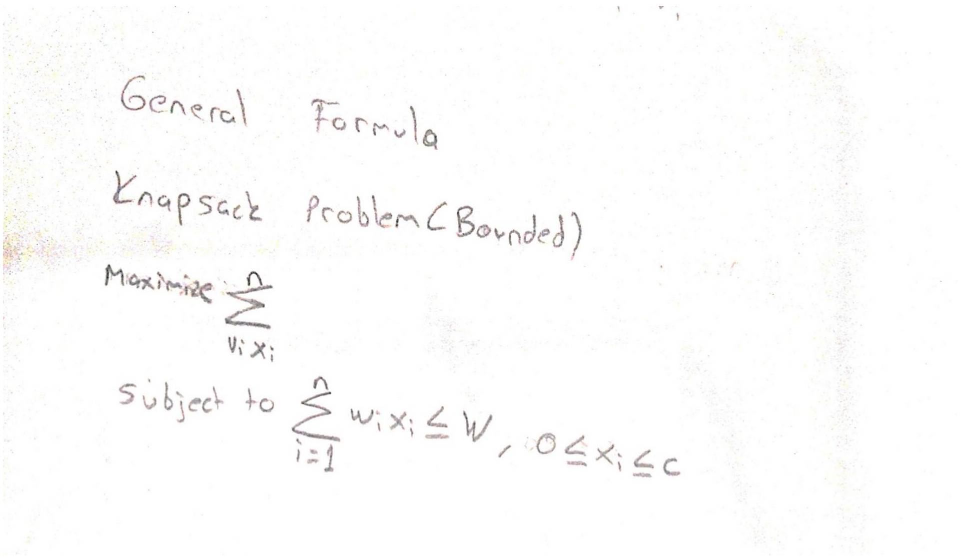
Ahmet Zafer SAĞLIK

150160519

1. First Question

First question was a variant of knapsack problem, total running time was the capacity of the sack, amount of bugs were the item values and the running times were the weights. To solve the problem a dynamic programming solution is implemented using the table method where I go increase the capacity iteratively and change or add items if necessary, as described in the lecture slides. When I run the program the output for the first question is given below:



General Formula

Knapsack Problem(Bounded)

Maximize $\sum\limits^{n}_{v_i x_i}$

Subject to $\sum\limits_{i=1}^{n} w_i x_i \le W$, $0 \le x_i \le c$

The *Mathematical representation of the algorithm is given below:*

Maximize sumof(item_values_in_the_sack)
s. t. sumof(item_weights_in_the_sack) <= sack_capacity

Of course I don’t solve this optimization problem with operations research approach thus with the dynamic programming approach our algorith complecity is O(N * W) where N is the number of items and W is the maximum capacity of the sack.

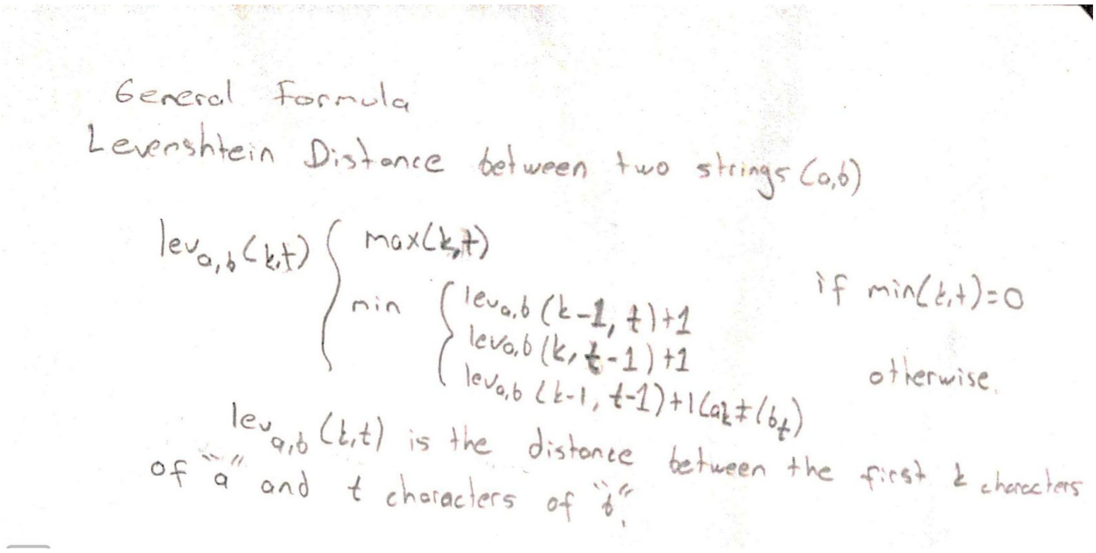
Output

```
Selected cases are: TS2 TS3 TS4 with a total of 51 bugs.
Elapsed time: 5e-06
```

