CSCI 3104 **Instructor Buxton**
Problem Set 5 **Summer 2019, CU-Boulder**

1. (60 points) Recall that the *string alignment problem* takes as input two strings x and y, composed of symbols $x_i, y_j \in \Sigma$, for a fixed symbol set Σ, and returns a minimal-cost set of *edit* operations for transforming the string x into string y.

   Let x contain n_x symbols, let y contain n_y symbols, and let the set of edit operations be those defined in the lecture (substitution, insertion, and deletion).

   Let the cost of *indel* be 1 and the cost of *sub* be 2, except when $x_i = y_j$, which is a "no-op" and has cost 0.

   In this problem, we will implement and apply three functions.

   (i) `alignStrings(x,y)` takes as input two ASCII strings x and y, and runs a dynamic programming algorithm to return the cost matrix S, which contains the optimal costs for all the subproblems for aligning these two strings.

```
alignStrings(x,y) :                 // x,y are ASCII strings
   S = table of length nx by ny     // for memoizing the subproblem costs
   initialize S                     // fill in the basecases
   for i = 1 to nx
        for j = 1 to ny
             S[i,j] = cost(i,j)     // optimal cost for x[0..i] and y[0..j]
   }}
   return S
```

   (ii) `extractAlignment(S,x,y)` takes as input an optimal cost matrix S, strings x, y, and returns a vector a that represents an optimal sequence of edit operations to convert x into y. This optimal sequence is recovered by finding a path on the implicit DAG of decisions made by `alignStrings` to obtain the value $S[n_x, n_y]$, starting from $S[0, 0]$.

```
extractAlignment(S,x,y) :  // S is an optimal cost matrix from alignStrings
   initialize a            // empty list of edit operations
   [i,j] = [nx,ny]         // initialize the search for a path to S[0,0]
   while i > 0 or j > 0
        a.prepend(determineOptimalOp(S,i,j,x,y)) // what was an optimal choice?
        [i,j] = updateIndices(S,i,j,a)           // move to next position
   }
   return a
```

   When storing the sequence of edit operations in a, use a special symbol to denote no-ops.

(iii) `commonSubstrings(x,L,a)` which takes as input the ASCII string x, an integer $1 \leq L \leq n_x$, and an optimal sequence a of edits to x, which would transform x into y. This function returns each of the substrings of length at least L in x that aligns exactly, via a run of no-ops, to a substring in y.

(a) From scratch, implement the functions `alignStrings`, `extractAlignment`, and `commonSubstrings`. You may not use any library functions that make their implementation trivial. Within your implementation of `extractAlignment`, ties must be broken uniformly at random.

Submit (i) a paragraph for each function that explains how you implemented it (describe how it works and how it uses its data structures), and (ii) your code implementation, with code comments.

Hint: test your code by reproducing the `APE` / `STEP` and the `EXPONENTIAL` / `POLYNOMIAL` examples in the lecture notes (to do this exactly, you'll need to use unit costs instead of the ones given above).

(b) Using asymptotic analysis, determine the running time of the call
`commonSubstrings(x, L, extractAlignment( alignStrings(x,y), x,y ) )`
Justify your answer.

(c) String alignment algorithms can be used to detect changes between different versions of the same document (as in version control systems) or to detect verbatim copying between different documents (as in plagiarism detection systems).

The two `data_string` files for PS5 (see class Moodle) contain actual documents recently released by two independent organizations. Use your functions from (1a) to align the text of these two documents. Present the results of your analysis, including a reporting of 10 of the longest substrings in x of length $L = 20$ or more that could have been taken from y, and briefly comment on whether these documents could be reasonably considered original works, under CU's academic honesty policy.

2. (10 points) Shadow is trying to figure out all the potential paths from the bank he is trying to rob to the party's hideout, given by the network below. Help him calculate the number of paths from node 1 to node 14.

   Hint: assume a "path" must have at least one edge in it to be well defined, and use dynamic programming to fill in a table that counts number of paths from each node j to 14, starting from 14 down to 1.

$$\begin{tikzcd}
1 \arrow[r] \arrow[dr] & 2 \arrow[r] \arrow[d] \arrow[dr] & 3 \arrow[r] \arrow[d] \arrow[dr] & 4 \arrow[r] & 5 \arrow[r] \arrow[d] & 6 \arrow[d] & \\
& 7 \arrow[r] & 8 \arrow[r] & 9 \arrow[r] \arrow[dr] & 10 \arrow[r] \arrow[dr] & 11 \arrow[dr] & \\
& & & & 12 \arrow[r] & 13 \arrow[r] & 14
\end{tikzcd}$$

Figure 1: Thormund's Network

3. (20 points) As a new event in the Triwizard Tournament, students must compete in pairs in the following game. An even number n of magical items are laid out in a row, with the i-th item having a value of $v(i)$ in Knuts for each $i = 1, 2, \ldots, n$. The players alternate turns, and on each turn, the player may choose either the first item or the last item remaining in the row, then remove it from the row and add it to their pile. The player with the highest-valued pile at the end wins. (The value of a player's pile at the end is simply the sum of the value of the items in the pile.) We will use dynamic programming to help us analyze this game.

   (a) For a given vector v of values, let $F(i, j)$ denote the maximum value that the first player can definitely achieve using the values $v[i], \ldots, v[j]$—this means a *tight*, *exact* upper bound, an amount which the first player cannot beat, but that the first player can in fact achieve if he/she plays optimally. Write down a recurrence relation for $F(i, j)$ and prove that it is correct. Be sure to include a base case! You can assume that this problem has optimal substructure.

   (b) Based on your recurrence from part (a), describe a dynamic programming table and the order in which it should be filled in.

   (c) Write pseudo-code for the dynamic programming solution you described in part (b) and analyze its run-time.