Test Selection and Prioritization with Dynamic Programming

Problem Description – Part 1

Now, you will design an algorithm in order to select a subset of test suites. Test suites contain a collection of related test cases that are used to test a functionality of the software. Each suite requires a different amount of time to complete its operation, in seconds. Since you already used those tests for testing the prior releases, you already measured the run time of each test cases and saved the information into your database. Also, during the previous release's test phase, you kept the information of the discovered bugs by a test suite. So, you have a dataset similar to below.

	# discovered bugs	running time
Test suite 1	24	12
Test suite 2	13	7
….		

You don't want to miss the potential bugs in the new release. You prioritize testing the modules that were previously buggy. You aim to test the new release of software by running as many test suites as by considering the time constraint due to the project plan. You should develop a dynamic programming solution that selects a subset of test cases by considering the two criteria; 1) the number of previously discovered bugs by a test suite, and 2) the running time of a test suite.

Before the test phase of the project starts, you want to be sure your test suite selection solution will work or not, by utilizing a dummy dataset. The dummy data is given in data.txt file to test your dynamic programming solution. In the given .txt file, there are test suites and their corresponding information (number of discovered bugs and running times), and time capacity. Time capacity represents the total available time for the regression test.

1)Code: Please code a dynamic programming solution for the test suite selection problem described above. The output of your program should contain the optimum
the complexity set of selected test suites and the number of previously discovered bugs by running those test suites. Please provide comments on your algorithm if it is needed.

2)Report: Write the mathematical representation of the optimization function you used in your code. Find of your algorithm and discuss in your report.

3)Report: Does your algorithm work if the running times of the test suites are given as real numbers instead of discrete values? If not, please provide a solution in your report as pseudo code and discuss why it works and the previous solution does not work.

Problem Description – Part 2

In the previous part of the problem, you aim to decrease the cost of testing in terms of time by considering the quality of the software product (keeping the bug detection rate as high as possible) will be released. In addition, you aim to discover the bugs in the software as early as possible while testing, because the earlier the bugs are discovered, the more available time the team has to fix them.

Each test suite contains a set of test cases. A test case includes a specification of inputs, conditions, etc. passed to the function to be tested. Once a test suite is run, each of the test cases included in the suite will run in some order and test the software by giving various inputs. A test case will pass if the tested functionality of the software responds correctly, which means the functionality is developed as designed. A test case will fail if the tested functionality of the software does not respond correctly, which means the functionality is not developed as designed and it contains bug.

As the second part of your test phase optimization solution, you decided to prioritize the execution of test cases inside the test suites by ordering them according to their potential to discover bugs. Code statement coverage is a widely used way of test case prioritization. If a code statement is executed while performing a test case, that means the code statement is covered by the test case. Covering as many as possible code statements during the execution of a test case increased the possibility of discovering bugs. Therefore, firstly you select the test case that covers the code statements the most among all test cases, and order the remaining test cases according to the pairwise similarities between the remaining ones and the selected test case.

You will design an algorithm that uses the dynamic programming technique to find the similarities between the code statements over their frequency profiles. Each test case has a frequency profile that represents the execution frequency of each code statement during a test case run. An example is given below to describe the problem better.

Statement id	**Function to be tested**	**Test case 1**	**Test case 2**	**Test case 3**	**Test case 4**
	calculate(int a) {	a = 0	a = 1	a = 2	a = 3
1	int b = 0;	1	1	1	1
2	while(a > 1){	1	1	2	3
3	if(a%2 == 0)	0	0	1	2
4	b += 1;	0	0	1	0
5	a -= 2;} / ∗ a− = 1 ∗ /	0	0	1	2
6	System.out.println(b); }	1	1	1	1
	Test result	**pass**	**pass**	**pass**	**fail**

The table represents a test suite for a function that calculates the quotient of a parameter divided by 2. Code statements of the function are given in the second column. The function contains a bug in the statement numbered as 5. There are a total number of four test cases that test the function by assigning various values to the variable *a*. Assigned values to variable *a* is {0, 1, 2, 3} and the expected responds of the test cases are {0, 0, 1, 1}. However, the function in the table will respond as {0, 0, 1, 0} and that means while the first three test cases passed the last one is failed because of the bug in the fifth statement. The frequency profile of a test case is given in the column represented by the test case. Frequencies show the number of executions corresponds to each statement.

In the given example the execution order of the test cases is test case 1, test case 2, test case 3, and test case 4. The bug in the program is discovered by the execution of the test case 4. However, by re-ordering test cases, you will make the 4th test case run earlier than others.

Your goal is to re-order the test cases by their significances in order to set a new execution order for the test cases based on the ordered sequence of statement execution frequencies. The ordered sequence of statement execution frequencies is an order of the statements of the tested program from the least executed statement to the most executed statement. An example is given below.

Current order of the test cases: 1, 2, 3, 4

Frequency profiles of each test case: 1-1-0-0-0-1, 1-1-0-0-0-1, 1-2-1-1-1-1, 1-3-2-0-2-1

Number of covered (executed at least once) statements by each test case: 3, 3, 6, 5

Ordered sequence of statement execution frequencies for each test case: 3-4-5-1-2-6, 3-4-5-1-2-6, 1-3-4-5-6-2, 4-1-6-3-5-2

The test case (3 in the example) that has the highest coverage (it executes all the statements in the program, and it is our starting point for catching the potential bugs) will be accepted as the most significant one and will become the first element of the new ordered cases. Later, you will calculate the similarities between the test case, which has the most coverage, and the others by using an edit distance measure. In the end, you will order the remaining test cases in order to maximize the diversity (least similar to most similar). The aim of maximizing diversity is to increase the bug detection rate.

Distance between the test case 3 and 1: difference(1-3-4-5-6-2, 3-4-5-1-2-6) = 3

Distance between the test case 3 and 2: difference(1-3-4-5-6-2, 3-4-5-1-2-6) = 3

Distance between the test case 3 and 4: difference(1-3-4-5-6-2, 4-1-6-3-5-2) = 4

The final order of the test cases: 3, 4, 1, 2 or 3, 4, 2, 1

The second part of your algorithm is expected to run integrated with the first part. In other words, firstly, your algorithm selects a set of test suites, secondly, orders the execution of test cases inside of selected test suites.

Code: Please code a dynamic programming solution for the test case prioritization problem described above. Your version of the edit distance algorithm may calculate the differences between the given pairs differently than the sample given in the problem description. The output of your program should contain the ordered test cases for each selected suite in addition to the outputs of the first part (the optimum set of selected test suites, and the number of previously discovered bugs by running those test suites).

Report: Describe the details regarding your algorithm design, such as which edit distance algorithm you used, what are the operation costs you set? Write the mathematical representation of the optimization function you used in your code. Find the complexity of your algorithm and discuss it in your report.