1. Suppose we are given an $n \times n$ grid, some of whose cells are marked; the grid is represented by an array $M[1..n, 1..n]$ of booleans, where $M[i, j] =$ True if and only if cell (i, j) is marked. A monotone path through the grid starts at the top-left cell, moves only right or down at each step, and ends at the bottom-right cell. Our goal is to cover the marked cells with as few monotone paths as possible.

   ► **Not to submit:** Describe an algorithm to find a monotone path that covers the largest number of marked cells.

   Solution: We construct a graph G.

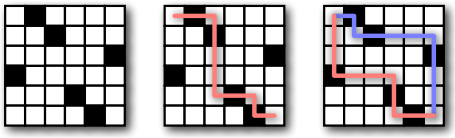
   - G has two vertices $u_{i,j}$ and $v_{i,j}$ for each pair of array indices i and j.
   - Each vertex $u_{i,j}$ has an edge to its partner $v_{i,j}$. Each partner edge has length 1 if $M[i, j] =$ TRUE and length 0 otherwise.
   - Each vertex $v_{i,j}$ has edges to its right neighbor $u_{i+1,j}$ and its downward neighbor $u_{i,j+1}$ (if they exist). Neighbor edges have length 0.

   Each monotone path through the grid is represented by a directed path in G. Conversely, every directed path in G represents a monotone path through the grid. Thus, if we can cover k marked cells with a monotone path, then there is a path of length k in the graph. Conversely, if the graph has a path of length k, that path corresponds to a monotone path that must cover k marked cells. So we can solve find the monotone path that covers the largest number of marked cells in the grid by finding the longest path in G from $u_{1,1}$.

   The graph G has $O(n^2)$ vertices and edges, and we can construct it in $O(n^2)$ time. Since G is a DAG, finding the longest path can be done in $O(|V| + |E|) = \boldsymbol{O(n^2)}$ time using Dynamic Programming. ■

   ► **Not to submit:** There is a natural greedy heuristic to find a small cover by monotone paths: If there are any marked cells, find a monotone path Π that covers the largest number of marked cells, unmark any marked cells covered by Π, and recurse. Show that this algorithm does *not* always compute an optimal solution.

   Solution: Here is a 6×6 counterexample:

   The greedy strategy is not optimal.

   The middle figure shows a greedy path; in fact any greedy monotone path covers the same four marked cells. If we start with a greedy path, we need two more monotone paths to cover the other two marked cells. The right figure shows that the marked cells can all be covered with just two monotone paths. ■

   ► Describe and analyze an efficient algorithm to compute the smallest set of monotone paths that covers every marked cell.

   Solution (Reduction to flow with lower bounds): Consider the following decision problem: Can the marked cells be covered using at most k monotone paths? If we can solve the decision problem efficiently, then we can find the minimum number of paths efficiently by performing binary search over the possible values of k.

   We construct a directed flow network G, with both capacities and lower bounds on the edges, as follows.

- G has two vertices $u_{i,j}$ and $v_{i,j}$ for each pair of array indices i and j, plus an additional source vertex s. The target vertex is $v_{n,n}$.
- Each vertex $u_{i,j}$ has an outgoing edge to its partner $v_{i,j}$. Each partner edge has lower bound 1 if $M[i,j] = \text{TRUE}$ and lower bound 0 otherwise; partner edges do not have capacities.
- Each vertex $v_{i,j}$ has outgoing edges to its right neighbor $u_{i+1,j}$ and its downward neighbor $u_{i,j+1}$ (if they exist). Neighbor edges have neither lower bounds nor capacities.
- Finally, there is an edge $(s, u_{1,1})$ with capacity k.

Each monotone path through the grid is represented by a directed path in G. Conversely, every directed path in G represents a monotone path through the grid. Thus, if we can cover the marked cells with k monotone paths, then there is a feasible flow through the network from s to $v_{n,n}$. Conversely, if the network admits an $(s, v_{n,n})$-flow with value k, since G is a DAG, that flow decomposes into k monotone paths from $(1,1)$ to (n,n), which must cover all the marked cells. So we can solve the decision problem by computing a maximum flow in G.

