BLG 336E Analysis of Algorithms Homework 3

Talha Çomak 150160726

I used -std=c++11 compiler options in my homework.

```
int knapsack(int n, int W, vector <int> wei, vector <int> val) {
    int** M = new int* [n + 1];
    v_trick.resize(n + 1);
    for (int k = 0; k < n + 1; k++) { // assign zero to array M
        M[k] = new int[W + 1];
        v_trick[k].resize(W + 1);
        for (int i = 0; i < W + 1; i++) {
            M[k][i] = 0;
        }
    }
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= W; j++) {
            if (wei[i-1] > j) {
                M[i][j] = M[i - 1][j];
                for (int k = 0; k < (int)v_trick[i - 1][j].size(); k++) {   // assign the
                    v_trick[i][j].push_back(v_trick[i - 1][j][k]);
                }
            }
            else {
                if (M[i - 1][j] < M[i - 1][j - wei[i-1]] + val[i-1]) {
                    M[i][j] = M[i - 1][j - wei[i-1]] + val[i-1];
                    for (int k = 0; k < (int)v_trick[i - 1][j - wei[i-1]].size(); k++) {
                        v_trick[i][j].push_back(v_trick[i - 1][j - wei[i-1]][k]);
                    }
                    v_trick[i][j].push_back(i-1);   // storing the index of the value
                }
                else {
                    M[i][j] = M[i - 1][j];
                    for (int k = 0; k < (int)v_trick[i - 1][j].size(); k++) {   // assign
                        v_trick[i][j].push_back(v_trick[i - 1][j][k]);
                    }
                }

            }
        }
    }
    return M[n][W];     // max value == max bugs
}
```

This is the optimization function i used in my code. The mathematical represantation of it

$$Maximum \sum_{i=1}^{n} v_i * x_i$$

$$s.t. \sum_{i=1}^{n} w_i * x_i \leq W \quad x_i \in \{0,1\}$$

The time complexity of knapsack algorithm is O(n*W) but in my function i added extra loops in order to store the used suites. So in my code the time complexity is a little higher than it. The size of the extra loops are number of the used suites. If the last size is S, the new time complexity may be O(n*W*S/2).

My algorithm doesn't work with real numbers instead of discrete ones. I used weight array variables as index of the matrix. So we can not put real numbers into them. As an alternative solution we can use the recursion version. Pseudocode is below

```
knapSack(n, W, wt, val)
  if n == 0 or W == 0
    then return 0;
  if  wt[n - 1] > W
    return knapSack(n – 1, W, wt, val);
  else
       then return max(val[n - 1] + knapSack(n – 1, W - wt[n - 1], wt, val),
                knapSack(n – 1, W, wt, val));
```

Part 2:

In that part i create a loop in selected Suites in order to calculate distances of testcases of them. Firstly i found the highest coverage cases and calculated the index arrays of the frequencies of the testcases. Then i found the distances via *editDistance* function. I assigned INT_MAX to distances of the highest coverage testcases. I stored that values in the *distance* vector array. Then i sorted them via bubble sort in descendent order. And finally i printed out them. The ouput of program with data.txt file is below.

Total amount of bugs: 51
Total amount of running time:26
TS2 1
TS3 5 6 11 12 2 4 10 1 3 8 9 7
TS4 3 4 1 2

In the *editDistance* algorithm (levenshtein distance) each operation has unit cost.

The time complexity of that algorithm is O(x*y) which is equal to multiplication of the size of the compared arrays.

The Mathematical represantion of that algorithm is

$$\operatorname{lev}_{a,b}(i,j) = \begin{cases} \max(i,j) & \text{if } \min(i,j) = 0, \\ \min \begin{cases} \operatorname{lev}_{a,b}(i-1,j) + 1 \\ \operatorname{lev}_{a,b}(i,j-1) + 1 \\ \operatorname{lev}_{a,b}(i-1,j-1) + 1_{(a_i \neq b_j)} \end{cases} & \text{otherwise.} \end{cases}$$