Data Structures and Algorithms II
Fall 2020
Programming Assignment #4

This problem came from the 1998 regional ACM Programming Contest. As I described in class, it was the only question my Columbia team did not complete in the time limit. We had a solution which would work given unlimited time, but we did not realize it was an exponential time solution for contrived input. I am letting you know that the solution (probably) requires dynamic programming to be implemented efficiently. If you want to see how the problem was stated at the competition, check out this link:

http://www.acmgnyr.org/year1998/prob_g.html

The problem defines a "merge" of two strings as a third string containing all the characters from each of the original two strings mixed together. The two sets of characters can be interspersed, but the characters from each individual string cannot be permuted. For example, one possible merge of "hello" and "world" would be "wohrellold". However, the string "wohrelldol" is not a valid merge. Although this string contains all the correct characters, and "hello" and "world" are both subsequences, there is no way to select two subsequences with distinct characters to form both of the original two strings.

You are asked to write a program that accepts three strings at a time; we'll call them A, B, and C. All strings will consist of only lowercase letters. You can assume that A and B will contain at most 1000 letters, and C will contain at most 2000 letters. Your program should determine whether or not C is a valid merge of A and B. If so, the program should output C with the characters from A converted to uppercase. If more than one merge is possible, the letters of A should be made to occur as early as possible. If no merge is possible, the output should read "`*** NOT A MERGE ***`".

For this assignment, your program should prompt the user for the names of an input file and an output file. The input file will consist of multiple sets of three strings, one string per line (i.e., the number of rows in the file will be a multiple of three). Your program should read three strings at a time, and it should determine whether or not the third string is a merge of the first two. The output for each set of strings should be written to the output file as specified in the previous paragraph. Your program should continue to process sets of three strings until it reaches the end of the input file. Every line in the input file will be, and every line in the output file should be, followed by a single Unix-style newline character.

My major hint to you is that dynamic programming is (probably) necessary to write this program in a way such that it will run correctly for certain inputs in reasonable time. Simple algorithms will either get some cases wrong or require exponential time to run. Either top-down dynamic programming or bottom-up dynamic programming is appropriate (although you might run into stack-size problems with top-down dynamic programming on some systems). Note that you should be able to declare a global matrix (i.e., a two-dimensional array) that is big enough to handle all instances of this problem. Do not try to make the matrix a local variable; you might overflow the stack. There is no need to allocate the matrix dynamically.

A sample run of the program might look like this:

```
Enter name of input file: input.txt
Enter name of output file: output.txt
```

If the input file looks like this:

```
chocolate
chips
cchocholaiptes
chocolate
chips
bananasplit
abac
bad
ababacd
hello
world
wohrelldol
ab
ba
abab
zzzzzzzzzzzzzzzzzzzab
zzzzzzzzzzzzzzzzzzzac
zzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzacab
zzzzzzzzzzzzzzzzzzzabc
zzzzzzzzzzzzzzzzzzzacb
zzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzabcbac
```

Then the output file should look exactly like this:

```
CcHOChOLAipTEs
*** NOT A MERGE ***
ABAbaCd
*** NOT A MERGE ***
AbaB
ZZZZZZZZZZZZZZZZZZZzzzzzzzzzzzzzzzzzzacAB
*** NOT A MERGE ***
```

The first three examples (i.e., the first nine rows of the input and the first three rows of the output) were the examples provided at the actual competition. I contrived the other four examples to show cases that make the problem more difficult. Of course, I will test your programs with several additional difficult (and in some cases, much longer) test cases.

Submit your program to me via e-mail (*carl.sable@cooper.edu*). I encourage you to send early presubmissions; I will reply with responses similar to those given at the contest (it may take a day or longer). The program is due before midnight on the night of Wednesday, December 9.