1. \( m \) and \( n \) are positive integers.

You are playing a game which is played on a board with \( n \) squares, labelled 1, 2, \ldots, \( n \). You are initially on square 1. You have to reach square \( n \) while collecting the maximum number of points. If you reach square \( i \), you collect \( a_i \) points.

When you are on square \( i \), then for every \( j \in \{1, 2, \ldots, m\} \), you have the ability to jump to square \( i + j \), at the cost of losing \( b_j \) points.

(a) (5 points) Give an instance where the following greedy algorithm fails to find the plan with the optimum payoff:

- “When you are at position \( i \), jump to the position \( i + j \) such that \( a_{i+j} - b_j \) is maximized.”

(b) (15 points) Give a dynamic programming algorithm which finds the plan with the optimum payoff.

2. (5 points) Give an example of a 4-vertex graph where Prim’s algorithm and Kruskal’s algorithm could give different minimum spanning trees as output.

3. (5 points) Give an example of a 4-vertex graph where Prim’s algorithm and Kruskal’s algorithm always give the same minimum spanning tree as output.

4. (10 points) Describe all the steps in the execution of the Bellman-Ford algorithm on the following instance. We are computing the shortest path from \( s \) to \( t \).
5. (10 points) Describe the execution of the Ford-Fulkerson algorithm on the following instance. We are computing the maximum flow from $s$ to $t$.

6. (20 points) For each nonnegative integer $k$, we have a $2^k \times 3^k$ matrix $M_k$ defined as follows:

$$M_0 = [1].$$

$$M_{k+1} = \begin{bmatrix}
M_k & 2M_k & 3M_k \\
4M_k & 5M_k & 6M_k
\end{bmatrix}$$

for each $k \geq 0$

Thus we have, for example:

$$M_1 = \begin{bmatrix}
1 & 2 & 3 \\
4 & 5 & 6
\end{bmatrix}$$

$$M_2 = \begin{bmatrix}
1 & 2 & 3 & 2 & 4 & 6 & 3 & 6 & 9 \\
4 & 5 & 6 & 8 & 10 & 12 & 12 & 15 & 18 \\
4 & 8 & 12 & 5 & 10 & 15 & 6 & 12 & 18 \\
16 & 20 & 24 & 20 & 25 & 30 & 24 & 30 & 36
\end{bmatrix}$$

Let $n = 3^k$. Design an algorithm running in time $\Theta(n)$ which does the following: given a vector $x$ of length $n$, the algorithm computes the matrix-vector product $M_k \cdot x$ (which is a vector of length $2^k$).