02-713 Homework #7: Dynamic Programming II
Due: Mar. 24 by 9:30am

You may discuss these problems with your current classmates, but you must write up your solutions independently, without using common notes or worksheets. You must indicate at the top of your homework who you worked with. Your write up should be clear and concise. You are trying to convince a skeptical reader that your answers are correct. Your homework should be submitted via Autolab (`https://autolab.cs.cmu.edu/02713-s14/`) as a typeset PDF. A LaTeX tutorial and template are available on the class website if you choose to use that system to typeset. For problems asking for an algorithm: describe the algorithm, an argument why it is correct, and an estimation of how fast it will run. Use O notation for running times.

1. Suppose we extend the definition of "edit distance" to include the operation **swap** that allows two adjacent characters to be swapped. We require that each character is involved in at most one swap so that we can represent this new edit distance via an alignment such as:

   ```
   tehs-icktoxin
   |\/X |||X||
   thequickfox--
   ```

   where the characters `\/` indicate that the `h` and `e` where swapped. We are given as input two strings a and b, and parameters *gap*, *swap*, and *cost(x,y)* that, respectively, give the cost of inserting a '-', swapping two characters, and aligning two characters. Give a dynamic programming algorithm to compute the lowest cost alignment in this case.

2. Bitonic traveling salesman. You are given a set of cities $p_1, \ldots, p_n$ in the plane. We assume the distance between two cities is the standard Euclidean distance, and that no two cities have exactly the same x or y coordinate. A *bitonic* traveling salesman tour is a tour that starts at the most western city, and then proceeds strictly to the east until the most eastern city, at which point it turns around and heads strictly to the west until it returns back to the starting city.

   Give an $O(n^2)$-time algorithm for finding the optimum bitonic tour.

3. A context-free grammar over alphabet Σ is given by a set of pairs called *productions* of one of the following two forms:

   $$A \to BC$$
   $$A \to a$$

   where capital letters such as A, B and C represent *non-terminals* and lowercase letters represent symbols from your alphabet Σ. There is always at least one production with the special start symbol S on its lefthand side (before the $\to$). More formally, a grammar G is a set of 4 things (N, Σ, P, S), where N is the set of non-terminals, Σ is the alphabet, P is the set of productions, and S is the starting non-terminal. This is called the Chomsky normal form of a grammar. The productions define replacement rules where the lefthand side can be replaced

by the righthand side. Using these rules, context free grammars can generate sets of strings. For example, suppose we have the grammar:

$$
\begin{aligned}
S &\to QA \\
S &\to RB \\
S &\to \texttt{a} \\
S &\to \texttt{b} \\
Q &\to AS \\
R &\to BS \\
A &\to \texttt{a} \\
B &\to \texttt{b}
\end{aligned}
$$

Then this grammar can generate the string `abababababa` via the productions:

> S → QA → ASA → aSa→ aRBa → aBSBa → abSba → abQAba → abASAba → abaSaba → abaRBaba → abaBSBaba → ababSbaba → abababababa

In fact, the grammar above can generate exactly the set of all odd-length palindromes on the alphabet $\{a, b\}$. The set of strings that a grammar G can generate is called its *language* and denoted by $L(G)$.

Given a grammar G over alphabet Σ with a set of productions P, give an $O(n^3)$ algorithm to determine whether a given string $c_1 c_2 \ldots c_n$ is in $L(G)$.