Complexity Theory

Part II

Time Complexity

- The **time complexity** of a TM M is a function denoting the *worst-case* number of steps M takes on any input of length n.
  - By convention, n denotes the length of the input.
  - Assume we're only dealing with deciders, so there's no need to handle looping TMs.
- We often use **big-O notation** to describe growth rates of functions (and time complexity in particular).
  - Found by discarding leading coefficients and low-order terms.

Polynomials and Exponentials

- A TM runs in **polynomial time** iff its runtime is some polynomial in n.
  - That is, time $O(n^k)$ for some constant k.
- Polynomial functions "scale well."
  - Small changes to the size of the input do not typically induce enormous changes to the overall runtime.
- Exponential functions scale terribly.
  - Small changes to the size of the input induce huge changes in the overall runtime.

The Cobham-Edmonds Thesis

A language L can be **decided efficiently** iff there is a TM that decides it in polynomial time.

Equivalently, L can be decided efficiently iff it can be decided in time $O(n^k)$ for some $k \in \mathbb{N}$.

Like the Church-Turing thesis, this is ***not*** a theorem!

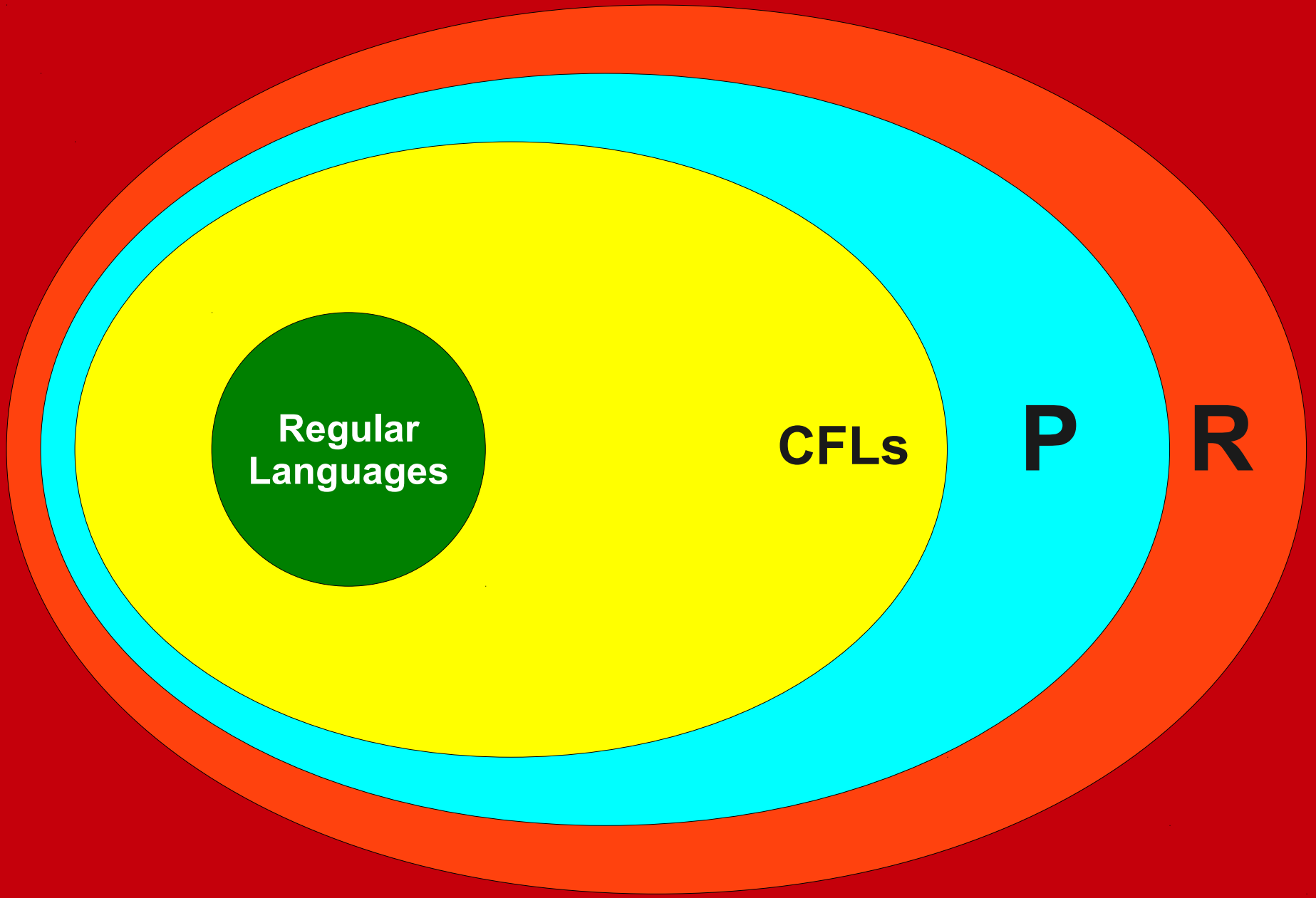
It's an assumption about the nature of efficient computation, and it is somewhat controversial.

The Complexity Class **P**

- The **complexity class P** (for **p**olynomial time) contains all problems that can be solved in polynomial time.
- Formally:

$$\mathbf{P} = \{ L \mid \text{There is a polynomial-time decider for } L \}$$

- Assuming the Cobham-Edmonds thesis, a language is in **P** iff it can be decided efficiently.


Regular Languages
CFLs
P
R
Undecidable Languages

Problems in **P**

- **Graph connectivity:**

  Given a graph G and nodes s and t, is there a path from s to t?

- **Primality testing:**

  Given a number p, is p prime? (Best known TM for this takes time $O(n^{72})$.)

- **Maximum matching:**

  Given a set of tasks and workers who can perform those tasks, can all of the tasks be completed in under n hours?

Problems in **P**

- **Remoteness testing:**

  Given a graph G, are all of the nodes in G within distance at most k of one another?

- **Linear programming:**

  Given a linear set of constraints and linear objective function, is the optimal solution at least n?

- **Edit distance:**

  Given two strings, can the strings be transformed into one another in at most n single-character edits?

Other Models of Computation

- **Theorem**: $L \in \mathbf{P}$ iff there is a polynomial-time TM or computer program that decides it.
- Essentially – a problem is in **P** iff you could solve it on a normal computer in polynomial time.
- Proof involves simulating a computer with a TM; come talk to me after lecture for details on how to do this.

Proving Languages are in **P**

- **Directly prove the language is in P**.
  - Build a decider for the language L.
  - Prove that the decider runs in time $O(n^k)$.
- **Use closure properties**.
  - Prove that the language can be formed by appropriate transformations of languages in **P**.
- **Reduce the language to a language in P**.
  - Show how a polynomial-time decider for some language L' can be used to decide L.

Proving Languages are in **P**

Directly prove the language is in P.

Build a decider for the language L.

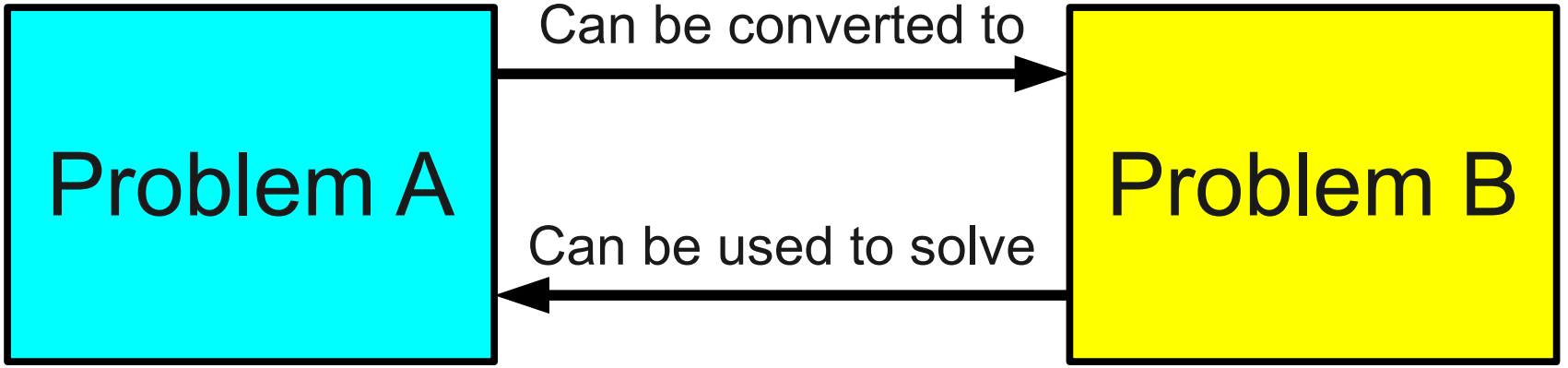
Prove that the decider runs in time $O(n^k)$.

Use closure properties.

Prove that the language can be formed by appropriate transformations of languages in **P**.

- **Reduce the language to a language in P.**
  - Show how a polynomial-time decider for some language L' can be used to decide L.

Reductions



If any instance of A can be converted into an instance of B, we say that A **reduces** to B.

Mapping Reductions and **P**

- When studying whether problems were in **R**, **RE**, or co-**RE**, we used mapping reductions.
- The construction we built using mapping reductions
  - computes the function f on some input string w, then
  - runs another TM on $f(w)$.
- When talking about class **P**, we need to make sure that this entire process doesn't take too much time.

Polynomial-Time Reductions

- Let $A \subseteq \Sigma_1^*$ and $B \subseteq \Sigma_2^*$ be languages.
- A **polynomial-time mapping reduction** is a function $f : \Sigma_1^* \to \Sigma_2^*$ with the following properties:
  - $f(w)$ can be computed **in polynomial time**.
  - $w \in A$ iff $f(w) \in B$.
- Informally:
  - A way of turning inputs to A into inputs to B
  - that can be computed **in polynomial time**
  - that preserves the correct answer.
- Notation: $\boldsymbol{A \leq_P B}$ iff there is a polynomial-time mapping reduction from A to B.

Polynomial-Time Reductions

- Suppose that we know that $B \in \mathbf{P}$.
- Suppose that $A \leq_P B$ and that the reduction f can be computed in time $O(n^k)$.

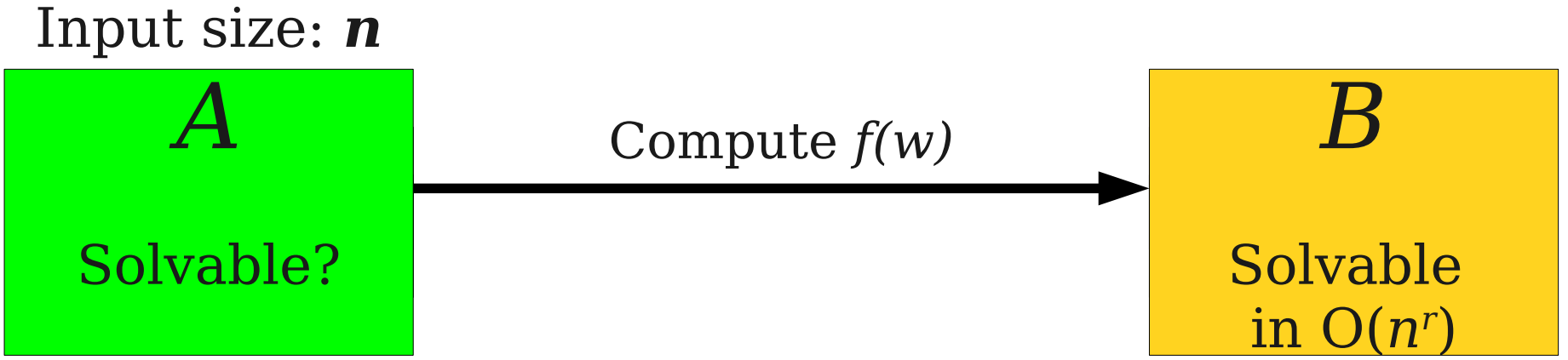
Polynomial-Time Reductions

- Suppose that we know that $B \in \mathbf{P}$.
- Suppose that $A \leq_P B$ and that the reduction f can be computed in time $O(n^k)$.

Input size: ***n***

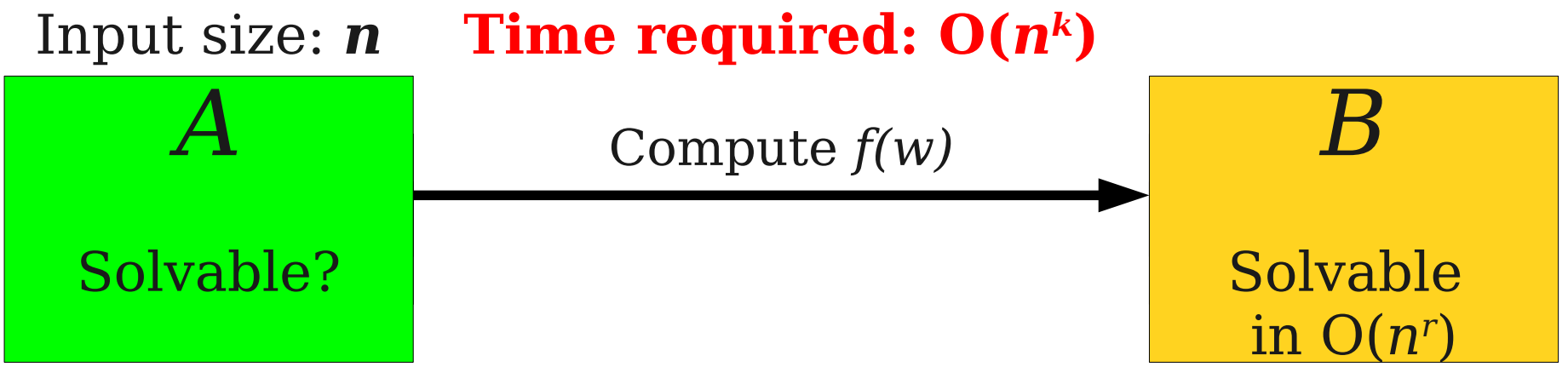
Polynomial-Time Reductions

- Suppose that we know that $B \in \mathbf{P}$.
- Suppose that $A \leq_P B$ and that the reduction f can be computed in time $O(n^k)$.



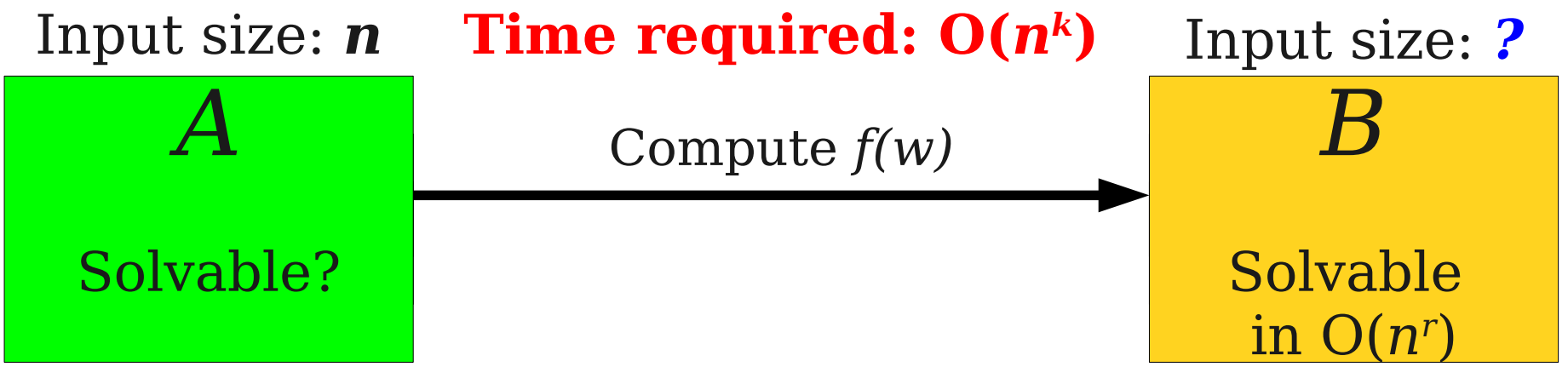
Polynomial-Time Reductions

- Suppose that we know that $B \in \mathbf{P}$.
- Suppose that $A \leq_P B$ and that the reduction f can be computed in time $O(n^k)$.



