1. Consider the following solitaire game. The puzzle consists of an \( n \times m \) grid of squares, where each square may be empty, occupied by a red stone, or occupied by a blue stone. The goal of the puzzle is to remove some of the given stones so that the remaining stones satisfy two conditions: (1) every row contains at least one stone, and (2) no column contains stones of both colors. For some initial configurations of stones, reaching this goal is impossible.

**Proof.** We show that this puzzle is NP-hard by a reduction from 3SAT.

Let \( \Phi \) be a 3CNF boolean formula with \( n \) variables \( x_1, x_2, \ldots, x_n \) and \( m \) clauses. We transform this board into a puzzle in polynomial time as follows. The puzzle grid has \( n \) columns, one for each variable in \( \Phi \), and \( m \) rows, one for each clause in \( \Phi \). For all indices \( i \) and \( j \), we place a stone at position \((i, j)\) as follows:

- If the variable \( x_j \) appears in the \( i \)th clause of \( \Phi \), we place a blue stone at \((i, j)\).
- If the negated variable \( \bar{x}_j \) appears in the \( i \)th clause of \( \Phi \), we place a red stone at \((i, j)\).
- Otherwise, we leave cell \((i, j)\) blank.

We claim that this puzzle has a solution if and only if \( \Phi \) is satisfiable. This claim immediately implies that solving the puzzle is NP-hard. We prove our claim as follows:

\[ \implies \] First, suppose \( \Phi \) is satisfiable, and consider an arbitrary satisfying assignment. For each index \( j \), remove stones from column \( j \) according to the value assigned to \( x_j \):

- If \( x_j = \text{TRUE} \), remove all red stones from column \( j \).
- If \( x_j = \text{FALSE} \), remove all blue stones from column \( j \).

In other words, remove precisely the stones that correspond to FALSE literals. Because every variable appears in at least one clause, no column contains stones of both colors. On the other hand, each clause of \( \Phi \) must contain at least one \( \text{TRUE} \) literal, and thus each row still contains at least one stone. We conclude that the puzzle is solvable.

\[ \iff \] On the other hand, suppose the puzzle is solvable; consider an arbitrary solution. For each index \( j \), assign a value to \( x_j \) depending on the colors of stones left in column \( j \):

- If column \( j \) contains blue stones, set \( x_j = \text{TRUE} \).
- If column \( j \) contains red stones, set \( x_j = \text{FALSE} \).
- If column \( j \) is empty, set \( x_j \) arbitrarily.

In other words, assign values to the variables so that the literals corresponding to the remaining stones are all \( \text{TRUE} \). Each row still has at least one stone, so each clause of \( \Phi \) contains at least one \( \text{TRUE} \) literal, so this assignment makes \( \Phi = \text{TRUE} \). We conclude that \( \Phi \) is satisfiable.

The reduction clearly requires only polynomial time, even if we use brute force. \( \square \)
2. Everyone's having a wonderful time at the party you're throwing, but now it's time to line up for **The Algorithm March** (アルゴリズムこうしん)! This dance was originally developed by the Japanese comedy duo Itsumo Kokokara (いつもここから) for the children's television show **PythagoraSwitch** (ピタゴラスイッチ). The Algorithm March is performed by a line of people; each person in line starts a specific sequence of movements one measure later than the person directly in front of them. Thus, the march is the dance equivalent of a musical round or canon, like "Row Row Row Your Boat". Proper etiquette dictates that each marcher must know the person directly in front of them in line, lest a minor mistake during lead to horrible embarrassment between strangers.

Suppose you are given a complete list of which people at your party know each other. Prove that it is NP-hard to determine the largest number of party-goers that can participate in the Algorithm March. You may assume without loss of generality that there are no ninjas at your party.

**Solution:** Suppose there are \( n \) people at the party, arbitrarily indexed from 1 to \( n \).

To make the problem concrete, I will assume that the input to **ALGORITHMMARCH** is consists of an array of \( n \) linked lists, where the \( i \)th list contains the indices of every other people that person \( i \) knows. This input representation is the adjacency list of a directed graph \( G = (V, E) \), where the vertices \( V \) are the \( n \) people, and there is an edge \( u \rightarrow v \) if and only if person \( u \) knows person \( v \).

A line of people can participate in the Algorithm March if and only if each person knows the person in front of them, or equivalently, if and only if each vertex has an edge to its successor. In other words, every valid line of Algorithm Marchers is a path in the input graph \( G \) and vice versa.

Thus, the Algorithm March problem is the **LONGESTPATH** problem, which is NP-hard according to the notes.

**Rubric:** This is more detail than necessary for full credit. Yes, really.