P1) Given the following graph with 2n vertices. Design a dynamic program that runs in polynomial time and outputs the number of independent sets of this graph. For example, if n = 2 you

\[
\begin{array}{c}
\text{\includegraphics[width=0.5\textwidth]{graph.png}} \\
\text{n}
\end{array}
\]

should output 7. See below for all these 7x independent sets when n = 2,

\[
\begin{array}{cccccccc}
\text{\includegraphics[width=0.1\textwidth]{set1.png}} & \text{\includegraphics[width=0.1\textwidth]{set2.png}} & \text{\includegraphics[width=0.1\textwidth]{set3.png}} & \text{\includegraphics[width=0.1\textwidth]{set4.png}} & \text{\includegraphics[width=0.1\textwidth]{set5.png}} & \text{\includegraphics[width=0.1\textwidth]{set6.png}} & \text{\includegraphics[width=0.1\textwidth]{set7.png}}
\end{array}
\]

P2) You have just learned wall climbing. The wall that you are practicing on is n + 1 squares tall and k + 1 squares wide. The bottom left square is at (0, 0) and the top right is at (k, n). There are two ladders on this wall one is from (z1, 0) all the way to (z1, n) and the other from (z2, 0) all the way to (z2, n). You can assume 0 ≤ z1 < z2 ≤ k.

Your friend has hanged two coins in every row of this wall; say the coins at row i are located at (x_i, y_i) where 0 ≤ x_i < y_i ≤ k for all i. You start at (0, 0) and you want to collect all coins as fast as possible. In each step, you either go left, go right, or if you are on the ladder you can go up, but if you go up you cannot come back down; each of these moves take 1 time-step. Design an algorithm that runs in time polynomial in n, k and outputs the smallest number of time-steps you need to collect all coins.

P3) You are given a tree T where every node i has weight w_i ≥ 0. Design a polynomial time algorithm to find the weight of the smallest vertex cover in T. For example, suppose in the following picture w_a = 3, w_b = 1, w_c = 4, w_d = 3, w_e = 6. Then, the minimum cost vertex cover has nodes a, b with weight 3 + 1 = 4.

\[
\begin{array}{c}
\text{\includegraphics[width=0.2\textwidth]{tree.png}}
\end{array}
\]

P4) Suppose we have received a large rectangular marble slab of size W × H and we want to cut the slab to obtain rectangular marble plates of sizes W_1 × H_1, W_2 × H_2, ..., W_n × H_n. Any
piece of marble (the slab or the plates cut from it) can be cut either horizontally or vertically into two rectangular plates with integral widths and heights, cutting completely through that piece. This is the only way to cut pieces and pieces cannot be joined together.

<table>
<thead>
<tr>
<th>10×4</th>
<th>10×4</th>
</tr>
</thead>
<tbody>
<tr>
<td>6×2</td>
<td>6×2</td>
</tr>
<tr>
<td>7×5</td>
<td>7×5</td>
</tr>
<tr>
<td>7×5</td>
<td>7×5</td>
</tr>
</tbody>
</table>

Since the marble has a pattern on it, the plates cannot be rotated: if a plate of size \( A \times B \) is cut, then it cannot be used as a plate of size \( B \times A \) unless \( A = B \). We can make zero or more plates of each desired size. A marble plate is wasted if it is not of any of the desired sizes after all cuts are completed. The question is how to cut the initial slab so that as little of it as possible will be wasted.

As an example, assume that original slab is \( 21 \times 11 \) and the desired plate sizes are \( 10 \times 4, 6 \times 2, 7 \times 5, \) and \( 15 \times 10 \). The minimum possible area wasted is 10, and the figure (below) shows one sequence of cuts with total waste area of size 10. Design an algorithm that runs in time polynomial in \( n, W, H \) that outputs the minimum possible waste area.

P5) **Extra Credit:** Given a sequence of positive numbers \( x_1, \ldots, x_n \) and an integer \( k \), design a polynomial time algorithm that outputs

\[
\sum_{S \in \binom{n}{k}} \prod_{i \in S} x_i,
\]

where the sum is over all subsets of size \( k \).