The network G has $O(n^2)$ vertices and edges, and we can construct it in $O(n^2)$ time by brute force. Computing a maximum flow in a network with both capacities and lower bounds requires $O(|V||E|) = O(n^4)$ time using Orlin's algorithm. Thus, the overall running time of this algorithm is $\boldsymbol{O(n^4 \log n)}$. ■

Remark. This solution can be rephrased using a *circulation* with lower bounds by replacing the edge $(s, u_{1,1})$ with an edge $(v_{n,n}, u_{1,1})$.

Solution (Reduction to min-cost flow; slower): We construct a directed flow network G where edges have lower bounds and costs, but no capacities.

- G has two vertices $u_{i,j}$ and $v_{i,j}$ for each pair of array indices i and j, plus an additional source vertex s. The target vertex is $v_{n,n}$.
- Each vertex $u_{i,j}$ has an edge to its partner $v_{i,j}$. Each partner edge has lower bound 1 if $M[i,j] = \text{TRUE}$ and lower bound 0 otherwise. Partner edges have cost 0.
- Each vertex $v_{i,j}$ has edges to its right neighbor $u_{i+1,j}$ and its downward neighbor $u_{i,j+1}$ (if they exist). Neighbor edges have lower bound 0 and cost 0.
- Finally, there is an edge $(s, u_{1,1})$ with cost 1 and lower bound 0.

Each monotone path through the grid is represented by a directed path in G. Conversely, every directed path in G represents a monotone path through the grid. Thus, if we can cover the marked cells with k monotone paths, then there is a feasible flow through the network from s to $v_{n,n}$. Conversely, if the network admits an $(s, v_{n,n})$-flow with value k, since G is a DAG that flow decomposes into k monotone paths from $(1,1)$ to (n,n), which must cover all the marked cells. So we can find the minimum number of monotone paths that cover all marked cells by computing a minimum-cost feasible flow in G.

The network G has $O(n^2)$ vertices and edges, and we can easily construct it in $O(n^2)$ time. Computing minimum-cost takes $O(|E|^2 \log^2 |V|) = \boldsymbol{O(n^4 \log^2 n)}$ time. ■

Rubric: Out of 10 points, we would allocate:

- 4 points for correct reduction
  - 1 for vertices
  - 1 for edges
  - 1 for capacities, lower bounds, and/or costs
  - 1 for remaining details
- 4 points for proof of correctness
  - 2 points for optimal flow $\Longrightarrow$ optimal solution
  - 2 points for optimal solution $\Longrightarrow$ optimal flow
- 2 points for time analysis
  - The time bound must be stated in terms of input parameters, not just parameters of the flow network.
  - No penalty for slower polynomial-time algorithms.
  - 1 point partial credit for ?polynomial time? but no explicit bound

2. Let $G = (V, E)$ be a graph with edge weights given by $c : E \to \mathbb{R}$. In the min-cost perfect matching problem the goal is to find a minimum cost perfect matching M in G (if there is one, otherwise the algorithm has to report that there is none) where the cost of a matching M is $\sum_{e \in M} c(e)$; note that the costs can be negative. In the maximum weight perfect matching problem the input is a graph $G = (V, E)$ and *non-negative* weights $w : E \to \mathbb{R}_+$ and the goal is to find a matching M of maximum weight. Note that a maximum weight matching may not be a maximum cardinality matching. Describe an efficient reduction from min-cost perfect matching to max-weight matching.

Solution: Let n be the number of vertices in graph G. If n is odd, then there trivially no perfect matching, so we will assume n is even. For each edge $e \in E$, let $\hat{c}(e) = c_{max} + 1 - c(e)$, where $c_{max} = \max_{e \in E} c(e)$ is the maximum capacity of an edge in the graph, and let $\Delta = \sum_{e \in E} \hat{c}(e)$. We construct a graph G' with same set of vertices and edges as G. For each edge $e \in G'$, set its weight to be $c'(e) = \hat{c}(e) + \Delta$. Let M be the maximum weight matching in graph G'. If $|M| = n/2$, then M is a min cost perfect matching in G, else G has no perfect matching.

Clearly we can construct the graph G' is polynomial time. So we now have to prove that it is a valid reduction, i.e., M is a min-cost perfect matching in $G \iff M$ is a max-weight perfect matching in G'.