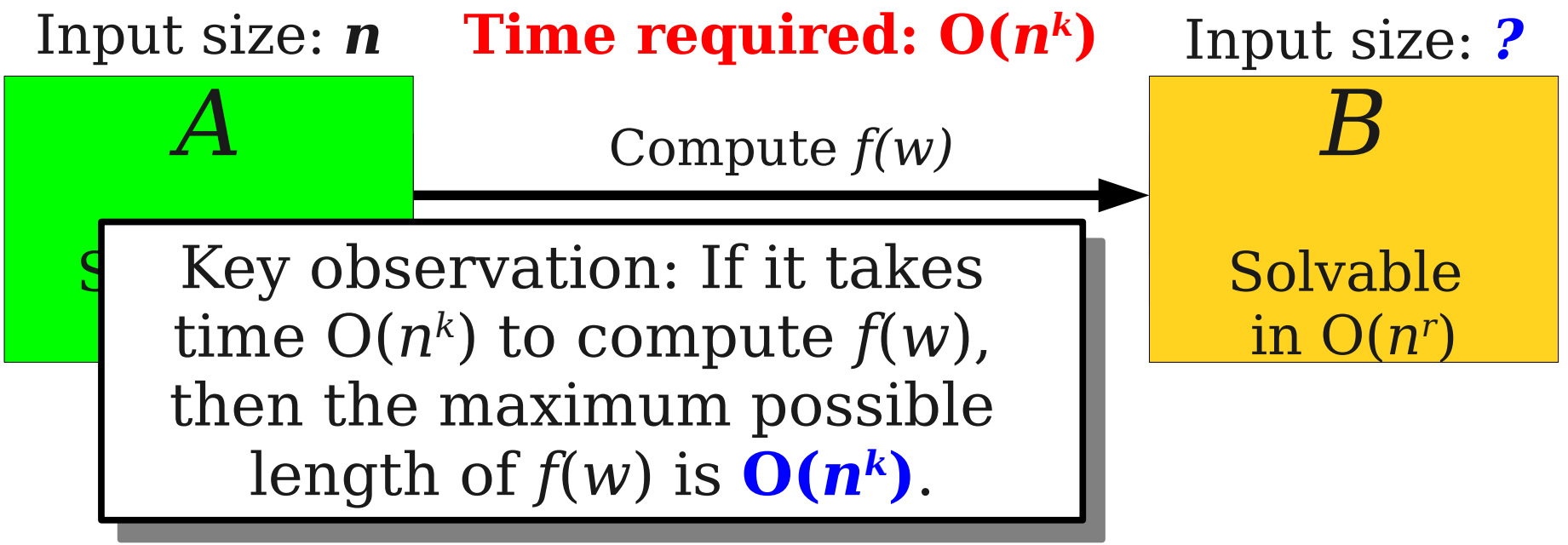
Polynomial-Time Reductions

- Suppose that we know that $B \in \mathbf{P}$.
- Suppose that $A \leq_P B$ and that the reduction f can be computed in time $O(n^k)$.



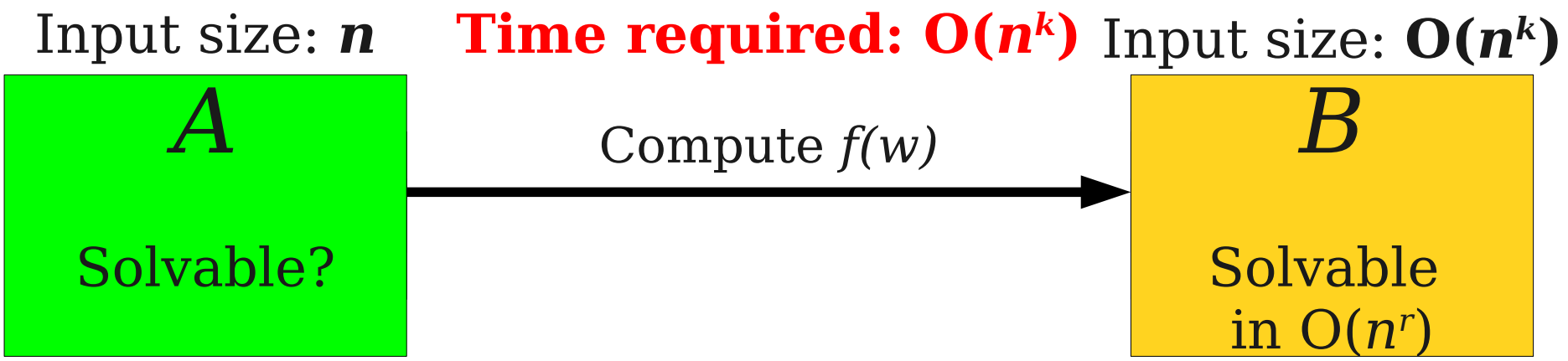
Polynomial-Time Reductions

- Suppose that we know that $B \in \mathbf{P}$.
- Suppose that $A \leq_P B$ and that the reduction f can be computed in time $O(n^k)$.



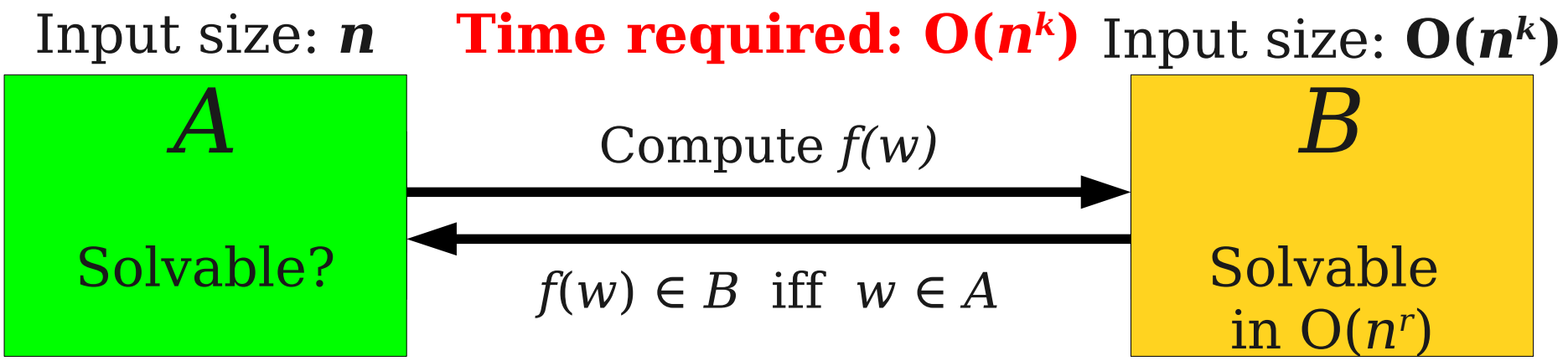
Polynomial-Time Reductions

- Suppose that we know that $B \in \mathbf{P}$.
- Suppose that $A \leq_P B$ and that the reduction f can be computed in time $O(n^k)$.



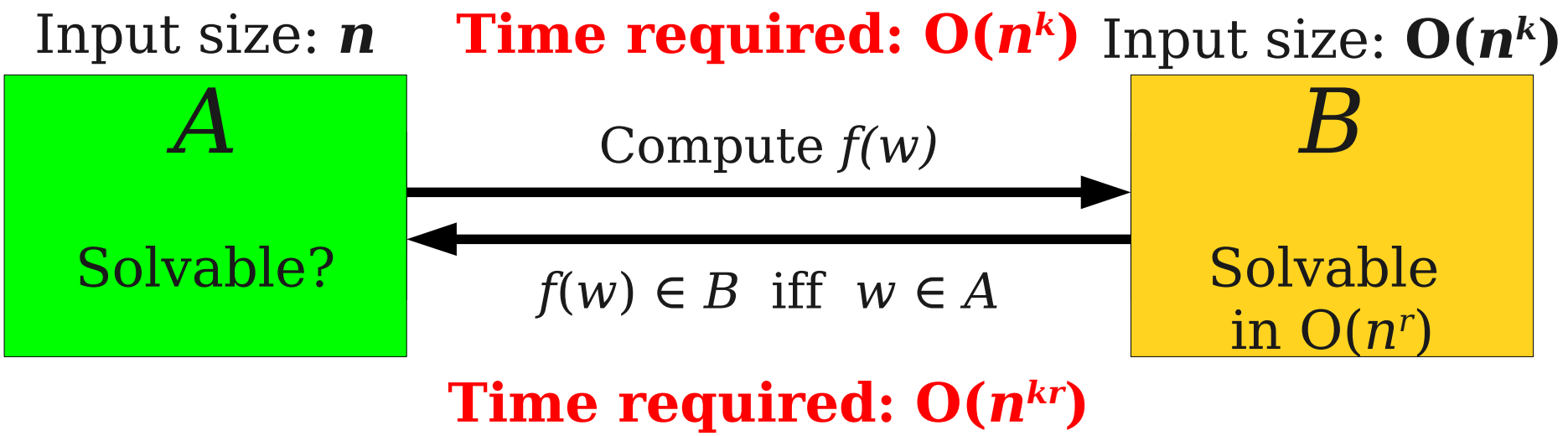
Polynomial-Time Reductions

- Suppose that we know that $B \in \mathbf{P}$.
- Suppose that $A \leq_P B$ and that the reduction f can be computed in time $O(n^k)$.



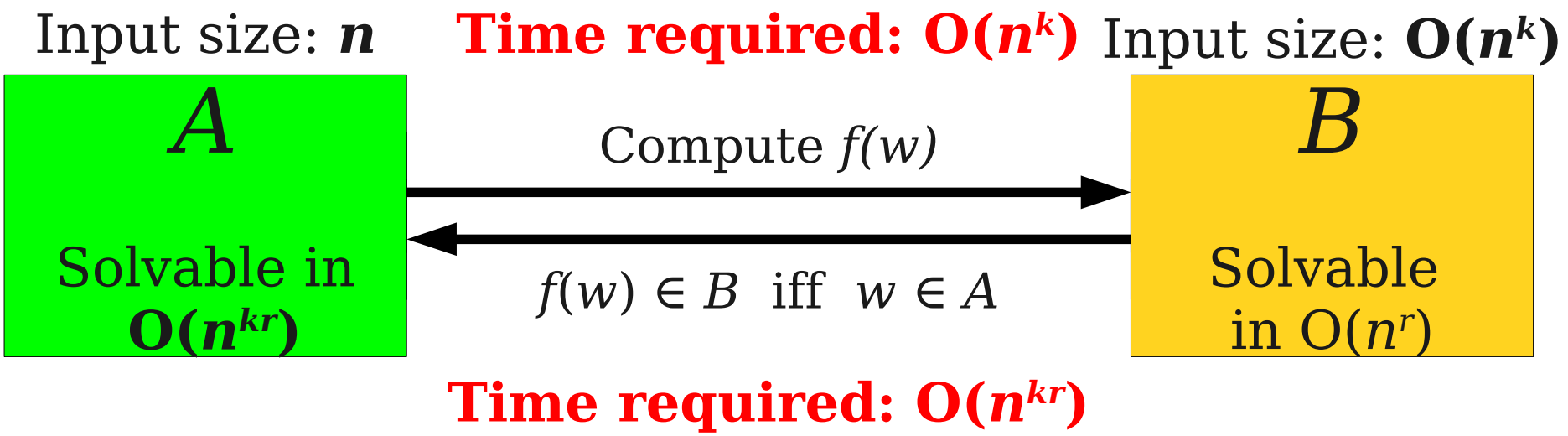
Polynomial-Time Reductions

- Suppose that we know that $B \in \mathbf{P}$.
- Suppose that $A \leq_P B$ and that the reduction f can be computed in time $O(n^k)$.



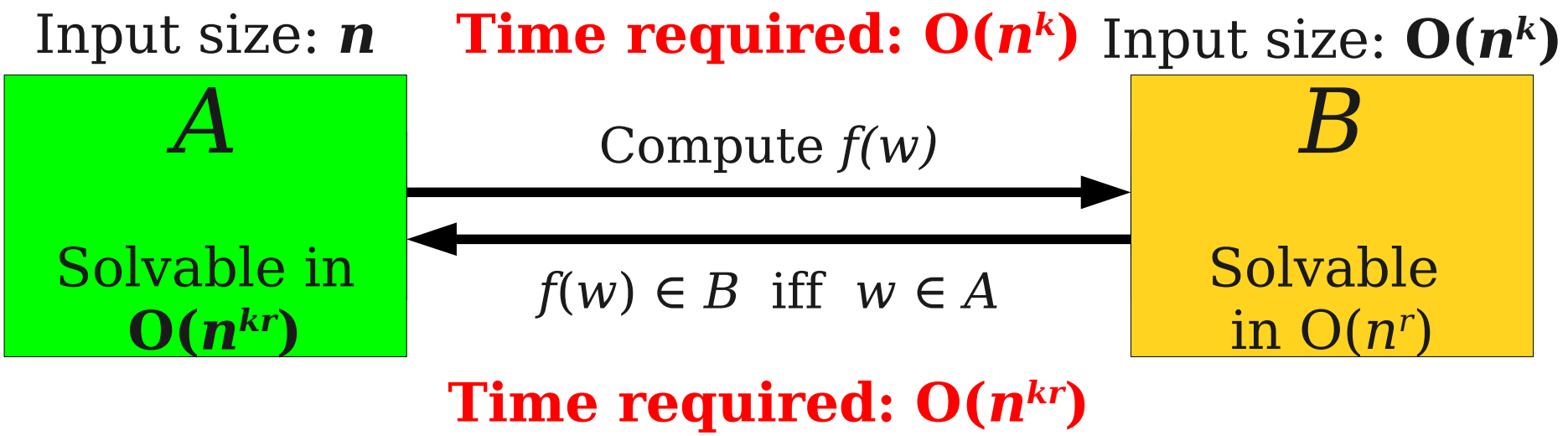
Polynomial-Time Reductions

- Suppose that we know that $B \in \mathbf{P}$.
- Suppose that $A \leq_P B$ and that the reduction f can be computed in time $O(n^k)$.



Polynomial-Time Reductions

- Suppose that we know that $B \in \mathbf{P}$.
- Suppose that $A \leq_P B$ and that the reduction f can be computed in time $O(n^k)$.
- Then $A \in \mathbf{P}$ as well.



Theorem: If $B \in \mathbf{P}$ and $A \leq_P B$, then $A \in \mathbf{P}$.

Proof: Let H be a polynomial-time decider for B. Consider the following TM:

> M = "On input w:
> Compute $f(w)$.
> Run H on $f(w)$.
> If H accepts, accept; if H rejects, reject."

We claim that M is a polynomial-time decider for A. To see this, we prove that M is a polynomial-time decider, then that $\mathscr{L}(M) = A$. To see that M is a polynomial-time decider, note that because f is a polynomial-time reduction, computing $f(w)$ takes time $O(n^k)$ for some k. Moreover, because computing $f(w)$ takes time $O(n^k)$, we know that $|f(w)| = O(n^k)$. M then runs H on $f(w)$. Since H is a polynomial-time decider, H halts in $O(m^r)$ on an input of size m for some r. Since $|f(w)| = O(n^k)$, H halts after $O(|f(w)|^r) = O(n^{kr})$ steps. Thus M halts after $O(n^k + n^{kr})$ steps, so M is a polynomial-time decider.

To see that $\mathscr{L}(M) = A$, note that M accepts w iff H accepts $f(w)$ iff $f(w) \in B$. Since f is a polynomial-time reduction, $f(w) \in B$ iff $w \in A$. Thus M accepts w iff $w \in A$, so $\mathscr{L}(M) = A$. ■

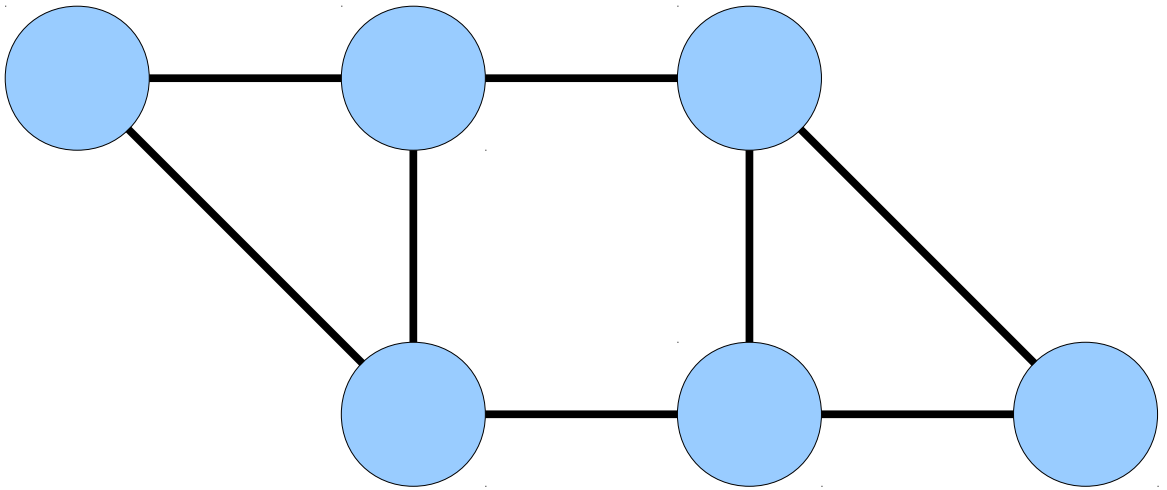
A Sample Reduction

Maximum Matching

- Given an undirected graph G, a **matching** in G is a set of edges such that no two edges share an endpoint.
- A **maximum matching** is a matching with the largest number of edges.

