Coefficients of the polynomial C, C_{coeff} will be obtained by performing an inverse transform on T_c as shown in part (c) of this problem

$$C_{coeff} = 6 \cdot \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 5 & 4 & 6 & 2 & 3 \\ 1 & 4 & 2 & 1 & 4 & 2 \\ 1 & 6 & 1 & 6 & 1 & 6 \\ 1 & 2 & 4 & 1 & 2 & 4 \\ 1 & 3 & 2 & 6 & 4 & 5 \end{pmatrix} \begin{pmatrix} 6 \\ 3 \\ 0 \\ 3 \\ 0 \\ 3 \end{pmatrix} = \begin{pmatrix} 6 \\ 1 \\ 1 \\ 3 \\ 1 \\ 1 \end{pmatrix} \tag{10}$$

Transforming 6 to -1 in mod 7 arithmetic we get the product polynomial C as

$$1 \cdot x^5 + 1 \cdot x^4 + 3 \cdot x^3 + 1 \cdot x^2 + 1 \cdot x^1 + (-1) \cdot x^0 \tag{11}$$

3 Problem 2.31

This problem presents a fast algorithm for computing the greatest common divisor (gcd) of two positive integers a and b. Before giving the algorithm we need to prove the following properties

$$gcd(a,b) = \begin{cases} 2 \cdot gcd(\frac{a}{2}, \frac{b}{2}) & \text{if a,b are even} \\ gcd(a, \frac{b}{2}) & \text{if a is odd, b is even} \\ gcd(\frac{a-b}{2}, b) & \text{if a, b are odd} \end{cases}$$

(a)If a and b are even numbers, 2 is surely a common divisor of a and b. Therefore the greatest common divisor will be 2 times the gcd of numbers $\frac{a}{2}$ and $\frac{b}{2}$. If a is odd and b is even, we know for sure that b is divisible by 2 while a is not. Therefore gcd(a,b) remains same as the gcd of a and $\frac{b}{2}$. The third property follows from the fact that if a and b are odd, then (a-b) will be even. Since gcd(a,b) = gcd(a-b,b) and a-b is even now we can apply the second property to get the desired result.
(b) The recursive algorithm for gcd is given as

```
procedure gcd(a,b)
Input:  Two n-bit integers a,b
Output:  GCD of a and b
if a = b:
  return a
else if (a is even ∧ b is even):
  return 2 · gcd(a/2, b/2)
else if (a is odd ∧ b is even):
  return gcd(a, b/2)
else if (a is odd ∧ b is odd ∧ a > b):
  return gcd((a−b)/2, b)
else if (a is odd ∧ b is odd ∧ a < b):
  return gcd(a, (b−a)/2)
```

(c)Complexity analysis of the algorithm: Assume that a and b are n-bit numbers. Size of a and b is $2n$ bits. Out of 4 if conditions, every one except the case when a is odd and b is even, decreases the size of a and b to $2n-2$ bits whereas that case decreases it to $2n-1$ bits. Each of the operations is constant time operation as we are dividing or multiplying the numbers by 2. For two cases subtraction of two n-bit numbers are involved which is c·n where n is the number of bits of the operand. Therefore in the worst case the recurrence is given by

$$\begin{aligned}
T(2n) &= T(2n-1) + cn \\
T(2n-1) &= T(2n-2) + cn \\
T(2n-2) &= T(2n-3) + c(n-1) \text{ both operands are n-1 bits} \\
T(2n-3) &= T(2n-4) + c(n-1) \\
&... \\
T(2) &= T(1) + c
\end{aligned}$$

Using substitution, we can obtain T(2n) as

$$T(2n) = 2c \cdot \sum_{i=1}^{n} i \tag{12}$$

which is O(n^2). Compared to the O(n^3) running time of Euclid's the divide-and-conquer algorithm is faster.

4 Problem 4.4

Counterexample is shown in Figure 1. Using the proposed approach we obtain the shortest cycle as {3, 4, 5, 6, 3} but the shortest cycle in this case is {2,3,6,2}. The depth first search labels for each vertex $i : i \in (1..6)$ is $i - 1$.

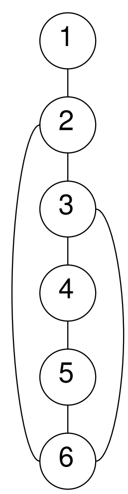

Figure 1: Counterexample

5 Problem 4.7

Given a directed graph G = (V,E) with no constraints on edges along with a specific node s $\in$ V and a tree T = (V,$\hat{E}$) where $\hat{E} \subseteq E$, we have to give a linear time algorithm to check whether T is a shortest path tree for G with starting point S. In this problem the idea is to effectively make use of shortest path distances given on the associated shortest path tree T. Obtain the shortest path distance from each vertex of the tree and annotate the shortest path distance on each vertex of the graph G. Now run subroutine *update* (Bellman-Ford algorithm page 117) on every edge of the graph G. By definition of shortest path distances, *update* should not change any shortest path distance on each node. If *update* changes the shortest path