Program Analysis Verification and Testing

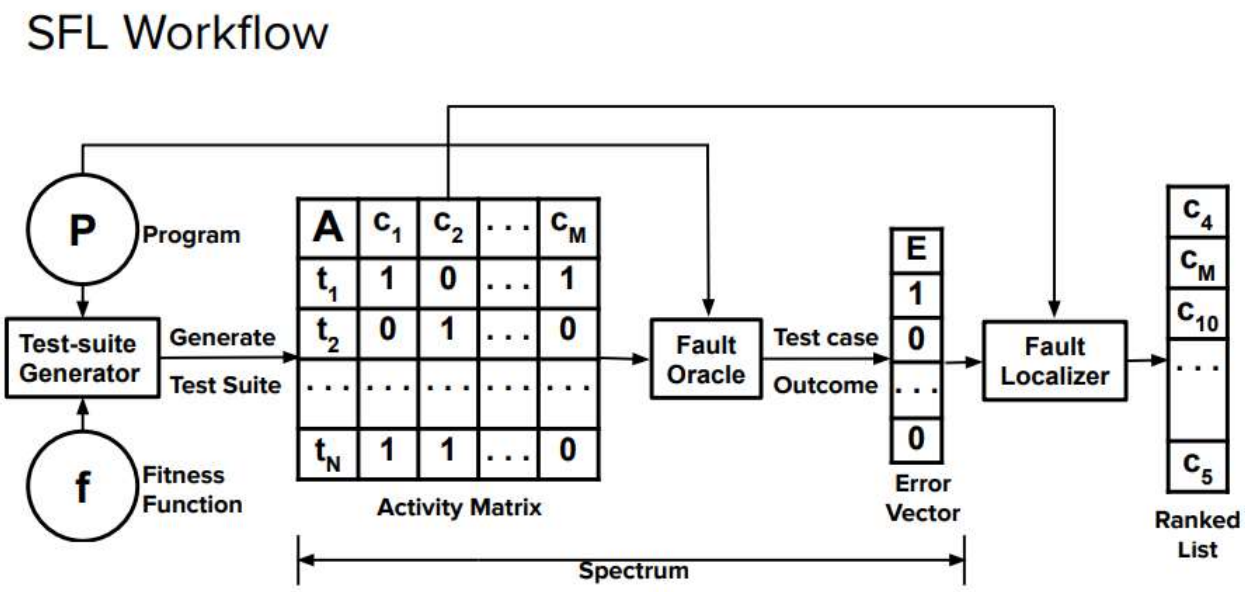
Assignment 3: Spectrum Based Fault Localization

Vaibhav Rahulkumar Sheth

Roll no. :- 231110054

IMPLEMENTATION:

This report describes about the functionalities of the program. This program takes data from two user written programs, one of which is correct while the other one is buggy. This program Implements the SBFL (Spectrum Based Fault Localization technique) to find probable bugs in the program.



The program has 3 major functionalities:-

1. **fitnessScore()** :
   This function takes as input an activity matrix that contains information about test cases and the components being executed in the case. This matrix consists of either 0 or 1 denoting whether that component was executed while running test case or not.
   I used **DDU (Density Diversity Uniqueness)** metric as a fitness function to denote whether the test case is appropriate or not.

DDU Metric

The quality of a test suite is quantified by:

$$DDU = \rho\prime . \delta . \mu$$

$$\rho\prime = 1 - |1 - 2\rho|$$

Density – It refers to the code coverage density or the percentage of covered code compared to the total code available. The density of an activity matrix can be calculated as the ratio of the number of covered elements.

Density

Density of a test suite:

$$\rho = \frac{\sum_{i,j} A_{ij}}{MXN}$$

Diversity – It refers to the variety or distinctiveness of the test cases represented by different rows in the matrix. It's a measure of how different or unique the test cases are in terms of their execution paths or coverage. I used gini-simpson metric taught in the class to find the diversity of the activity matrix. It is applied on test cases.

Gini-Simpson metric:

$$\delta = 1 - \frac{\sum_{k=1}^{l} |n_k|(|n_k|-1)}{N(N-1)}$$

Uniqueness – It refers to the set of distinguishable components present in the program.

Uniqueness is defined as:

$$\mu = \frac{|\Omega|}{M}$$

2. **Suspiciousness()** :

This function takes component number from the user and returns a suspicious number related to the component of the program. The suspiciousness of the component has been calculated using the Ochiai metric as given in “An evaluation of similarity coefficients for software fault localization.”

Ochiai Metric*

C_f	Number of failing tests that execute C
C_p	Number of passing tests that execute C
N_f	Number of failing tests that do not execute C
N_p	Number of passing tests that do not execute C

$$Ochiai(C) = \frac{C_f}{\sqrt{(C_f+N_f).(C_f+C_p)}}$$

3. **getRankList():**

Using the suspiciousness calculated in the above function, I used individual columns to find their suspicious score and then arranged such that it contained [component_number:suspicious_score]. In order to present the solution to the developer, the ranks need to maintained in the non decreasing, therefore I used python sorting function on suspicious values of components to arrange them. The non decreasing sorted rankList is then passed back to the program.
Ranks are given in non decreasing order such that if both components has same value then they are given same rank.

ASSUMPTIONS:

1. **Bugs in Components:** The program is expected to misbehave in cases of multiple bugs in various or same components. It is expected that the program has a single bug as of now for this sample program.
2. **Python Library Support:** The program uses some general libraries of python programming language, erefore the program is expected to have those libraries included in the program.
   I have used 2 libraries math and operator**.**

3. **Consistency:** It may assume that the activity matrix is consistent, meaning that there are no conflicting or contradictory test case representations.

LIMITATIONS:

1. **Bugs in Components:** The program is expected to misbehave in cases of multiple bugs in various or same components. It is expected that the program has a single bug as of now for this sample program.

2. **No Error Handling:** The method assumes that comp_index is a valid component index. If an invalid index is provided, the method might result in an IndexError. Adding error handling to check the validity of the index would be beneficial.

3. **Ranking Ties:** In case of ties in suspiciousness scores, the ranking does not account for this, and it assigns consecutive ranks without distinguishing ties. Depending on the application, a more sophisticated ranking strategy for tied scores might be needed.

4. **Assumption of Binary Values:** The method assumes that the components have binary values (0 or 1). If the components can have values other than 0 and 1, the calculation of suspiciousness may need to be adapted.