is just the support for each paired function or single function.

Step2: Traverse the HashMap ‘location’ to calculate support and confidence. Before printing the single function bug, its location and related pipair function, we use **DFS algorithm to search if there is a callee function in neighbor function can make up that pipair together with that single function.** If there is the callee function, ignore that bug. If no call function, print that bug.

Step3: In DFS algorithm, **HashMap ‘callerCallee’ can be regarded as a graph that each function is a node and a relation between caller and callee function is an edge. There is a parameter to control expansion level for HashMap ‘callerCallee’, so that we can search the callee function to make up that pipair until the callee function is found or HashMap ‘callerCallee’ has been traversed on limited expansion level.**

## The output:

Run the command as below. The fourth parameter after ‘pipair\_java’ is to control the expansion level. In this example, the fourth parameter is 1, which means to expand 1 level. We will use the same command line and replace the fourth parameter with 2 or 3 to test the output for more expansion level:



Table1 the number of bug line for testcase3 (support is 3 and confidence is 0.65)

|  |  |  |  |
| --- | --- | --- | --- |
|  | PartC solution1 | PartC solution2 | PartA not expand |
| Level=1 | 1630 | 174 | 205 |
| Level=2 | 5792 | 166 |
| Level=3 | 13556 | 163 |

Table2 the number of bug line for testcase3 (support is 10 and confidence is 0.8)

|  |  |  |  |
| --- | --- | --- | --- |
|  | PartC solution1 | PartC solution2 | PartA not expand |
| Level=1 | 155 | 30 | 34 |
| Level=2 | 724 | 30 |
| Level=3 | 2093 | 30 |

## The analysis

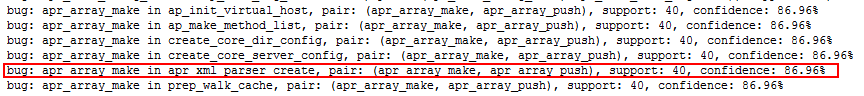
For solution1, the amount of bug increases with the raising expansion level. And the bug include original bug after reducing the false positive, and new bug after expanding the caller function. The reason to generate new bug is that the support and confidence of pipair increase, and there are more bugs meet the threshold for support and confidence.

Additionally, we meet two questions for this algorithm. The first one is whether to expand main function. As main function include all functions, any two of callee functions will turn to be paired if the expansion level is very high. However the pipair in main is also show some functions should appear together, for example, if there are lock and unlock functions in main, the support of this pipair will added one, and this support will influence the may belief that the lock and unlock functions should appear together. So we prefer to expand main function. The second one is whether to delete caller function after expanding that function. If we do not delete caller function, there may be more pipair including caller function, but useless to analyze false positive. And the number of bug increases rapidly with raising expansion level, we choose to delete caller function to reduce meaningless pipair.

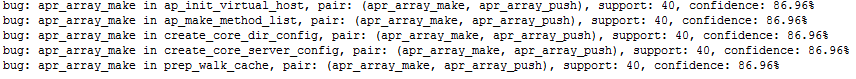
For solution2, the amount of bug in test3\_3\_65.out reduces with the raising expansion level, while that in test3\_10\_85.out reduce one false This algorithm is the improved algorithm that it can only reduce the false positive but not increase any other new bug, because it is only search the possible pipair hidden in inter-procedural relation, then reduce the bug in the output of PartA.

For both of two solutions, the time to generate bug output increases with the raising expansion level, because the more functions to expand, the more time it will take to store the relation of caller and callee function, or search additional inter-procedural relation.

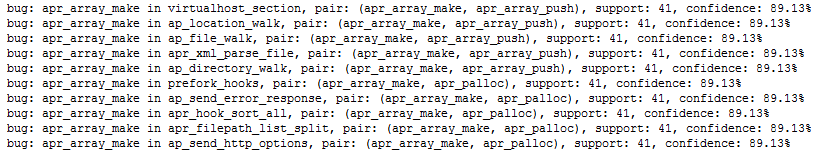
PartA:



PartC solution1



PartC solution2



In PartC output, the bug marked is false positive. Because In ‘apr\_xml\_parser\_create’, we can get pair: (apr\_array\_make, apr\_array\_push) after expanding ‘apr\_xml\_insert\_uri’ as below. So the solution for PartC reduce that false positive, but the second solution of PartC relatively add more new bugs because there will be more pipair after expansion.

