Department of Computer Engineering

BLG 336E
Analysis of Algorithm II
Homework 3
Report

ID	*Name*	*Surname*
150140126	*Emre*	*Reyhanlıoğlu*

PART 1

In the first part of the homework, I used 0/1 Knapsack Algorithm by using dynamic programming approach.

1.2) Mathematical Expression of the Optimization Function

This Knapsack algorithm has space and time complexity which is “O(n * r)” (n is the number of test suites, r is the max allowed running time).

Algorithm fills the two dimensional array with number of bugs values to maximize the total number of detected bugs in a limited time. Also it has a optimization function to set these array values;

array[i, b] = max{ array[i-1, b] , array[i-1, b - b[i]]+ t[i] }

array → stores total number of bugs , i → index number, b → array of number of bugs,

t-> needed time array for the test suites

You can see the generated array by using the given test data(data.txt file) below;

```
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0  0  24 24 24 24 24 24 24 24 24 24 24 24 24 24 24
0  0  0  0  0  0  0  13 13 13 13 13 24 24 24 24 24 24 24 37 37 37 37 37 37 37 37
0  0  0  0  0  0  0  13 13 13 13 23 24 24 24 24 24 24 36 37 37 37 37 47 47 47 47
0  0  0  0  0  0  0  13 15 15 15 23 24 24 24 28 28 28 36 38 39 39 39 47 47 47 51
0  0  0  0  0  0  0  13 15 16 16 23 24 24 24 28 29 31 36 38 39 40 40 47 47 47 51
```

Figure 1: Generated array values

Also this video lecture on the internet helped me a lot while doing this homework. (link of the video lecture: https://www.youtube.com/watch?v=nLmhmB6NzcM)

1.3) Running with Real Number Inputs

Yes, my algorithm can work with real numbers. Steps to be done are below;

- ReadFile class parses test cases' frequency values as integers with *“std::stoi(str)”* function. I need to use a different method to parse it as a float.
- Then, I should edit the TestCase class which has a variable called *“frequencyList“* which holds an integer list. It should be float list instead of integer list.
- Finally, other necessary variables, arrays and methods used in this Knapsack algorithm should has float type instead of integer.

PART 2

In the second part of the homework, I used Levenshtein's distance algorithm to calculate distance between two test cases by using dynamic programming approach. It uses look-up table to memorize internal values to run faster than exponential solutions by using trees. However, it has a space and time complexity which is *"O(n*n)"* (n is the length of the test case's statement id array).

Code itself with comments shows the mathematical expression and we can easily understand how it works. Please look at this distance calculator method below;

```
// HELPER METHOD FOR PART 2
int DistanceCalculator::calculateDistanceBetweenTests(TestCase* testCase1, TestCase* testCase2){

    int size1 = testCase1->frequencyLength;
    int size2 = testCase2->frequencyLength;

    // Result array
    int resultArray[size1+1][size2+1];

    // Initializing the result array
    for (int i = 0; i <size1 + 1; i++) {
        for (int j = 0; j <size2 + 1; j++) {
            // When first test case statement ids is empty, minimum number of operations is j
            if (i == 0){
                resultArray[i][j] = j;
            }

            // When second test case statement ids is empty, minimum number of operations is i
            else if (j == 0){
                resultArray[i][j] = i;
            }

            // When last statement ids are same, we can ignore it and continue with the remaining statement ids
            else if (testCase1->orderedIdList[i-1] == testCase2->orderedIdList[j-1])
                resultArray[i][j] = resultArray[i-1][j-1];

            // If the last statement ids are different, we should select the minimum cost of 3 operations and increment it by 1
            else{
                resultArray[i][j] = 1 + min(min(resultArray[i][j - 1], // Insert operation
                                                resultArray[i - 1][j]),        // Remove operation
                                                resultArray[i - 1][j - 1]);    // Replace operation
            }
        }
    }

    return resultArray[size1][size2];
}
```

Figure 2: Distance calculator method