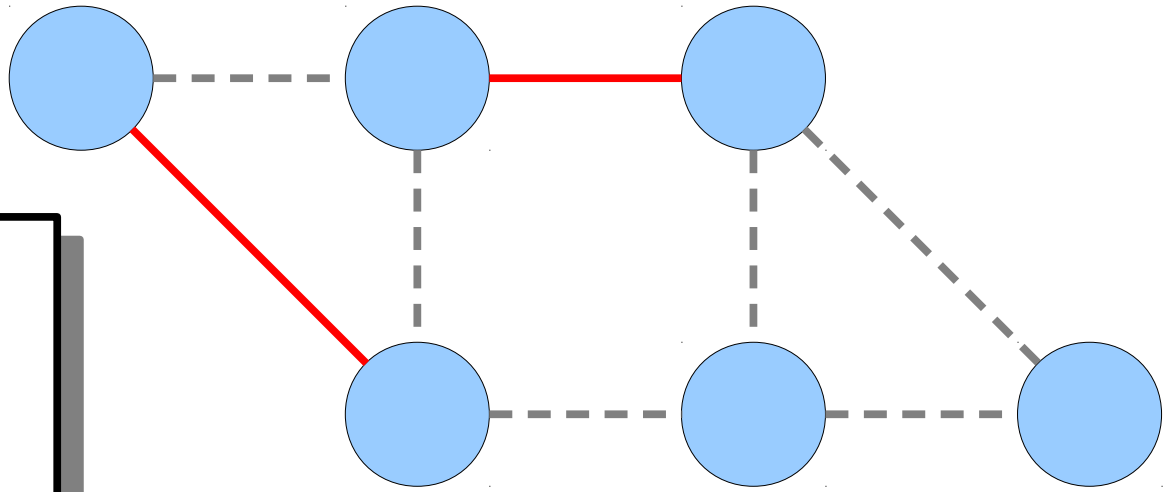
Maximum Matching

- Given an undirected graph G, a **matching** in G is a set of edges such that no two edges share an endpoint.
- A **maximum matching** is a matching with the largest number of edges.



Maximum Matching

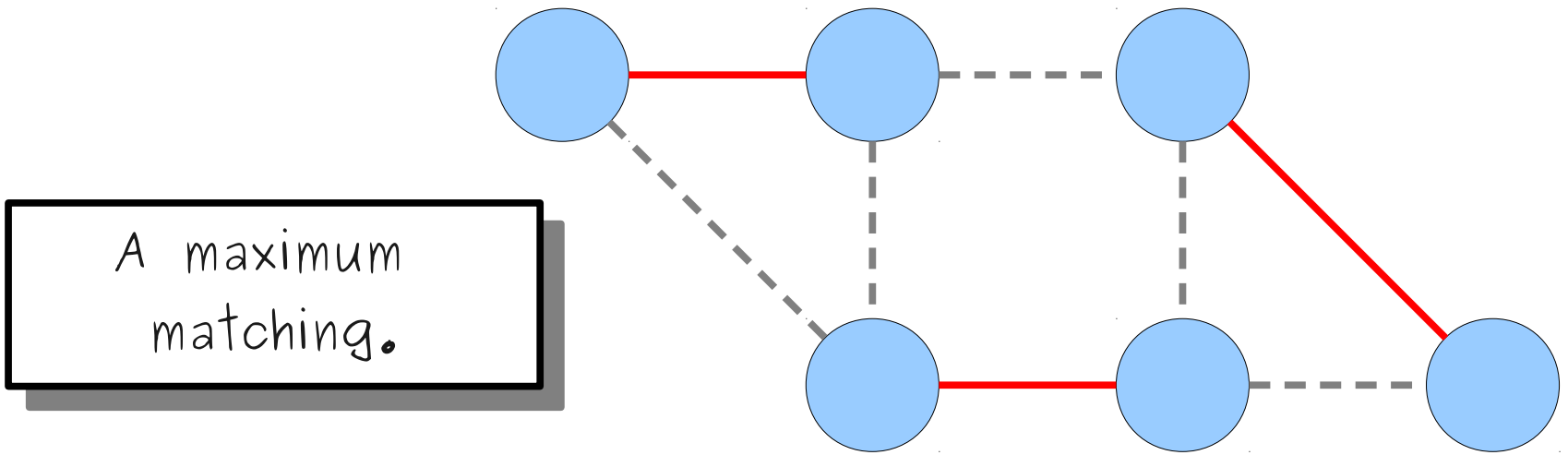
- Given an undirected graph G, a **matching** in G is a set of edges such that no two edges share an endpoint.
- A **maximum matching** is a matching with the largest number of edges.



A matching, but not a maximum matching.

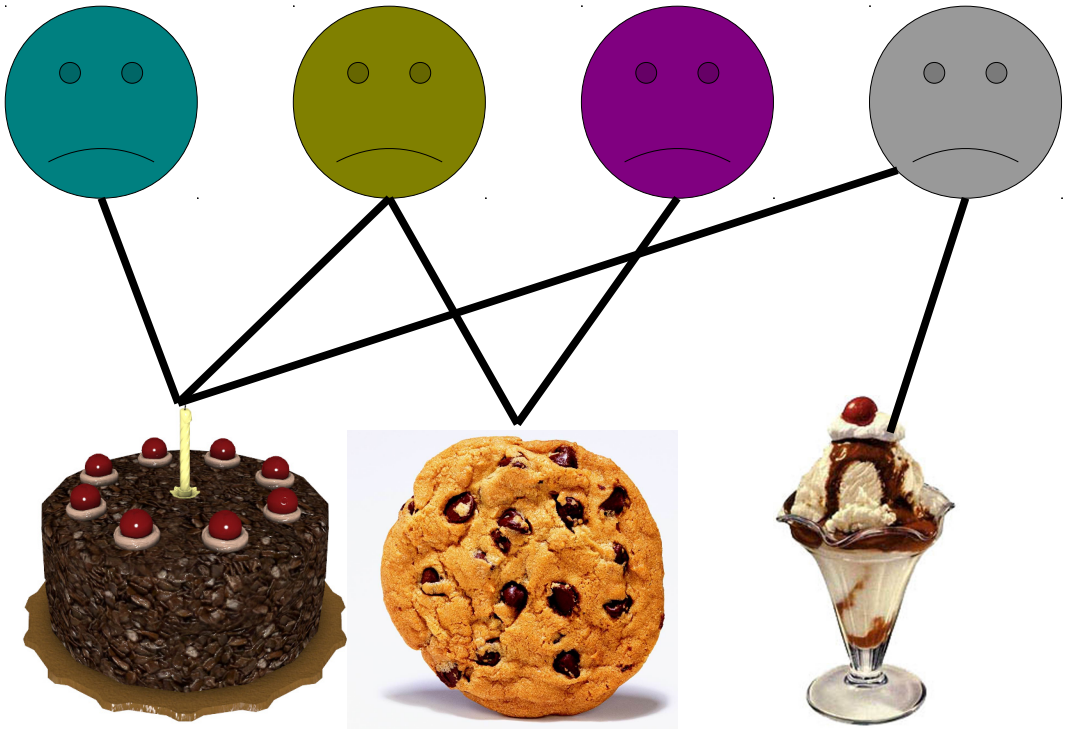
Maximum Matching

- Given an undirected graph G, a **matching** in G is a set of edges such that no two edges share an endpoint.
- A **maximum matching** is a matching with the largest number of edges.



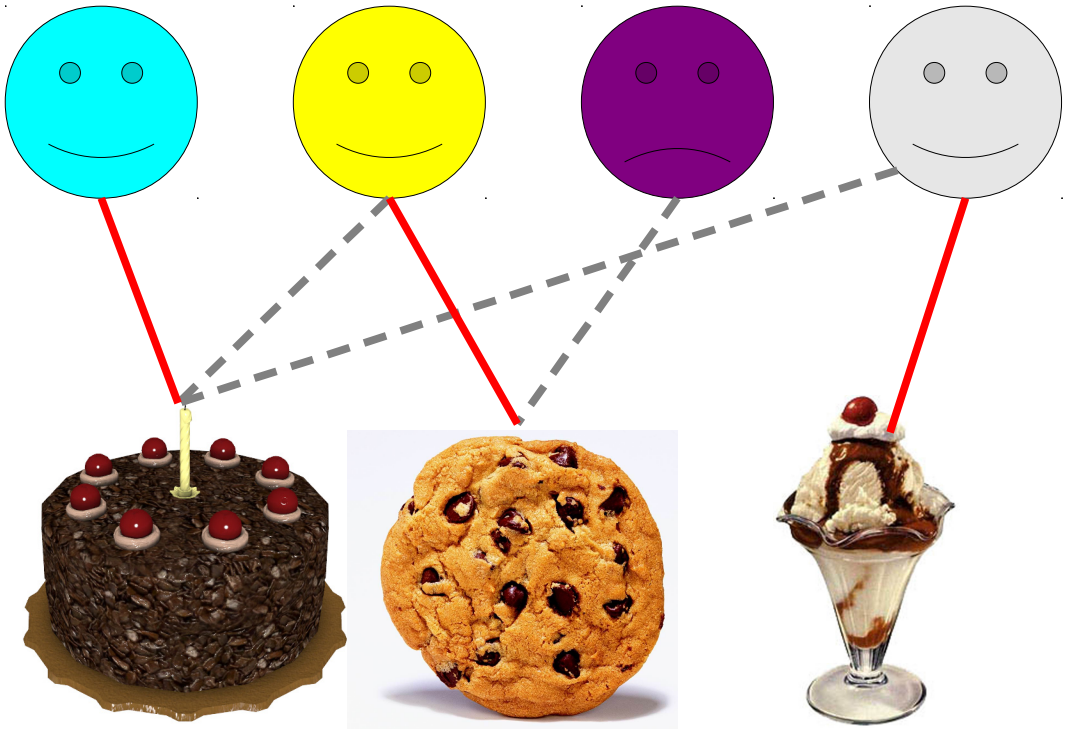
Maximum Matching

- Given an undirected graph G, a **matching** in G is a set of edges such that no two edges share an endpoint.
- A **maximum matching** is a matching with the largest number of edges.



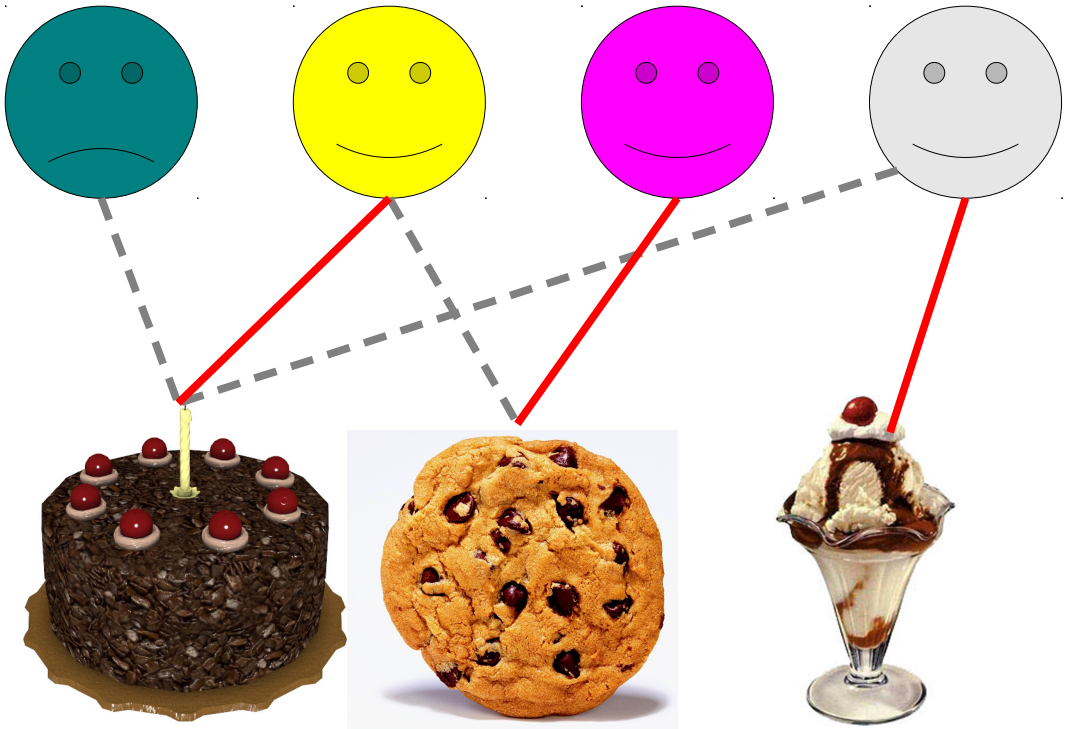
Maximum Matching

- Given an undirected graph G, a **matching** in G is a set of edges such that no two edges share an endpoint.
- A **maximum matching** is a matching with the largest number of edges.



Maximum Matching

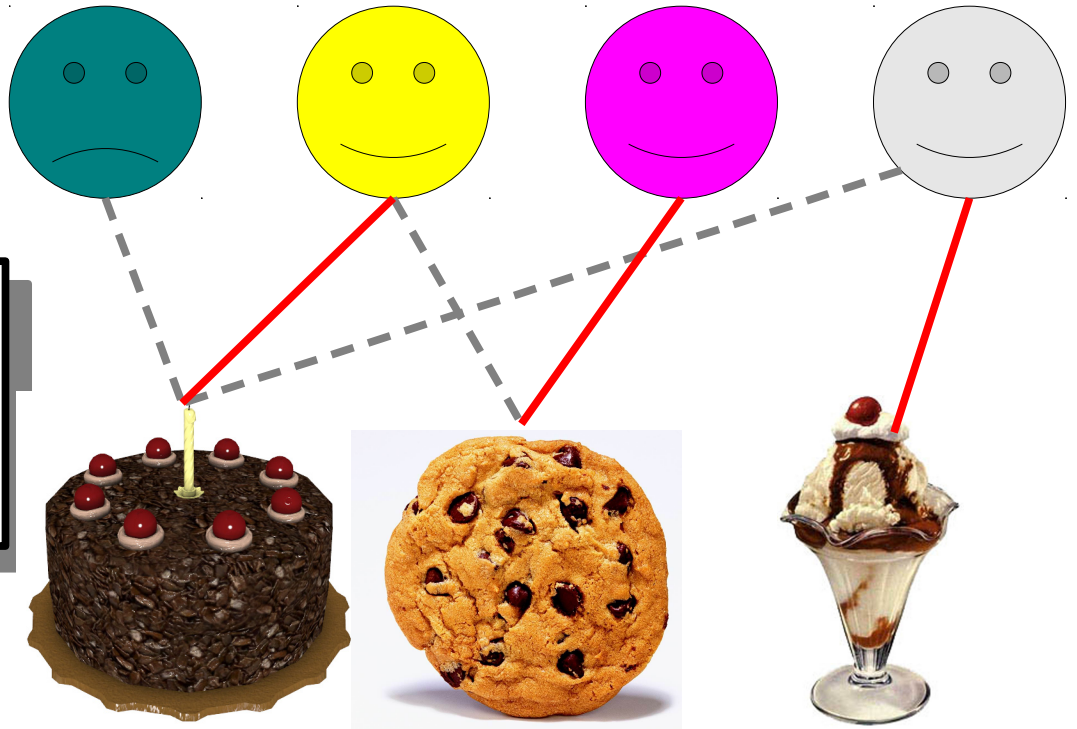
- Given an undirected graph G, a **matching** in G is a set of edges such that no two edges share an endpoint.
- A **maximum matching** is a matching with the largest number of edges.



