Introduction to Analysis of Algorithms	Homework 6
CS 4820 Fall 2020	Due Tuesday, November 3, 2020

Instructions. Submit a .pdf file with your typeset solutions to questions 1 and 2 and a text file with your source code for question 3 in a .zip file to CMS. Do not include the compiled object or other compilation artifacts. You will also need to run your program on the autograder as with previous programming exercises.

All questions are worth 10 points. Remember that when asked to design an algorithm, you must also prove correctness and analyze its running time. When a problem asks for an efficient algorithm, you should strive to be as efficient as possible.

1. Consider the following variant of TSP. You are a traveling salesperson with a product that you can sell to retail stores in several major cities. For each pair of cities i and j, a ticket for a direct flight from city i to city j will cost you c_{ij}. You also know that you can achieve v_i in sales if you visit city i. You would like to arrange an itinerary that starts from your home city s, visits some subset of the cities (not necessarily all) in some order, and returns to s so as to maximize your total profit, which is the total sales revenue minus transportation costs. State a decision-problem version of this optimization problem and prove that it is NP-complete.

2. Show that the following problem is NP-complete. Given an undirected graph $G = (V, E)$ and numbers $k, \ell \geq 0$, does there exist a set $U \subseteq V$ of k or fewer nodes such that every node in V has a path of length ℓ or less to a node in U? (If you get stuck, a hint is given here, but try first without it.)

3. In this programming exercise, you will write a program to compute edit distances. The *edit distance* between two words is the smallest number of single-letter insertions, deletions, or substitutions needed to change one word to the other. This is a special case of the sequence alignment problem as described in K&T §6.6, p. 278 in which the gap penalty and all mismatch penalties are 1. In the notation of K&T, $\alpha_{pq} = 1$ for $p \neq q$, $\alpha_{pp} = 0$, and $\delta = 1$. The input to your program will be a pair of words over the alphabet a-z. The output of your program should be the edit distance followed by an optimal alignment.

   Input format The input to your program will be provided as an ASCII character stream on the standard input. It will consist of two lines terminated by newline characters (`'\n'`, `0x0A`). The two lines will contain the two words to be aligned, each word on a line by itself. Each word will consist of a sequence of characters from the alphabet a-z containing no spaces or other whitespace characters.

   Output format Your program should produce three lines, each terminated with a newline, on the standard output. The first line should contain a single nonnegative integer representing the edit distance (penalty of the optimal alignment). The second and third lines should contain the two words, one on each line, with spaces inserted to show an optimal alignment. If an occurrence of p in the first word is aligned with an occurrence of q in the second, then those two occurrences should be the same distance from the left end of the line. Thus the two occurrences should appear one directly above the other when your output is printed in a fixed-width font. If an occurrence of p in either word is unmatched in the alignment, then that occurrence of p should appear across from a space in the other word. There should be no extraneous gaps; that is, there should be no spaces across from each other.

   Examples The input

   ```
   occurrence
   occurance
   ```

   might generate the output

```
2
occurrence
occura nce
```

The input

```
abracadabra
candelabra
```

might generate the output

```
7
 abracadabra
candela  bra
```

Note there can be more than one alignment with the same edit distance.

Important points We guarantee that any input we provide to test your code will conform to the format described above, so you do not need to check for format errors. All words we use to test your code will be over the alphabet a-z.

We will test your code on some large instances of the problem, so make sure your implementation is efficient.

You may use any of the following languages: Python, Java, C, C++, or OCaml. Please format your code attractively and provide copious comments. As before, we will grade your program using an autograder. The autograder can be accessed at https://cs4820.cs.cornell.edu/, code named *Alignment*. The source code submitted to CMS must match the latest version uploaded to the autograder.