Proof of Correctness. We can immediately make some observations about our constructed graph G'. Since, G and G' have the same set of vertices and edges, G has a perfect matching iff G' has a perfect matching. For each edge $e \in G$, $\hat{c}(e) \geq 1$. Furthermore, for each edge $e \in G'$, $c'(e)$ is non-negative. The remaining proof follows from the following two claims.

Claim 1. *If there exists a perfect matching in G', then the max weight matching in G' is also a max weight perfect matching.*

Proof: Let M be the maximum weight perfect matching in G'. Let M' be a non-perfect matching. The weight of M' is

$$\sum_{e \in M'} c'(e) = \sum_{e \in M'} (\hat{c}(e) + \Delta) < \sum_{e \in E} \hat{c}(e) + |M'| \cdot \Delta = (|M'| + 1) \cdot \Delta \leq |M| \cdot \Delta < \sum_{e \in M} c'(e)$$

where the first inequality follows from $M' \subset E$, and $\hat{c}(e) \geq 1$. So M' cannot be a max weight matching. □

Claim 2. *M is a min cost perfect matching in $G \iff M$ is a max weight perfect matching in G'.*

Proof:

($\implies$:) Suppose M is not a max weight perfect matching, then there exists a perfect matching M' in G' which has strictly larger weight. Then

$$\sum_{e \in M'} c'(e) > \sum_{e \in M} c'(e) \implies \sum_{e \in M'} (c_{max} + 1 + \Delta - c(e)) > \sum_{e \in M} (c_{max} + 1 + \Delta - c(e))$$
$$\implies \sum_{e \in M'} c(e) < \sum_{e \in M} c(e)$$

so M is not a min-cost perfect matching in G.

($\impliedby$:) Suppose M is not a min cost perfect matching, then there exists a perfect matching M' in G which has strictly smaller cost. Then

$$\sum_{e \in M'} c(e) < \sum_{e \in M} c(e) \implies \sum_{e \in M'} (c_{max} + 1 + \Delta - c'(e)) < \sum_{e \in M} (c_{max} + 1 + \Delta - c'(e))$$
$$\implies \sum_{e \in M'} c'(e) > \sum_{e \in M} c'(e)$$

so M is not a max weight perfect matching in G'. □

■

Rubric: Out of 10 points, we would allocate:

- Atmost 6 points, if they reduce min-cost PM to max-weight perfect matching.
  - 2 points for construction
  - 4 points for proof (2 points each direction)
- Full 10 points, if they reduce min-cost PM to max-weight matching.
  - 4 points for construction
  - 2 points for showing that max weight matching will be perfect matching.
  - 4 points for remaining proof (2 points each direction)

3. Consider a polyhedron P in n dimensions defined by a set of m inequalities $Ax \leq b$. Let $z \in \mathbb{R}^n$ be a point. Describe a polynomial-time algorithm to check whether z is a basic feasible solution of P.

Solution: To check that z is feasible, we compute $y = Az$, and check that $y_i \leq b_i$ for $1 \leq i \leq m$.

Next, to check that z is basic, ~~ask if z wants a pumpkin spice latte~~ let $\bar{A}x \leq \bar{b}$ be the set of inequalities that z satisfied with equality. z is basic if and only if there are at least n linearly independent inequalities in the set. "Gaussian"[1] elimination gives us the rank of the augmented matrix $[\bar{A} | \bar{b}]$, which is also the number of inequalities in the set that are linearly independent.

Computing $y = Az$ takes $O(mn)$ time using standard matrix multiplication, checking $y \leq b$ takes $O(m)$ time, and elimination on $[\bar{A} | \bar{b}]$ takes $O(m^3)$ time[2]. So the overall running time is $\boldsymbol{O(mn + m^3)}$. ■

Rubric: Out of 10 points, we would allocate:
- 3 points for checking feasibility
- 5 points for checking basicness
- 2 points for runtime analysis

[1] Gauss actually called this procedure "ordinary", and Legendre called it "common". For an explanation of how it came to be named after Gauss, see https://arxiv.org/abs/0907.2397.

[2] ...if you assume arbitrary precision arithmetic in $O(1)$ time. Although the number of arithmetic operations used is $O(n^3)$, the actual running time is more subtle. See http://cstheory.stackexchange.com/q/3921/16739 for more details.