Maximum Matching

- Given an undirected graph G, a **matching** in G is a set of edges such that no two edges share an endpoint.
- A **maximum matching** is a matching with the largest number of edges.

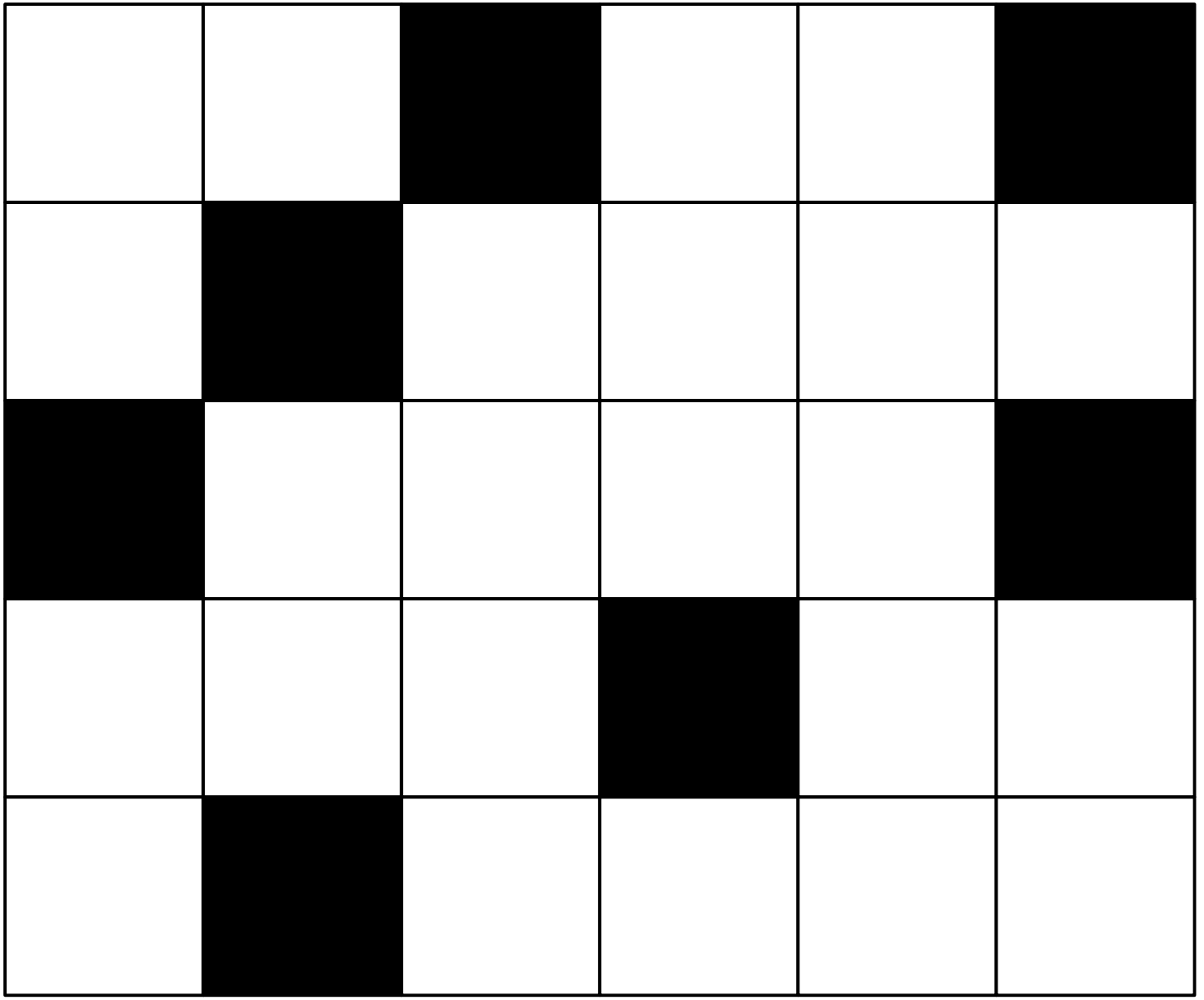
Maximum matchings are not necessarily unique.



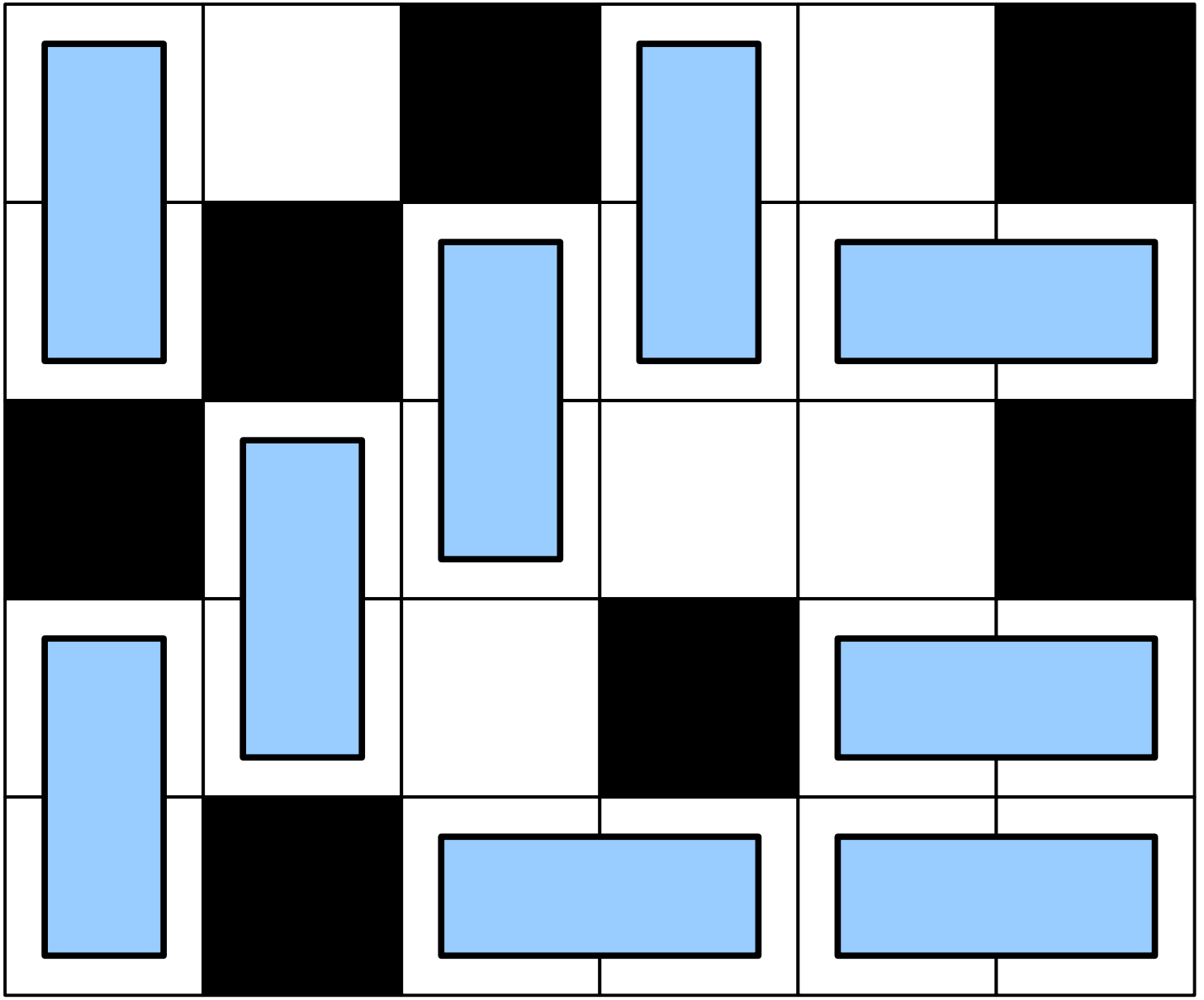
Maximum Matching

- Jack Edmonds' paper “Paths, Trees, and Flowers” gives a **polynomial-time algorithm** for finding maximum matchings.
  - (This is the same Edmonds as in “Cobham-Edmonds Thesis.)
- Using this fact, what other problems can we solve?

