1. A *metapalindrome* is a decomposition of a string into a sequence of palindromes, such that the sequence of palindrom lengths is itself a palindrome. For instance, the string BOBSMAMASEESAUKULELE can be decomposed as (BOB)(S)(MAM)(ASEESA)(UKU)(L)(ELE) where the lengths form the sequence (3,1,3,6,3,1,3). The length of the metapalindrome is the number of parts to which the string has been decomposed. Design an efficient algorithm to find the shortest metapalindrome of a given input string.

   Assume that you have access to a function IsPalindrome that checks if a given string is a palindrome.

2. Design an efficient algorithm to compute the longest contiguous subarray that appears both in forward and backward in a string $S[1, 2, \ldots, n]$ such that the occurrences do not overlap. For instance, in REDIVIDE, the string EDIV appears in both directions, but the letter V overlaps. The correct value for this string is 3, corresponding to the string EDI.

3. Suppose you are given a set L of n line segments such that each line segment has one endpoint on the line $y = 0$ and another on the line $y = 1$. Assume that all the $2n$ endpoints are distinct.
   (a) Design an efficient algorithm to find the largest subset $L' \subseteq L$ such that no line segments in L' intersect.

   Hint: Reduce this problem to finding the longest increasing subsequence of a sequence.

   (b) Design an efficient algorithm to find the largest subset $L' \subseteq L$ such that every pair of line segments in L' intersect.

   Hint: Reduce this problem to finding the longest common subsequence of two sequences.

4. Let $T(V, E, w)$ be an edge-weighted tree where $w : E \to \mathbb{R}^+$. A matching is a subset $E' \subseteq E$ of edges such that no two edges in E' share an end-point. Give a linear-time algorithm to compute the maximum-weight matching in a tree.

5. Let $p(u, v)$ denote the vertex occuring just before v in the shortest path from u to v in a graph. Describe an efficient algorithm to construct the matrix of predecessors $p(u, v)$ by suitably modifying the Floyd-Warshall algorithm.

6. Suppose that we have an edge-weighted graph $G(V, E, w)$ where the edge-weights could potentially be negative. Furthermore, the graph could contain negative weight cycles. Give an efficient algorithm that obtains the APSP $d(u, v)$ that satisfies the following conditions.

   - If v is not reachable from u, $d(u, v)$ should be ∞.
   - If v is reachable from u via a walk that contains a negative weight cycle, then $d(u, v)$ should be set to $-\infty$.
   - Otherwise, $d(u, v)$ should be the actual shortest distance from u to v in G.

7. Suppose that $G(V, E, w)$ is a weighted digraph that do not contain negative weight cycles. We will design a slightly different $O(V^3)$-time algorithm to find APSP.
   (a) Let $v \in V$ be an arbitrary vertex. Design an $O(V^2)$-time algorithm to create a new graph $G'(V \setminus \{v\}, E', w')$ such that all the shortest path distances in G' is same as in G.
   (b) Suppose that we have computed APSP in G'. Design an $O(V^2)$-time algorithm to compute the shortest paths from v to all the other vertices, and from all the other vertices to v.
   (c) Describe how Parts (a) and (b) can be combined to obtain an $O(V^3)$-time algorithm for APSP.

8. Suppose you have a currency exchange where there are n different currencies and for each pair of currencies (i, j) the value $E(i, j)$ gives the exchange rate for converting 1 unit of currency i to currency j. An *arbitrage cycle* is a sequence of conversions that you can do such that starting with 1 unit of currency i, you end up with >1 unit of i. For instance, if the currencies are \$, ¥ and € such that the exchange rates are as follows 1\$= 150¥, 1¥=0.0068€, and 1€=1.1\$, then we can convert 1\$ to ¥ to € and then back to \$ to obtain 1.122\$. Give an efficient algorithm to check if the given input of currency exchanges contain an arbitrage cycle.