DNA

For most of us biology is not on the list of the most interesting subjects. But not Ivancho – he is a maniac on theme genetics. And now he wants to put his theoretical knowledge in use and create an army of mutants. But to do that, he will have to play around with the deoxyribonucleic acid, also known as DNA. Ivancho knows how the mutants should look like, so he knows what the final form of their DNA should be. However, to make it real he will have to perform a number of operations on an already existing DNA molecule.

We can look on the initial (the existent) DNA and the one, desired by Ivancho as sequences of lowercase latin characters (strings). Ivancho wants to find a sequence as close to the desired one as possible, using the following operations on the initial (current) sequence:

1) Move a part of the sequence from one place to another
2) Reverse the order of the letters in a part of the sequence
3) Remove a letter from the sequence
4) Insert a letter in the sequence
5) Ivancho also has a dictionary with smaller parts of DNA, which he can use to overlap part of the sequence, replacing its contents

For a mutant to live it is important that the sequence you build has the same length as the desired sequence. Ivancho also has a matrix which he uses to define the “genetic difference” between two letters, that determines how close he is to the desired DNA.

As you all know this will take Ivancho too much time to do on his own. In addition he wants to have the mutants as soon as possible (who doesn't want to rule over the world anyways), so he is turning towards you, good programmers, to help him create his army. Write a program **fixdna**, which by a given starting and desired DNA molecule, does a number of operations so that the final sequence is as close as possible to the desired one.

Input: The first row of the input file **fixdna.in** will contain tree integers **N**, **K** and **T**, where **N** is the length of the starting and final sequence, **K** is the size of the alphabet, composed of the first **K** lowercase latin letter and **T** is the limit of operations you can do on the current sequence.
The following two rows will contain two strings – **startSequence** and **targetSequence**, respectively the starting DNA sequence (the sequence we can modify) and the desired one.

It is true that:
The length of **startSequence** = the length of **targetSequence** = **N**;
startSequence and **targetSequence** contain only characters from the alphabet (the first **K** lowercase latin letters)

The following **K** rows contain **K** integers each – this is the **charDifference** matrix which is used to define the difference between the letters, where:
charDifference[i][i] = 0, **charDifference**[i][j] = **charDifference**[j][i] for each i ≠ j
The difference between characters **c1** and **c2** equals **charDifference**[**c1** - 'a'][**c2** - 'a']

The following row contains a single integer **V** - the size of the dictionary (number of contained words)
The following **V** rows contain one string each $\mathbf{word}_i$ - the i-th word in the dictionary, containing only characters from the alphabet.

Output: The output file **fixdna.out** should contain one row with the integer **C** - the number of operations the program will do. 0 <= **C** <= **T**.
The following **C** rows should contain the parameters of the command that is to be executed. The first parameter should be the command code (from 1 to 5). Operations should be in the order they are to be executed.

Remark: indexing starts from 0.

Operations:

1) Move Substring

Parameters: **commandCode** = 1, **left**, **right**, **lettersOnTheLeft**

Action: We take the substring **currentSequence**[**left**...**right**] and we delete it from **currentSequence**. After that, we add it between positions **lettersOnTheLeft - 1** and **lettersOnTheLeft** (we insert it in a way that we keep **lettersOnTheLeft** characters before the substring).

Conditions: 0 <= **left** <= **right** < |**currentSequence**|.

0 <= **lettersOnTheLeft** <= |**currentSequence**| (position is calculated after removing the chosen substring)

2) Reverse Substring

Parameters: **commandCode = 2, left, right**

Action: Reverse all elements in the substring **currentSequence**[**left** ... **right**]. The substring keeps its position.

Conditions: 0 <= **left** <= **right** < |**currentSequence**|

3) Delete Character

Parameters: **commandCode = 3, index**

Action: Deletes the symbol at position **index**.

Condition: 0 <= **index** < |**currentSequence**|

Operation is allowed only if **currentSequence** is not empty

4) Insert Character

Parameters: **commandCode = 4, index, letter**

Action: All symbols in interval [**index**, |**currentSequence**| - 1] are moved with one position to the right, after which the symbol **letter** is emplaced at position **index**.

Condition: 0 <= **index** <= |**currentSequence**|.

letter is a character from the alphabet defined by **K**.

Current string is allowed to be longer than **N**.

5) Put Word

Parameters: **commandCode = 5, wordIndex, positionIndex**

Action: We take the word **dictionary**[**wordIndex**] (0-indexed) and put it over the sequence, starting from **positionIndex**. The substring [**positionIndex, positionIndex** + |**word**|] is replaced by the word.

Condition: 0 <= **wordIndex** < |**dictionary**|**.**

0 <= **positionIndex** <= **N** – |**word**|.

The word should not get out of the boundaries of the sequence.

Grading:

If there is an invalid operation in the output file, the program receives 0 points for the test case.
If the length of the final sequence is not **N**, the program receives 0 points for the test case.

Otherwise we define *yourScore* = ∑ **charDifference**[**finalSequence**[i]][**targetSequence**[i]] for i = {0 ... N - 1}
If *minScore* is the minimum score, received by all programs for this test case, your program receives (*minScore / yourScore*) ^ 2 percent of the points for the test case

Limits:

In all tests:
K <= 26

In 10% of tests:
N <= 100
T <= 20
V <= 5
$|word_i|$ <= 10
charDifference[i][j] <= 20

In 20% of tests:
N <= 1000
K <= 5
T <= 100
V = 0
charDifference[i][j] <= 20

In 20% of tests:
N <= 5000
T <= 500
V <= 10
$|word_i|$ <= 25
charDifference[i][j] <= 30

In 25% of tests:
N <= 20 000
K <= 15
T <= 1000
V <= 25
$|word_i|$ <= 50
charDifference[i][j] <= 40

In 25% of tests:
N <= 200 000
T <= 15 000
V <= 50
$|word_i|$ <= 100
charDifference[i][j] <= 50

Time limit: 8 sec
Memory limit: 256 MB

Preliminary tests: 20
Final tests: 100

Sample test:

fixdna.in	**fixdna.out**
10 3 5 abcbaaccbb bbaaccacba 0 2 3 2 0 1 3 1 0 3 abc aaba bcc	5 1 3 6 2 3 0 5 0 6 2 6 8 4 6 a
Score: 0	

Output explanation:
The current sequence after each operation is:
“abc**baac**cbb” -> “ab**baac**ccbb”
“**a**bbaacccbb” -> “bbaacccbb”
“bbaacccbb” -> “bbaacc**abc**”
“bbaacc**abc**” -> “bbaacc**cba**”
“bbaacccba” -> “bbaacc**a**cba”

The score is 0 since for i={0..N-1} charDifference[startSequence[i]][currentSequence[i]] == 0.
In other words startSequence == currentSequence.