**ISE328 – Algorithms and Optimization Methods**

**2023-2024 Spring Semester**

**Homework #2 Report**

**Solving Test Case Selection Problem With Greedy Approach**

As of my greedy strategy, I have implemented an efficiency point approach. When getting the input from the user, my code directly writes the cost of the test case and frequency of lines covered for that specific test case into a dictionary. With this way, when I need the cost of the test case or line frequency of the test case, I could directly reach them with only key values like “T1, T5 etc.”. When calculating the test case’s efficiency point, I use the fallowing formula:

0.5\*costOfTheTestCase (TN)-0.5\*frequencyOfLines (TN). According to the testcase point calculation formula, getting less points mean you have less cost and high frequency of lines. So, I have sorted the efficiency point dictionary in ascending order, which means if I want to reach most efficient test case, I should look at the start of the dictionary. I haven’t normalized the values of each part since I found that the algorithm worked better without normalization of the values.

So, in my greedy approach, I start traversing lines that are not yet checked by test case, and for each line I look at my dictionary from start to end (to find most efficient test case), and for each of them I check whether line is covered by that efficient test case. If not, I traverse through the efficient test case dictionary till I find the most efficient test case to cover that line. After finding the most efficient test case for the specific line, my algorithm starts doing everything in the same way with the fallowing “lines to be tested”. So, my greedy algorithm decides the most efficient test case for each line.

**Line By Line Code Analysis:**

**metin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu**

In the first 2 lines, I get the data from the user about how many lines of the module have and how many test cases that we have. And store the data into numOfLines and numOfTestCases variables.

frequencyOfLinesDict and costOfTestCasesDict are dictionaries where I store the how many lines frequency each test case has and what is the cost for each test case.

In the loop, I get the details about each test cases and what lines do they cover. Moreover, I fill the dictionaries that I have mentioned above. In the testCaseDetailArray, I store the covered lines by each test cases. For example if we want to reach lines covered by the T1 test case, I use testCaseDetailArray[0].

ekran görüntüsü, metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

In the above code snippet, I create efficiencyPointDict which stores efficiency point for each of the test cases. I use the formula that I have mentioned in the first page in order to calculate the efficiency point. In the last line of the code, I sort the dictionary based on the values (efficiency point) in the ascending order. efficiencyPointDict is essential part of my greedy algorithm.

metin, ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu

In the above code snippet, linesToBeTested stores the lines that are not covered by the any test case that we have chosen. finalTestCasesDetermined stores the chosen test cases in the end (final solution). Initially, we fill lines to be tested with all of the lines available and we fill the testCasesAvailable with all of the test cases.

In the while loop part, we keep doing the same thing until linesToBeTested array is empty. We iterate lineToBeTested array in order to cover all the lines. After choosing the line, we try to find most efficient testcase by iterating throught the efficiencyPointDict. If the chosen line is covered by the test case and if we haven’t use the test case, we determined that we want that test case. So, we put that specific test case into the finalTestcasesDetermined, and we remove that test case from the available test cases. In the end for loop, we delete all the lines covered by the test case from the linesToBeTested array. After doing everything necessary to our found Test case, we break in order to find the most efficient test cases for the rest of the lines(With breaking, we start from the left most efficientDictionary). So, this part finds the most efficient test cases for each of the lines.

metin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

Sample output of my greedy algorithm.

siyah, karanlık içeren bir resim

Açıklama otomatik olarak oluşturuldu