Since this is knapsack problem it works with integer values, yet a solution can be proposed for real values. Here of course I will not be getting an exact optimum solution but a pseudo-optimum. Let us say the running times are rounded to the second decimal place for example if a running time is 4.123123 I will take it as 4.12, then I can take 0.01 as our discrete smallest weight, thus evething will be converted to 0.01 units, meaning I will divide each weight value and our capacity to 0.01 then round it. After doing this I can run the same knapsack algorithm and find a close enough solution.

2. **Second Question**

Second question is an edit distance question which is again a common dynamic programming problem.I used Levenshtein Distance algorithm. In the implementation distances between spesific locations on the array is kept in memory so while I continue calculating I don’t constantly calculate the value for already calculated operations. I am basically countring the number of remove and adds to make the two given arrays the same. The output of the program for the second question is given below:



General Formula

Levenshtein Distance between two strings (a,b)

$$\mathrm{lev}_{a,b}(k,t) \begin{cases} \max(k,t) & \text{if } \min(k,t)=0 \\ \min \begin{cases} \mathrm{lev}_{a,b}(k-1,t)+1 \\ \mathrm{lev}_{a,b}(k,t-1)+1 \\ \mathrm{lev}_{a,b}(k-1,t-1)+1(a_k \neq (b_t) \end{cases} & \text{otherwise.} \end{cases}$$

$\mathrm{lev}_{a,b}(k,t)$ is the distance between the first k characters of "a" and t characters of "b".

<u>The mathematical representation is given below:</u>

Minimize num_of_inserts(a) + num_of_deletions(b)

s.t. inserted(a) == deleted(b)

and the comlexity is O(nxm) n and m are being the lenght of the arrays.

Output

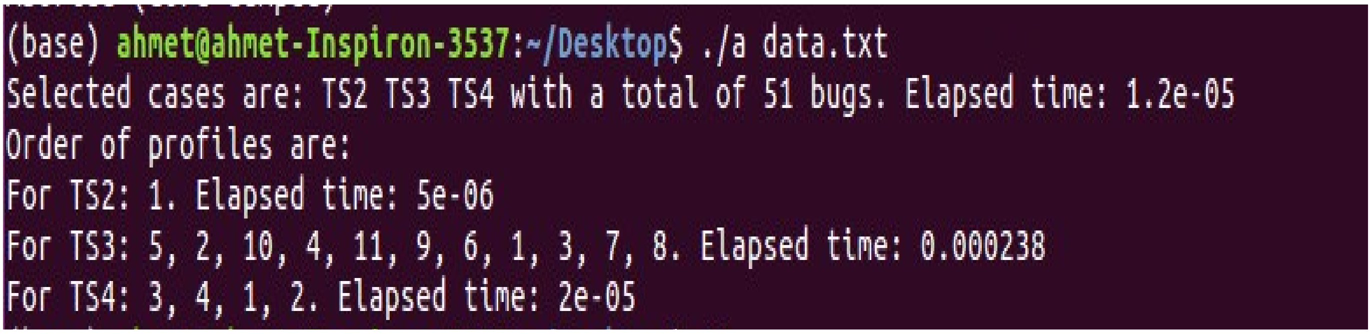
```
Order of profiles are:

For TS2: 1. Elapsed time: 2e-06

For TS3: 5, 2, 10, 4, 11, 9, 6, 1, 3, 7, 8. Elapsed time:
0.000276

For TS4: 3, 4, 1, 2. Elapsed time: 1.2e-05
```

Total Output



```
(base) ahmet@ahmet-Inspiron-3537:~/Desktop$ ./a data.txt
Selected cases are: TS2 TS3 TS4 with a total of 51 bugs. Elapsed time: 1.2e-05
Order of profiles are:
For TS2: 1. Elapsed time: 5e-06
For TS3: 5, 2, 10, 4, 11, 9, 6, 1, 3, 7, 8. Elapsed time: 0.000238
For TS4: 3, 4, 1, 2. Elapsed time: 2e-05
```

Importent Note!

In the text file (input file) Sometimes TAB sometimes Space has been used.In order to get rid of any problem ı have chanced all tabs into space.

It is working both texts but please used the one I send in the folder