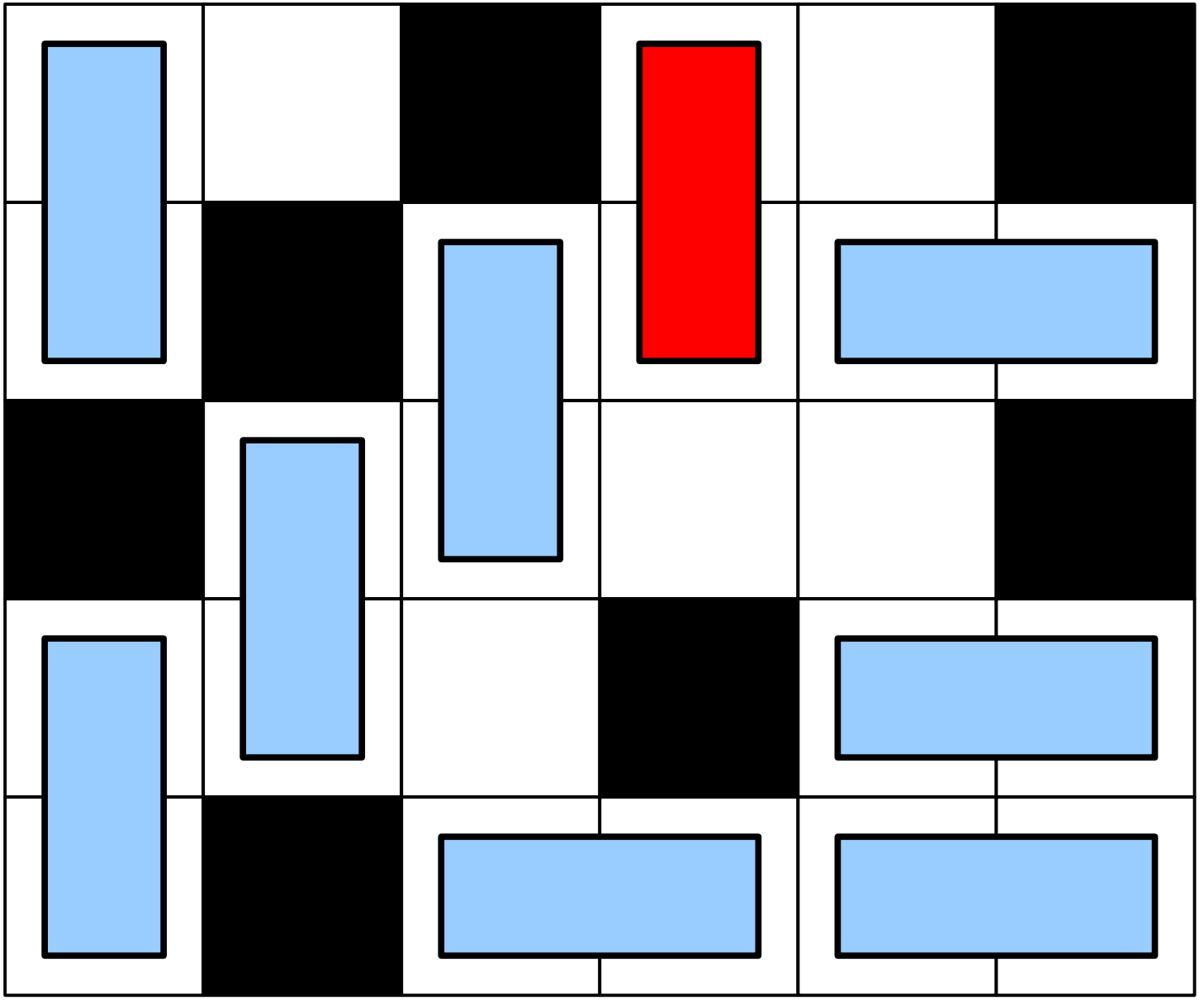
Domino Tiling



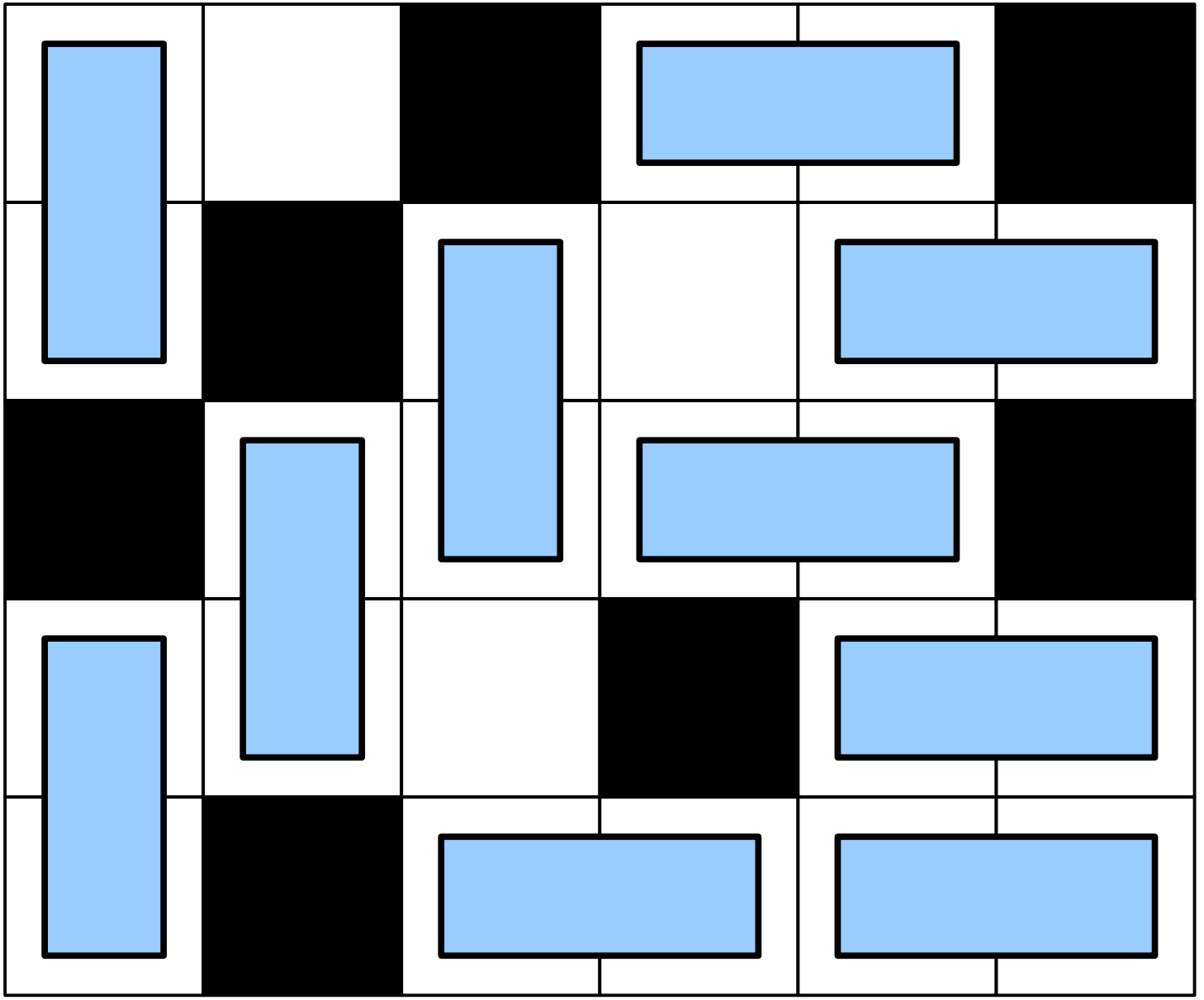
Domino Tiling



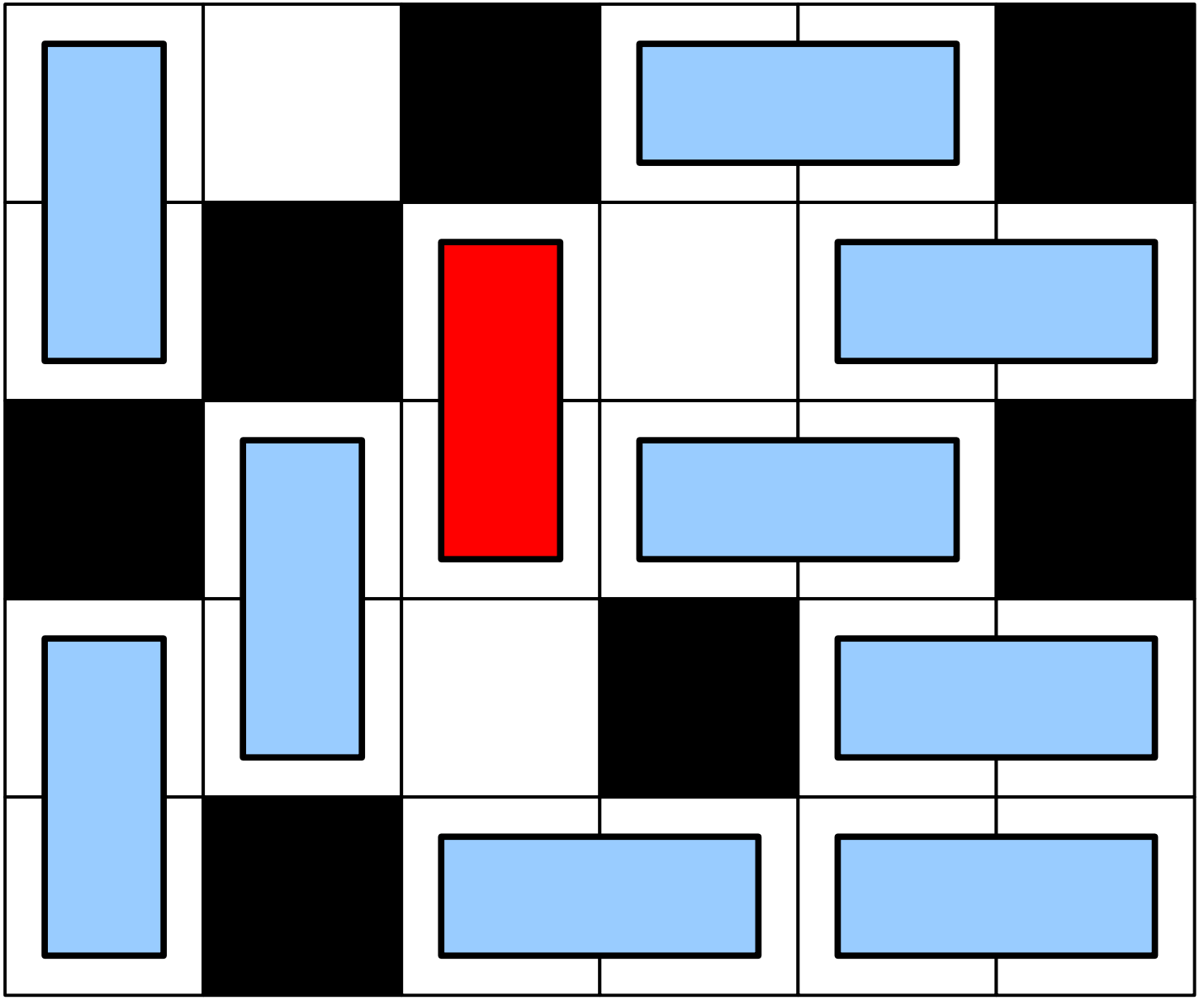
Domino Tiling



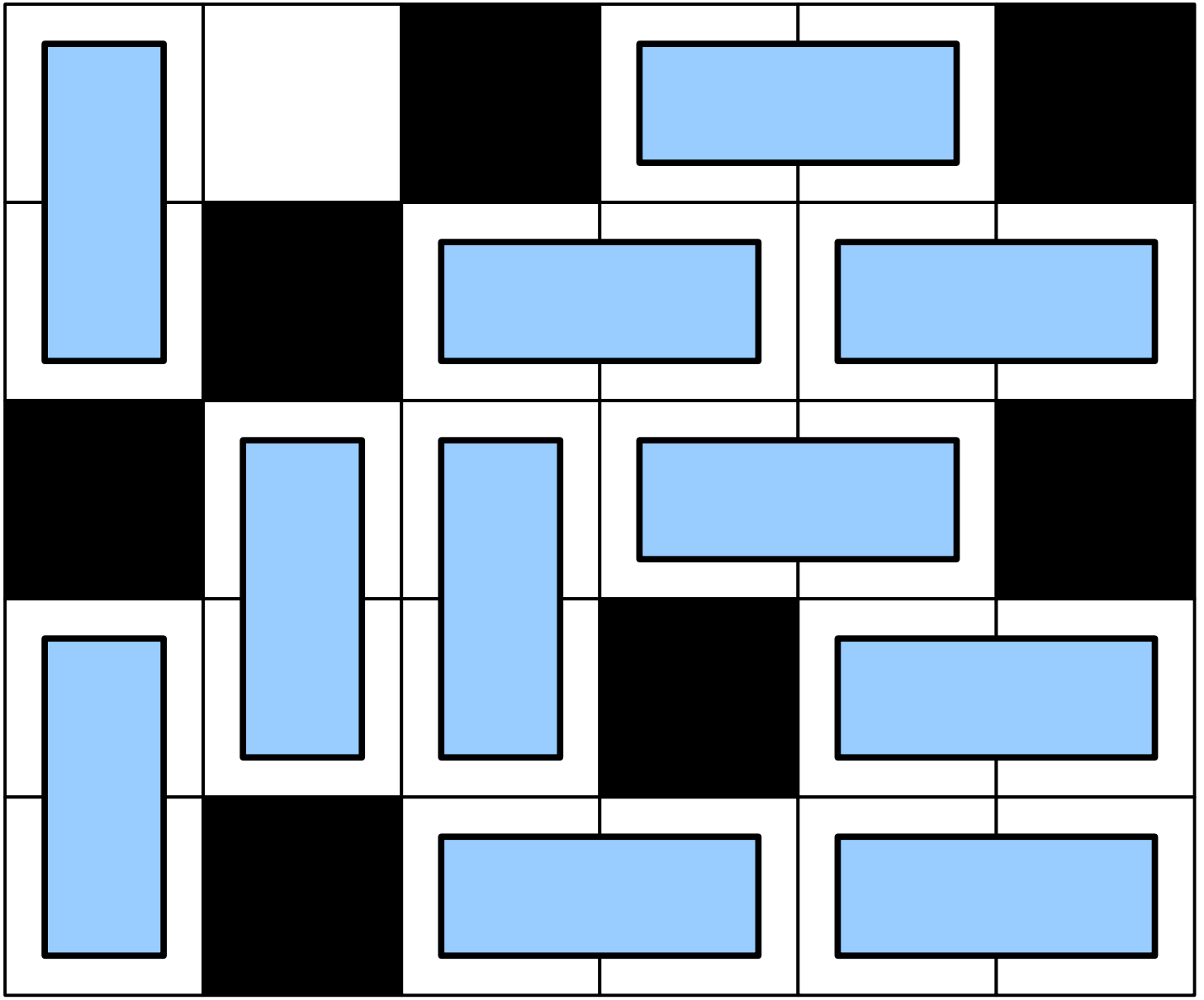
Domino Tiling



Domino Tiling



Domino Tiling



A Domino Tiling Reduction

- Let *MATCHING* be the language defined as follows:

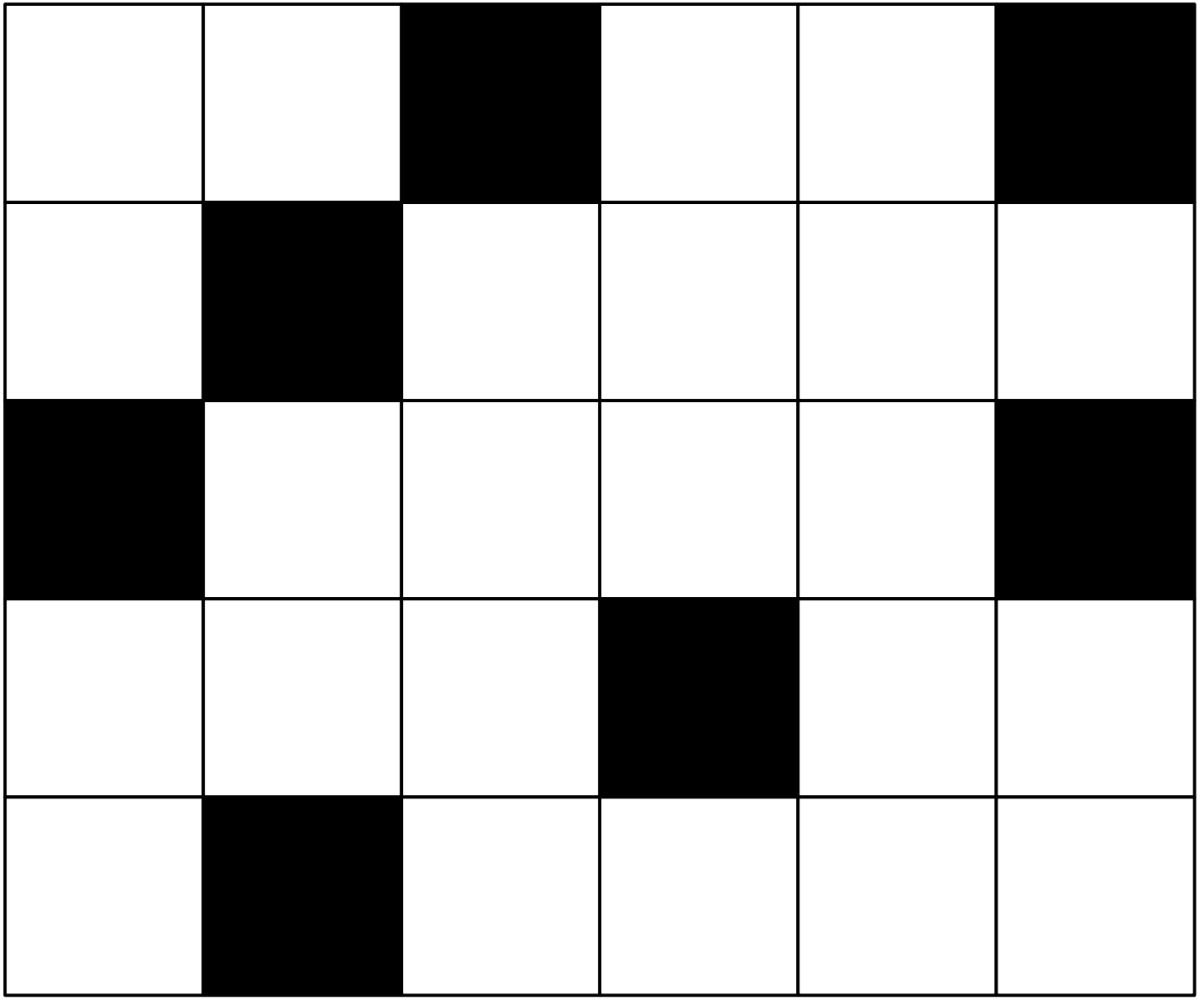
  $MATCHING = \{ \langle G, k \rangle \mid G$ is an undirected graph with a matching of size at least $k \}$

- **Theorem** (Edmonds): $MATCHING \in \mathbf{P}$.
- Let *DOMINO* be this language:

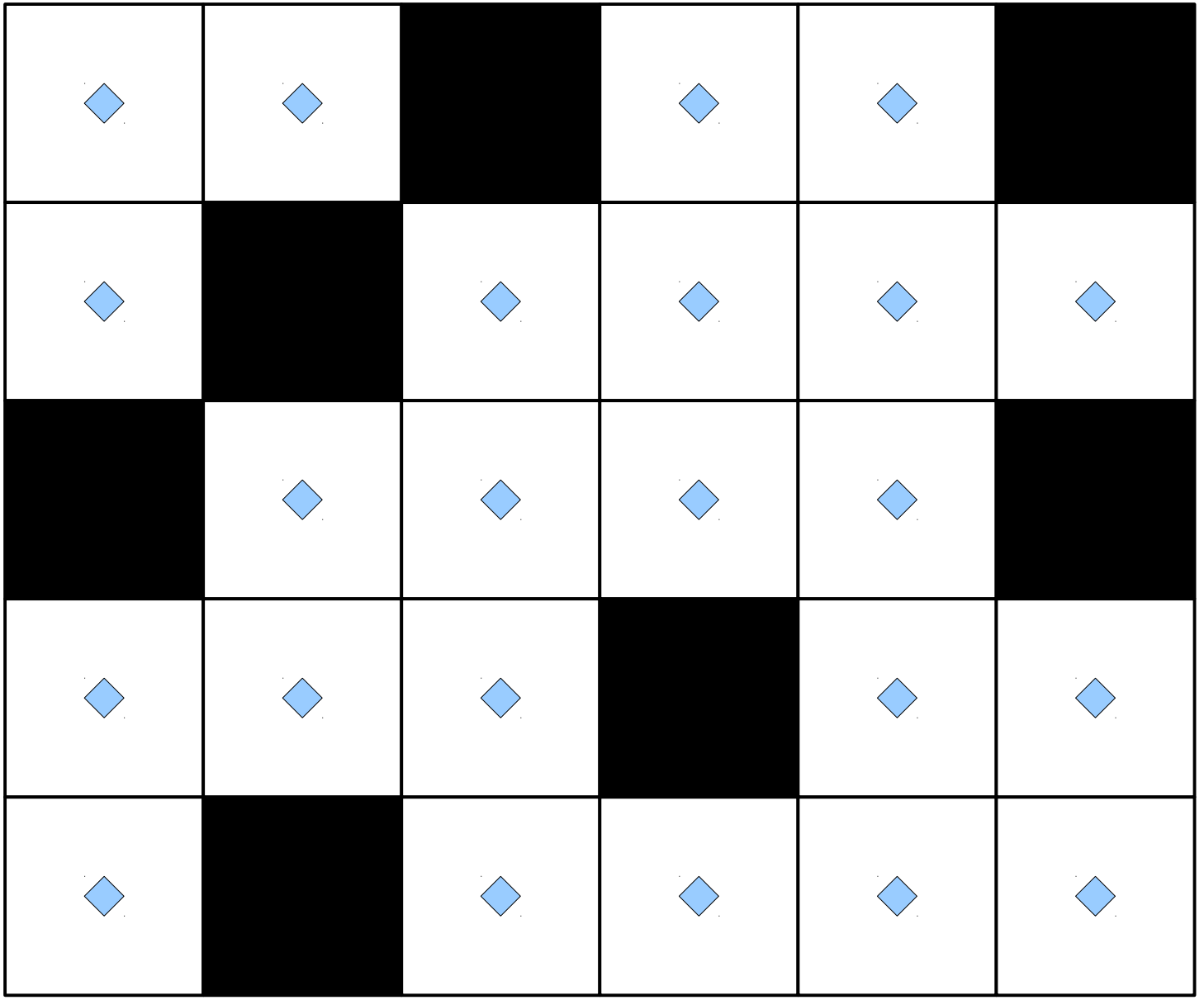
  $DOMINO = \{ \langle D, k \rangle \mid D$ is a grid and k nonoverlapping dominoes can be placed on D. $\}$

- We'll prove $DOMINO \leq_P MATCHING$ to show that $DOMINO \in \mathbf{P}$.

