Homework 7 Solution

1. *Let G be a directed weighted graph, where $V(G) = \{v_1, v_2, \cdots, v_n\}$. Let B be an $n \times n$ matrix such that entry b_{ij} denotes the distance in G from v_i to v_j (using a directed path). Now we are going to insert a new vertex $\mathrm{v_{n+1}}$ into G. Let $\mathrm{w_i}$ denote the weight of the edge $(\mathrm{v_i}, \mathrm{v_{n+1}})$ and $\mathrm{w_i'}$ denote the weight of the edge $(\mathrm{v_{n+1}}, \mathrm{v_i})$. (if there is no edge from $\mathrm{v_i}$ to $\mathrm{v_{n+1}}$ or from $\mathrm{v_{n+1}}$ to $\mathrm{v_i}$, then $\mathrm{w_i}$ or $\mathrm{w_i'}$ is infinity, respectively.) Describe an algorithm to construct an $(\mathrm{n}+1) \times (\mathrm{n}+1)$ distance matrix $\mathrm{B'}$ from B and values of $\mathrm{w_i}$ and $\mathrm{w_i'}$ for $1 \le \mathrm{i} \le \mathrm{n}$. (Note that the graph G itself is not given.) Your algorithm should work in $\mathrm{O(n^2)}$ time. (Hint: Use the Floyd's dynamic programming algorithm for finding all pairs shortest paths.)*

   Step 1: Compute $B_{n+1,\mathrm{i}} = \min\limits_{1 \le j \le n} \{w_{\mathrm{j}}' + B_{j,\mathrm{i}}\}$ and $B_{i,n+1} = \min\limits_{1 \le j \le n} \{\mathrm{w_j} + B_{\mathrm{i,j}}\}$

   Step 2: Compute $B_{i,j}' = \min\{\mathrm{B_{i,j}}, \mathrm{B_{i,n+1}} + \mathrm{B_{n+1,j}}\}$ for $1 \le i \le n+1$,$1 \le j \le n+1$

   Time complexity analysis:

   Step 1 takes $O(n^2)$. Step 2 takes $O(n^2)$. Thus, the algorithm works in $O(n^2)$

2. *Given two strings X and Y, respectively, of length m and n defined over a set $\Sigma = \{\mathrm{a_1}, \mathrm{a_2}, \cdots, \mathrm{a_k}\}$ of finitely many symbols, we are interested in computing an optimal (i.e., minimum cost) alignment of two strings, where two possible alignments are defined as (i) a mismatch with cost $\mathrm{c_m}$ and (ii) a gap with cost $\mathrm{c_g}$.*

   Consider the following two sequences defined over $\Sigma = \{\mathrm{A, G, C, T}\}$, where

   $\mathrm{X} = \{\mathrm{A\ \ A\ \ C\ \ A\ \ G\ \ T\ \ T\ \ A\ \ C\ \ C}\}$ *and*

   $\mathrm{Y} = \{\mathrm{T\ \ A\ \ A\ \ G\ \ G\ \ T\ \ C\ \ A}\}$

   In the following alignment, there are two mismatches and four gaps with total cost $2\mathrm{c_m} + 4\mathrm{c_g}$:

   $\{-\mathrm{A\ \ A\ \ C\ \ A\ \ G\ \ T\ \ T\ \ A\ \ C\ \ C}\}$ *and*

   $\{\mathrm{T\ \ A\ \ A} - \mathrm{G\ \ G\ \ T}\ \ -\ \ -\ \mathrm{C\ \ A}\}$.

   Give a dynamic programming algorithm to solve this problem.

   Let $C(i,j)$ denote the minimum cost of the aliment of two strings $X_i = \{\mathrm{x_1}, x_2, \cdots, x_i\}$ and $Y_j = \{y_1, y_2, \cdots, y_j\}$.

   Let $S(i,j) = \begin{cases} c_m, & \text{if } \mathrm{x_i} \ne y_j \\ 0, & \text{if } \mathrm{x_i} = y_j \end{cases}$

   Then $C(i,j) = \min\{C(i-1,j-1) + S(i,j), C(i,j-1) + c_g, C(i-1,j) + c_g\}$

   The goal is to compute $C(\mathrm{m,n})$.

   Time complexity analysis.

   Calculating each block of the DP matrix takes $O(1)$. Thus, the algorithm runs in $O(n^2)$.

3. *You are given a Boolean expression consisting of a string of the symbols 'true', 'false', 'and', 'or', and 'xor'. Count the number of ways to parenthesize the expression such that it will evaluate to true. For example, there are 2 ways to parenthesize 'true and false xor true' such that it evaluates to true. Give a dynamic programming algorithm to solve this problem and analyze the time complexity of your algorithm.*

Let $x_i = \begin{cases} \text{true} \\ \text{false} \end{cases} 0 \le i \le n, o_j = \begin{cases} \text{and} \\ \text{or} \\ \text{xor} \end{cases}, 1 \le j \le n$

Define $T(i,j) = \{\text{number of ways to parenthesize the string } \mathrm{x_i o_i \cdots x_j} \text{ into true}\}$

Define $\mathrm{F}(i,j) = \{\text{number of ways to parenthesize the string } \mathrm{x_i o_i \cdots x_j} \text{ into false}\}$

Define $C_{i,j,k} = \begin{cases} T(i,k)\cdot T(k+1,j), o_k = \text{and} \\ T(i,k)\cdot T(k+1,j) + T(i,k)\cdot F(k+1,j) + F(i,k)\cdot T(k+1,j), o_k = \text{or} \\ T(i,k)\cdot F(k+1,j) + F(i,k)\cdot T(k+1,j), o_k = \text{xor} \end{cases}$

$T(i,j) = \Sigma_{i \le k \le j} C_{i,j,k}$

Our goal is to compute DP matrix T to get $T(0,n)$. It takes $O(n)$ to compute one block of matrix. Thus, the algorithm should run in $O(n^3)$.

4. *The input to this problem is a sequence* S *of* n *integers (not necessarily positive). The problem is to find the consecutive subsequence of* S *with maximum sum. Consecutive means that you are not allowed to skip numbers. For example, if the input was 12, -14, 1, 23, -6, 22, -34, 13, the output would be 1, 23, -6, 22. Give a linear time algorithm for this problem.*

Applying Kadane's Algorithm can solve the problem.

Algorithm implementation in Matlab:

```
function [ maxsum,maxstartindex,maxendindex ] maximumsub( input)
maxsum=-Inf;
maxstartindex=0;
maxendindex=0;
currentmaxsum=0;
currentstartindex=1;
for currentendindex=1:n
    currentmaxsum=currentmaxsum+input(currentendindex);
    if currentmaxsum>maxsum
        maxsum=currentmaxsum;
        maxstartindex=currentstartindex;
        maxendindex=currentendindex;
    end
    if currentmaxsum<0
        currentmaxsum=0;
        currentstartindex=currentendindex+1;
    end
end
end
This can be viewed as a trivia of DP.
```