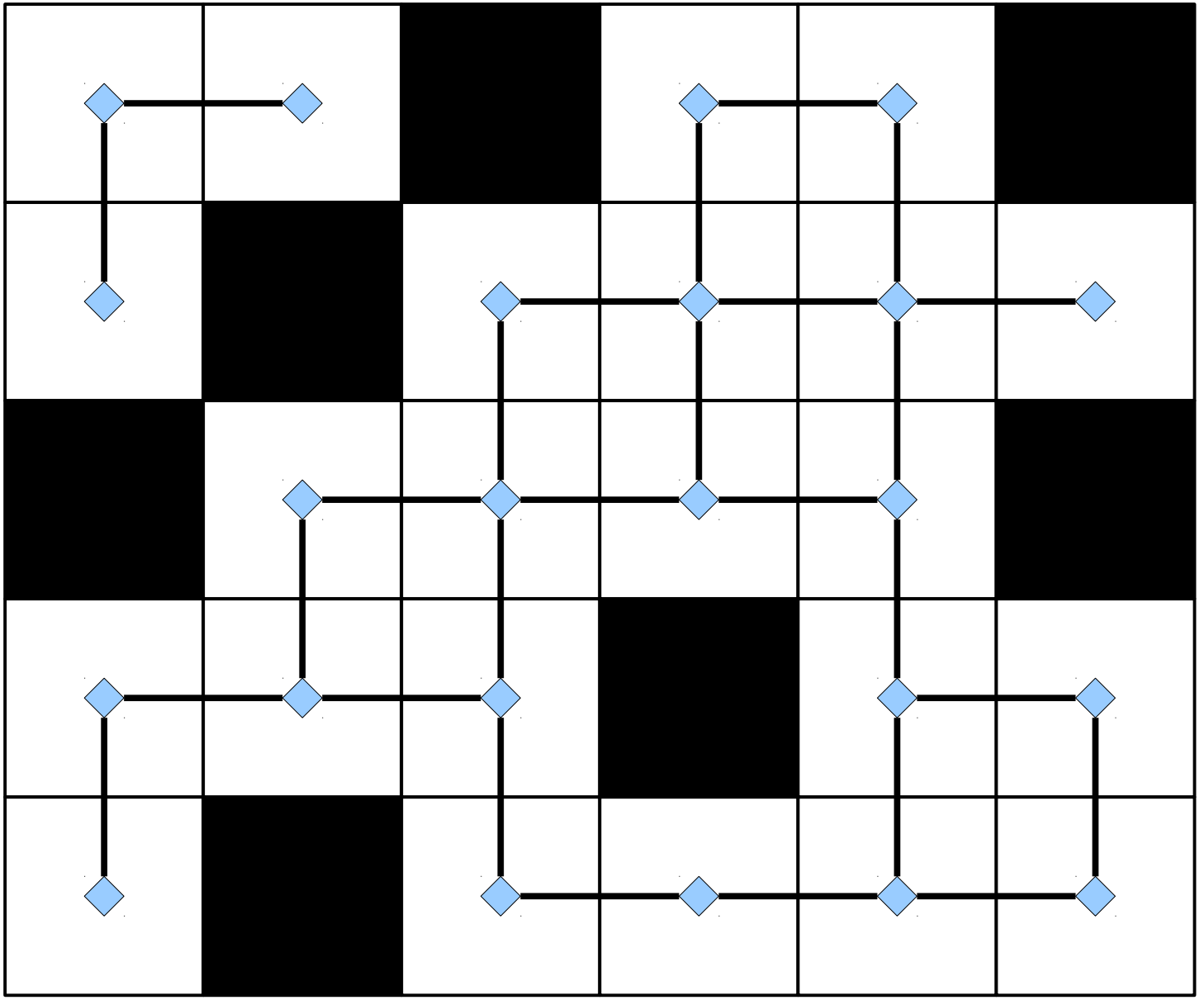
Solving Domino Tiling



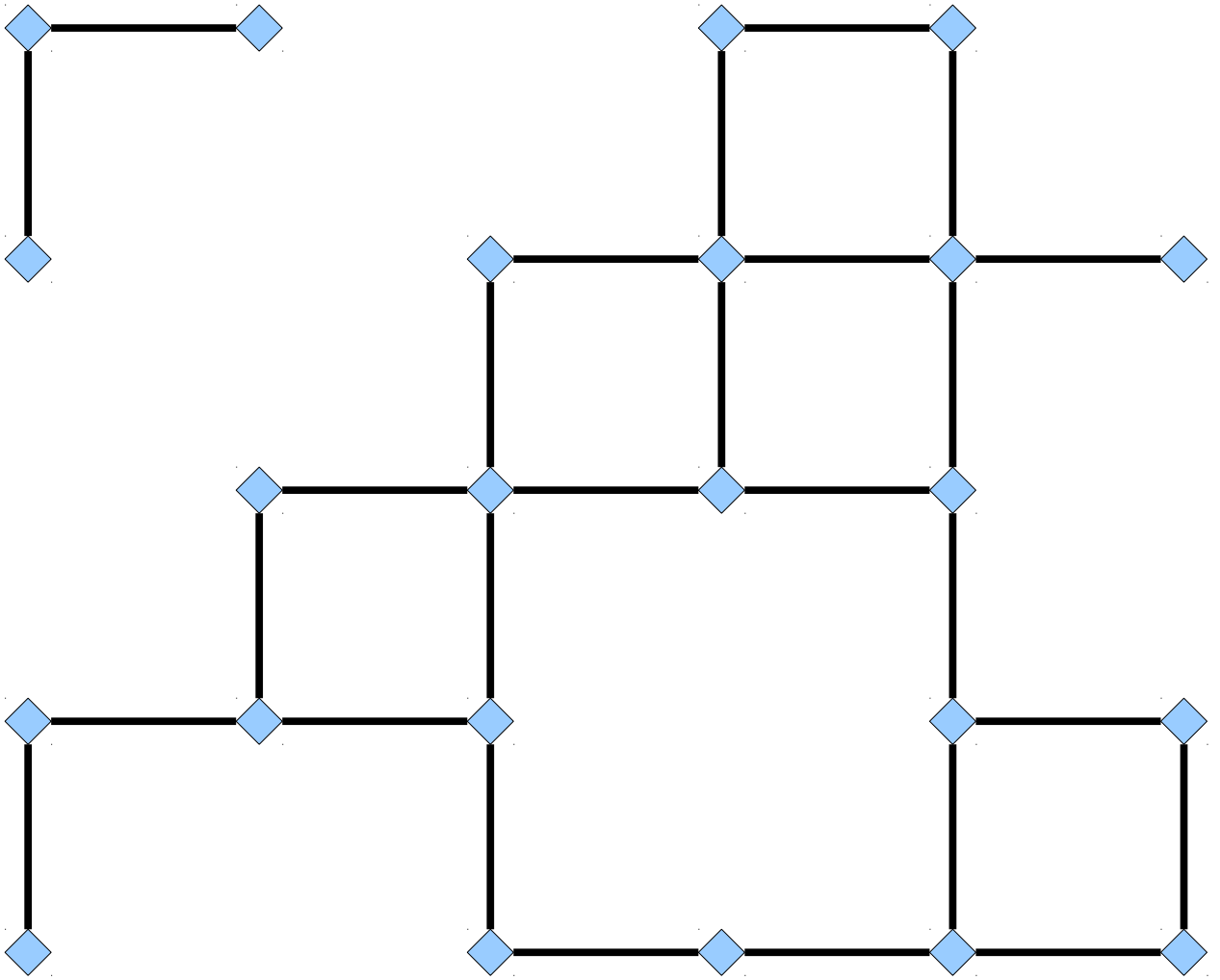
Solving Domino Tiling



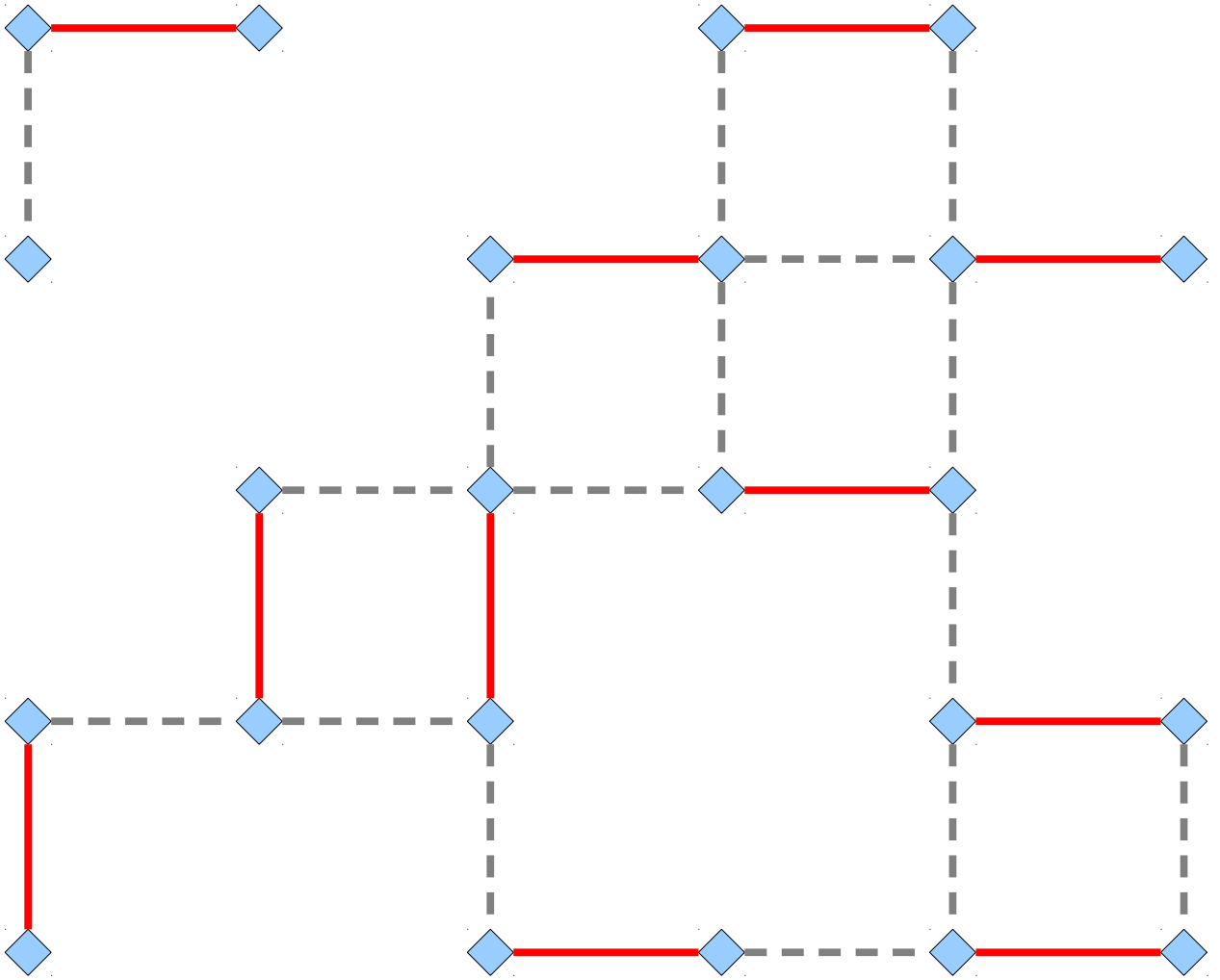
Solving Domino Tiling



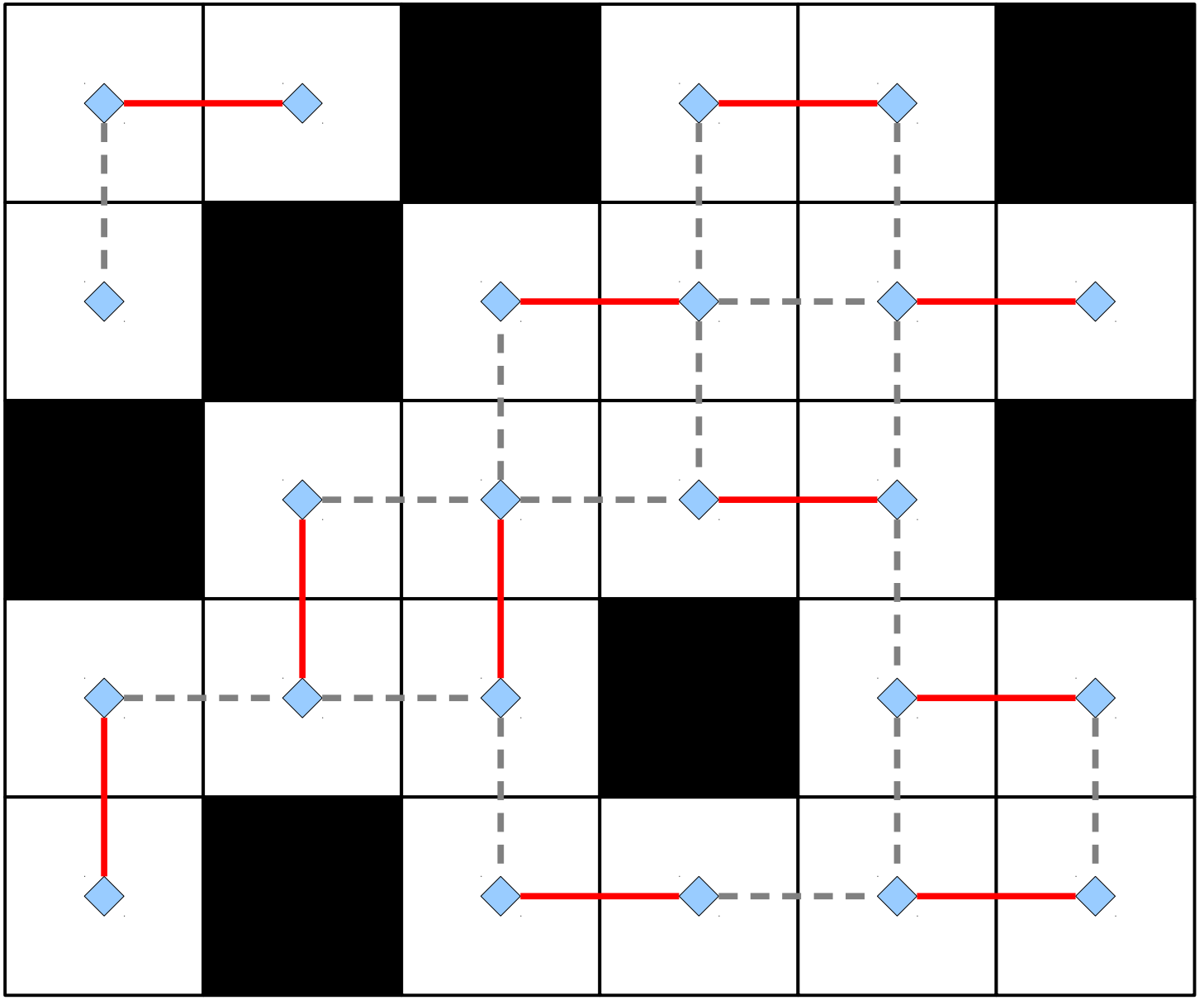
Solving Domino Tiling



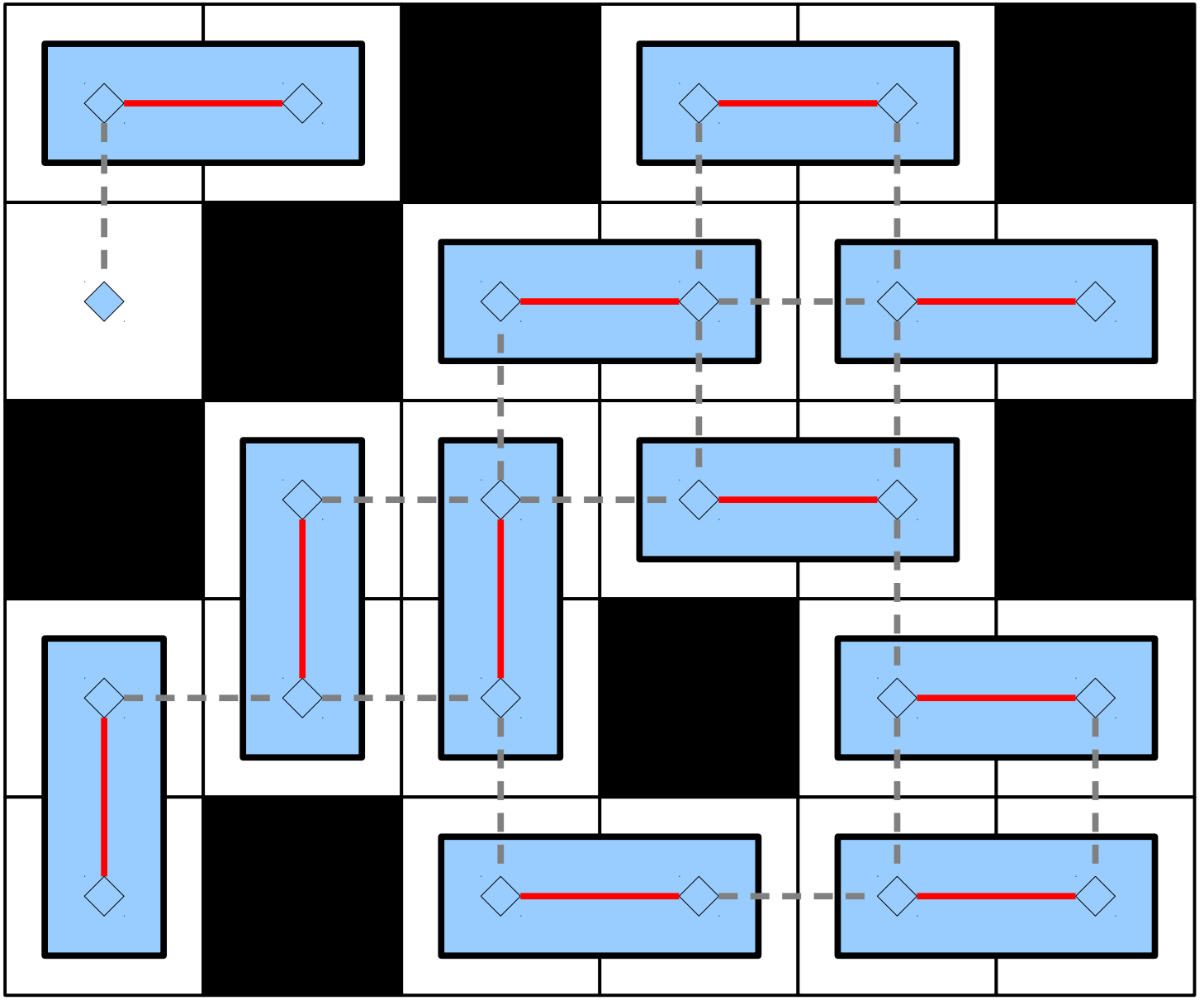
Solving Domino Tiling



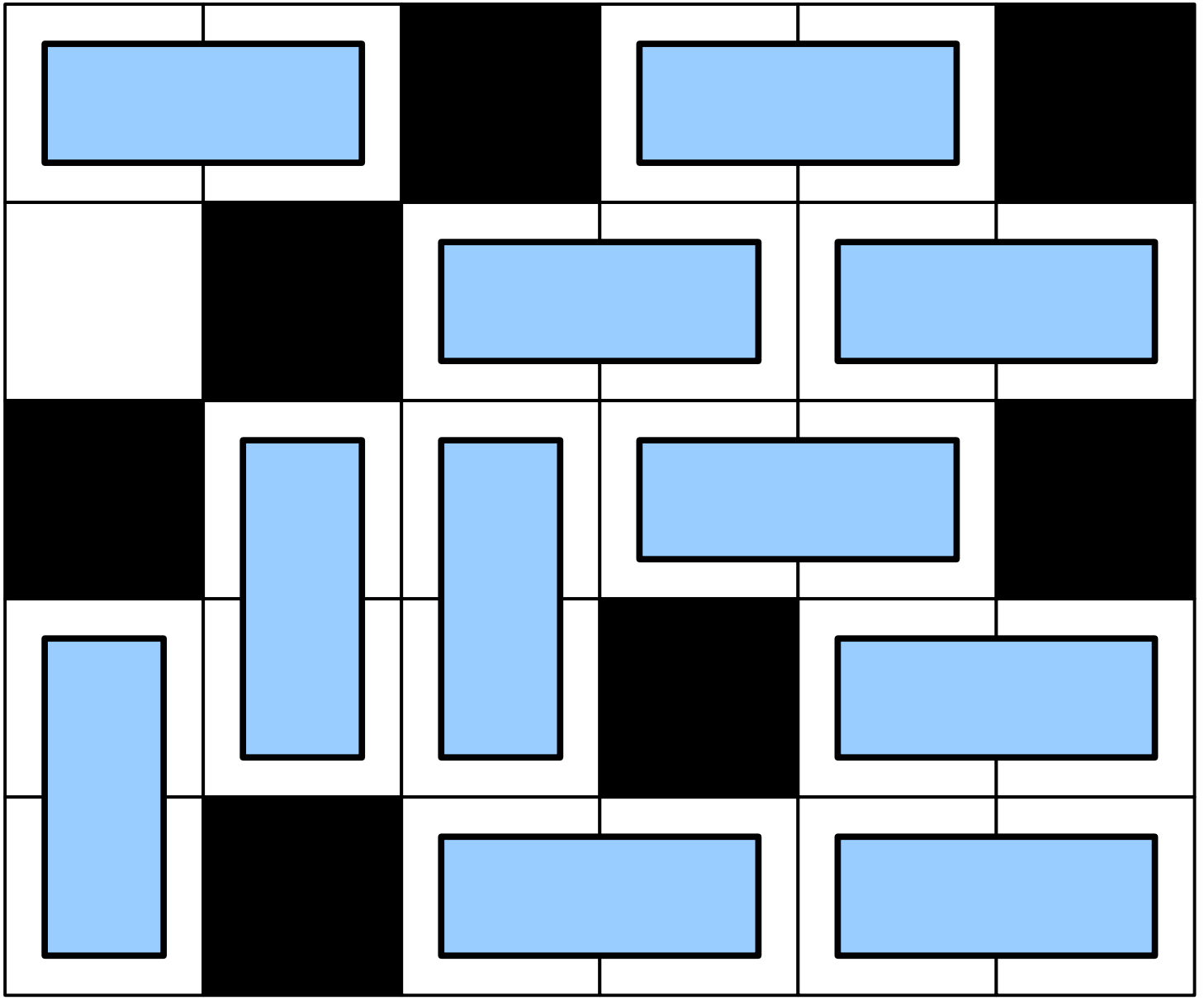
Solving Domino Tiling



Solving Domino Tiling

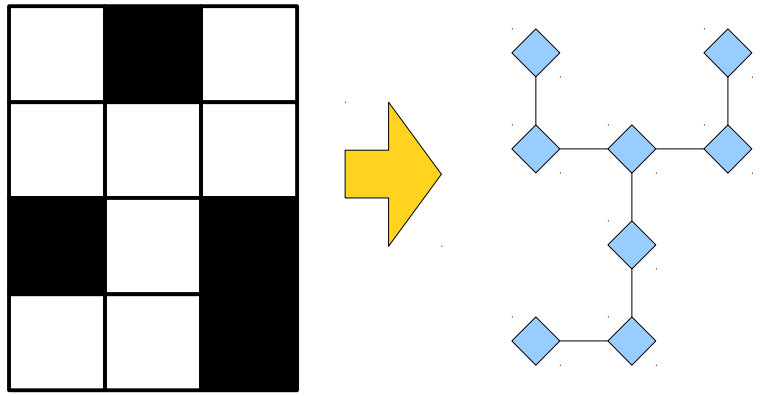


Solving Domino Tiling



Our Reduction

- Given as input $\langle D, k \rangle$, construct the graph G as follows:
  - For each empty cell, construct a node.
  - For each pair of adjacent empty cells, construct an edge between them.



- Let $f(\langle D, k \rangle) = \langle G, k \rangle$.

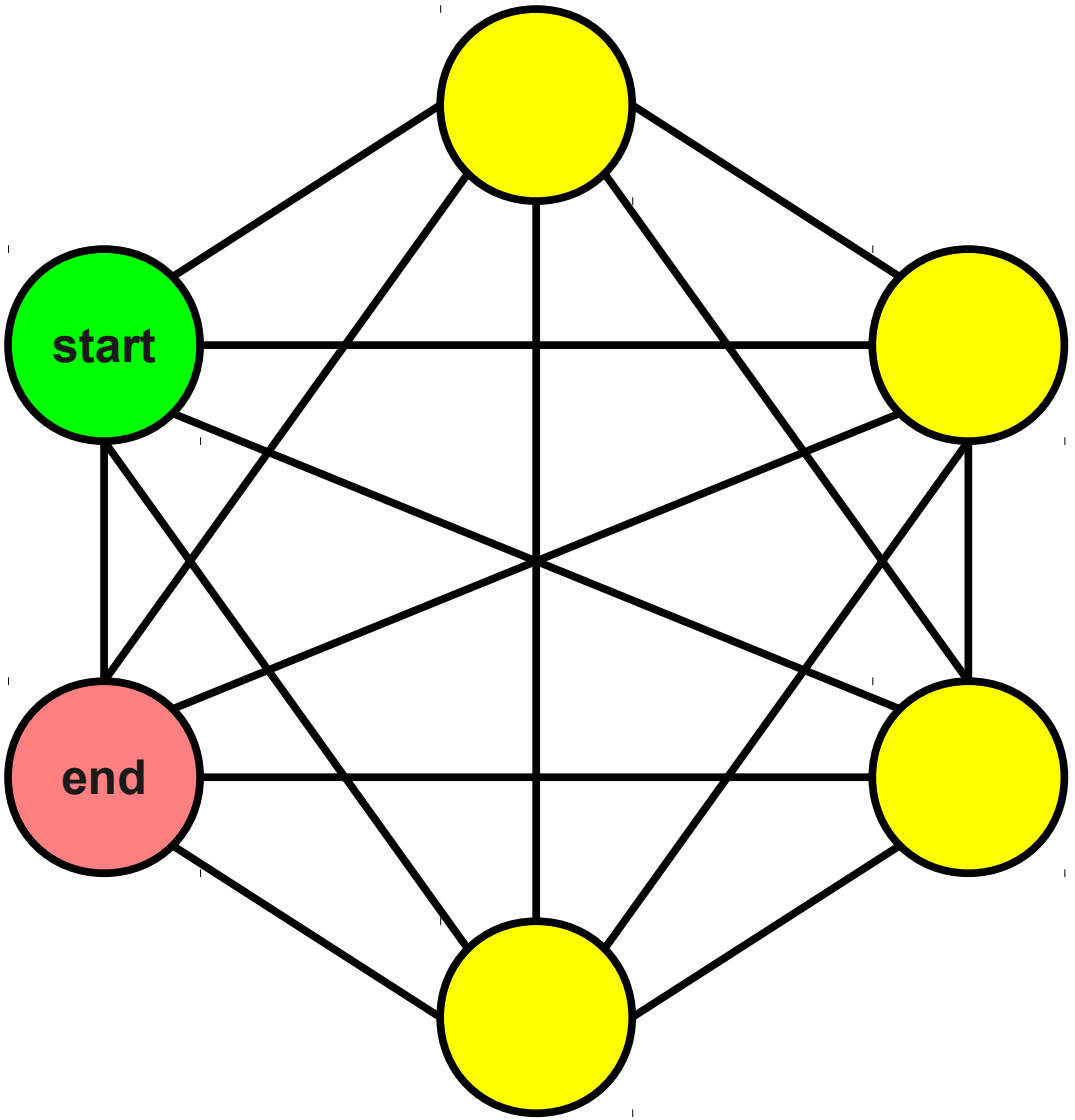
Lemma: f is computable in polynomial time.

Proof: We show that $f(\langle D, k\rangle) = \langle G, k\rangle$ has size that is a polynomial in the size of $\langle D, k\rangle$.

For each empty cell x_i in D, we construct a single node v_i in G. Since there are $O(|D|)$ cells, there are $O(|D|)$ nodes in the graph. For each pair of adjacent, empty cells x_i and x_j in D, we add the edge (x_i, x_j). Since each cell in D has four neighbors, the maximum number of edges we could add this way is $O(|D|)$ as well. Thus the total size of the graph G is $O(|D|)$. Consequently, the total size of $\langle G, k\rangle$ is $O(|D| + |k|)$, which is a polynomial in the size of the input.

Since each part of the graph could be constructed in polynomial time, the overall graph can be constructed in polynomial time. ■

What *can't* you do in polynomial time?



How many simple paths are there from the start node to the end node?

{ 

, 

, 

, 

}

How many subsets of this set are there?

An Interesting Observation

- There are (at least) exponentially many objects of each of the preceding types.
- However, each of those objects is not very large.
  - Each simple path has length no longer than the number of nodes in the graph.
  - Each subset of a set has no more elements than the original set.
- This brings us to our next topic...

What if you could magically guess which element of the search space was the one you wanted?

A Sample Problem

4	3	11	9	7	13	5	6	1	12	2	8	0	10

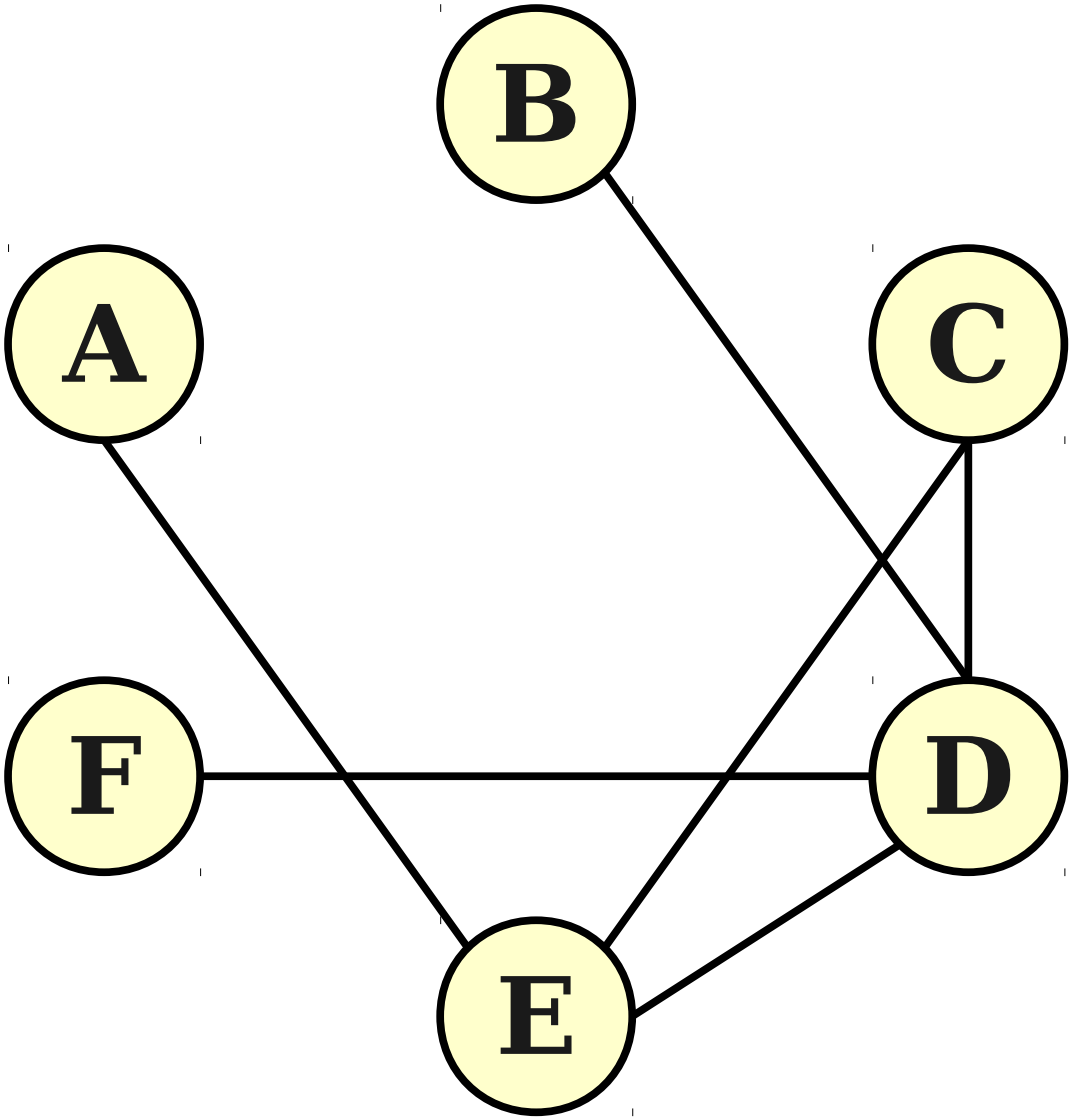
A Sample Problem

4	3	11	9	7	13	5	6	1	12	2	8	0	10

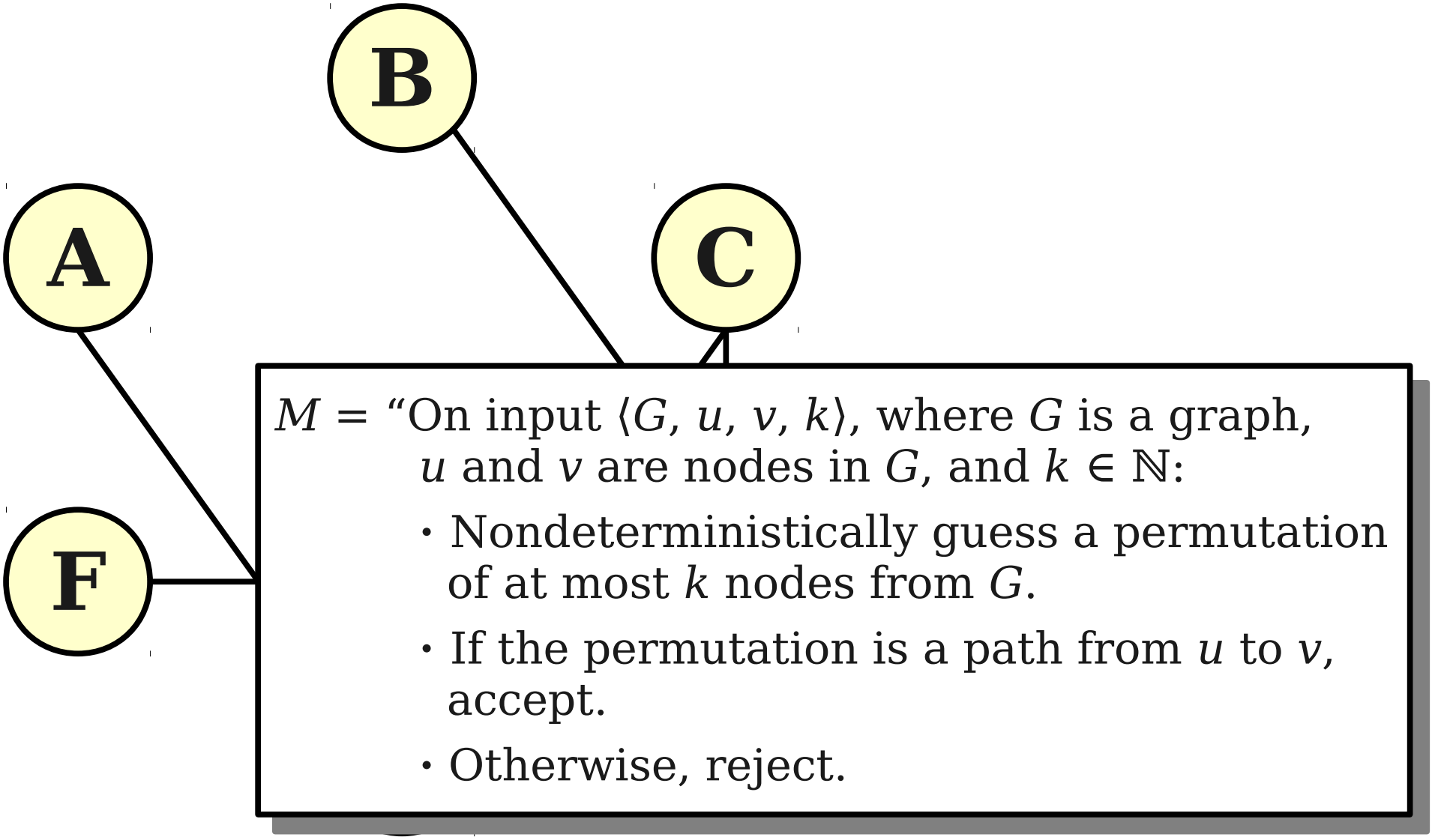
M = "On input $\langle S, k \rangle$, where S is a sequence of numbers and k is a natural number:

- Nondeterministically guess a subsequence of S.
- If it is an ascending subsequence of length at least k, accept.
- Otherwise, reject."

Another Problem



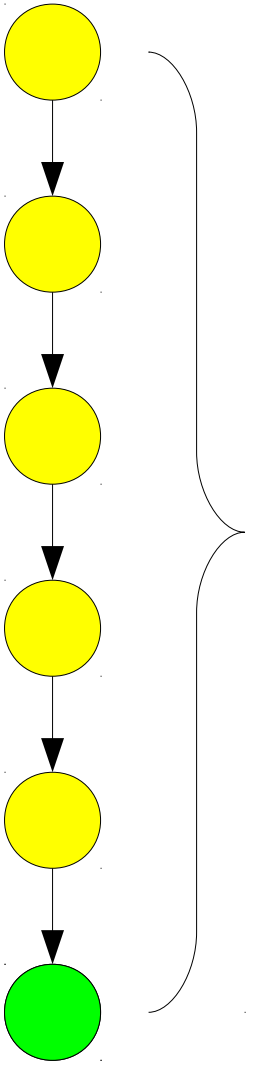
Another Problem



How do we measure NTM efficiency?

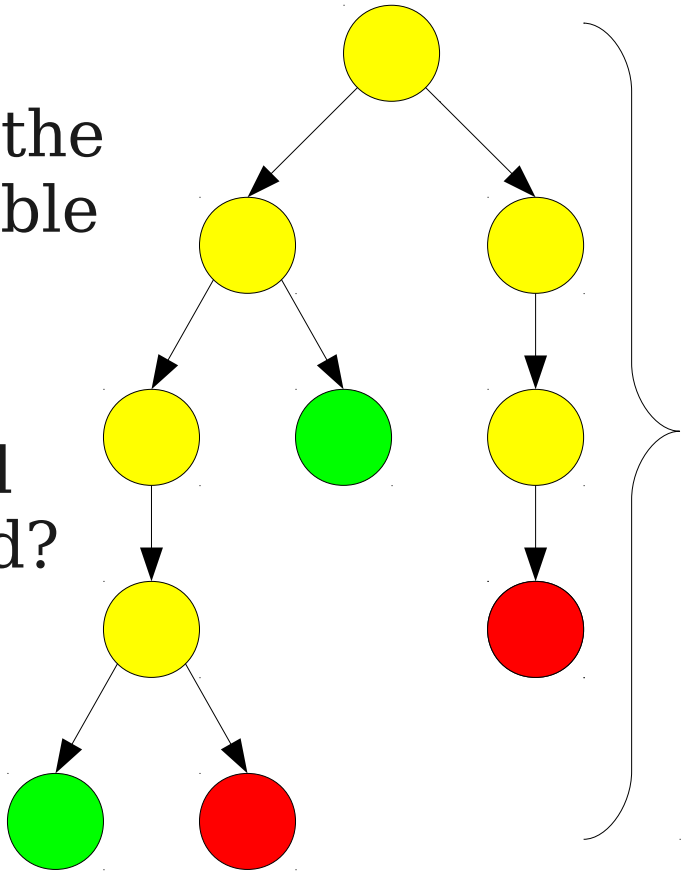
Analyzing NTMs

- When discussing deterministic TMs, the notion of time complexity is (reasonably) straightforward.
- **Recall:** One way of thinking about nondeterminism is as a tree.
- In a **deterministic** computation, the tree is a straight line.
- The time complexity is the height of that straight line.

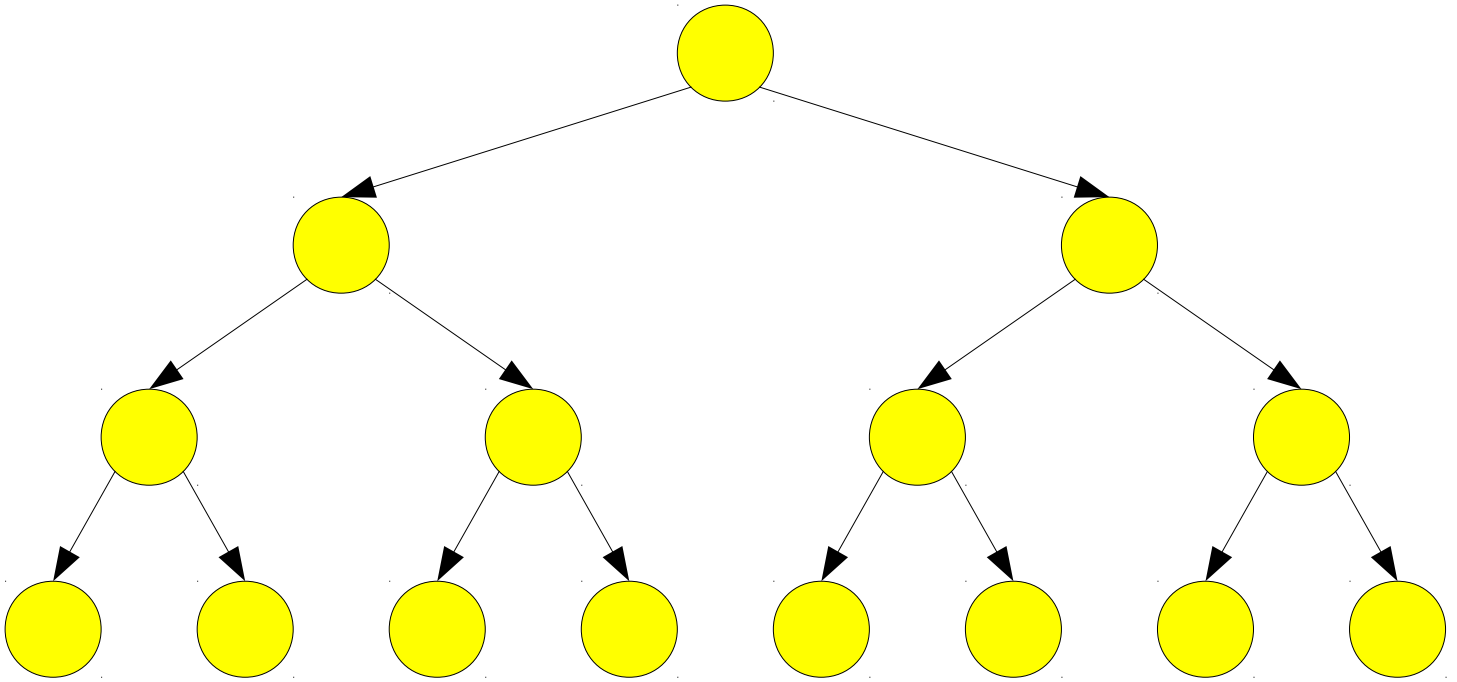


Analyzing NTMs

- When discussing deterministic TMs, the notion of time complexity is (reasonably) straightforward.
- **Recall:** One way of thinking about nondeterminism is as a tree.
- The time complexity is the height of the tree (the length of the **longest** possible choice we could make).
- Intuition: If you ran all possible branches in parallel, how long would it take before all branches completed?



The Size of the Tree



From NTMs to TMs

- **Theorem**: For any NTM with time complexity $f(n)$, there is a TM with time complexity $2^{O(f(n))}$.
- **It is unknown whether it is possible to do any better than this in the general case.**
- NTMs are capable of exploring multiple options in parallel; this "seems" inherently faster than deterministic computation.

The Complexity Class **NP**

- The complexity class **NP** (**nondeterministic polynomial time**) contains all problems that can be solved in polynomial time by an NTM.
- Formally:

  NP = { L | There is a nondeterministic TM that decides L in polynomial time. }

  What types of problems are in **NP**?

A Problem in **NP**

- Does a Sudoku grid have a solution?
  - M = "On input $\langle S \rangle$, an encoding of a Sudoku puzzle:
    - **Nondeterministically** guess how to fill in all the squares.
    - **Deterministically** check whether the guess is correct.
    - If so, accept; if not, reject."

		7		6		1		
					3		5	2
3			1		5	9		7
6		5		3		8		9
	1						2	
8		2		1		5		4
1		3	2		7			8
5	7		4					
		4		8		7		

A Problem in **NP**

- Does a Sudoku grid have a solution?
  - M = "On input $\langle S \rangle$, an encoding of a Sudoku puzzle:
    - **Nondeterministically** guess how to fill in all the squares.
    - **Deterministically** check whether the guess is correct.
    - If so, accept; if not, reject."

2	5	7	9	6	4	1	8	3
4	9	1	8	7	3	6	5	2
3	8	6	1	2	5	9	4	7
6	4	5	7	3	2	8	1	9
7	1	9	5	4	8	3	2	6
8	3	2	6	1	9	5	7	4
1	6	3	2	5	7	4	9	8
5	7	8	4	9	6	2	3	1
9	2	4	3	8	1	7	6	5

A Problem in **NP**

- Does a Sudoku grid have a solution?
  - M = "On input $\langle S \rangle$, an encoding of a Sudoku puzzle:
    - **Nondeterministically** guess how to fill in all the squares.
    - **Deterministically** check whether the guess is correct.
    - If so, accept; if not, reject."

For an arbitrary $n^2 \times n^2$ grid:

2	5	7	9	6	4	1	8	3
4	9	1	8	7	3	6	5	2
3	8	6	1	2	5	9	4	7
6	4	5	7	3	2	8	1	9
7	1	9	5	4	8	3	2	6
8	3	2	6	1	9	5	7	4
1	6	3	2	5	7	4	9	8
5	7	8	4	9	6	2	3	1
9	2	4	3	8	1	7	6	5

A Problem in **NP**

- Does a Sudoku grid have a solution?
  - M = "On input $\langle S \rangle$, an encoding of a Sudoku puzzle:
    - **Nondeterministically** guess how to fill in all the squares.
    - **Deterministically** check whether the guess is correct.
    - If so, accept; if not, reject."

For an arbitrary $n^2 \times n^2$ grid:

Total number of cells in the grid:

2	5	7	9	6	4	1	8	3
4	9	1	8	7	3	6	5	2
3	8	6	1	2	5	9	4	7
6	4	5	7	3	2	8	1	9
7	1	9	5	4	8	3	2	6
8	3	2	6	1	9	5	7	4
1	6	3	2	5	7	4	9	8
5	7	8	4	9	6	2	3	1
9	2	4	3	8	1	7	6	5

A Problem in **NP**

- Does a Sudoku grid have a solution?
  - M = "On input $\langle S \rangle$, an encoding of a Sudoku puzzle:
    - **Nondeterministically** guess how to fill in all the squares.
    - **Deterministically** check whether the guess is correct.
    - If so, accept; if not, reject."

For an arbitrary $n^2 \times n^2$ grid:

Total number of cells in the grid: $\boldsymbol{n^4}$

2	5	7	9	6	4	1	8	3
4	9	1	8	7	3	6	5	2
3	8	6	1	2	5	9	4	7
6	4	5	7	3	2	8	1	9
7	1	9	5	4	8	3	2	6
8	3	2	6	1	9	5	7	4
1	6	3	2	5	7	4	9	8
5	7	8	4	9	6	2	3	1
9	2	4	3	8	1	7	6	5

A Problem in **NP**

- Does a Sudoku grid have a solution?
  - M = "On input $\langle S \rangle$, an encoding of a Sudoku puzzle:
    - **Nondeterministically** guess how to fill in all the squares.
    - **Deterministically** check whether the guess is correct.
    - If so, accept; if not, reject."

For an arbitrary $n^2 \times n^2$ grid:

Total number of cells in the grid: $\boldsymbol{n^4}$

Total time to fill in the grid:

2	5	7	9	6	4	1	8	3
4	9	1	8	7	3	6	5	2
3	8	6	1	2	5	9	4	7
6	4	5	7	3	2	8	1	9
7	1	9	5	4	8	3	2	6
8	3	2	6	1	9	5	7	4
1	6	3	2	5	7	4	9	8
5	7	8	4	9	6	2	3	1
9	2	4	3	8	1	7	6	5

A Problem in **NP**

- Does a Sudoku grid have a solution?
  - M = "On input $\langle S \rangle$, an encoding of a Sudoku puzzle:
    - **Nondeterministically** guess how to fill in all the squares.
    - **Deterministically** check whether the guess is correct.
    - If so, accept; if not, reject."

For an arbitrary $n^2 \times n^2$ grid:

Total number of cells in the grid: $\boldsymbol{n^4}$

Total time to fill in the grid: $\boldsymbol{O(n^4)}$

2	5	7	9	6	4	1	8	3
4	9	1	8	7	3	6	5	2
3	8	6	1	2	5	9	4	7
6	4	5	7	3	2	8	1	9
7	1	9	5	4	8	3	2	6
8	3	2	6	1	9	5	7	4
1	6	3	2	5	7	4	9	8
5	7	8	4	9	6	2	3	1
9	2	4	3	8	1	7	6	5

A Problem in **NP**

- Does a Sudoku grid have a solution?
  - M = "On input $\langle S \rangle$, an encoding of a Sudoku puzzle:
    - **Nondeterministically** guess how to fill in all the squares.
    - **Deterministically** check whether the guess is correct.
    - If so, accept; if not, reject."

For an arbitrary $n^2 \times n^2$ grid:

Total number of cells in the grid: $\boldsymbol{n^4}$

Total time to fill in the grid: $\boldsymbol{O(n^4)}$

Total number of rows, columns, and boxes to check:

2	5	7	9	6	4	1	8	3
4	9	1	8	7	3	6	5	2
3	8	6	1	2	5	9	4	7
6	4	5	7	3	2	8	1	9
7	1	9	5	4	8	3	2	6
8	3	2	6	1	9	5	7	4
1	6	3	2	5	7	4	9	8
5	7	8	4	9	6	2	3	1
9	2	4	3	8	1	7	6	5

A Problem in **NP**

- Does a Sudoku grid have a solution?
  - M = "On input $\langle S \rangle$, an encoding of a Sudoku puzzle:
    - **Nondeterministically** guess how to fill in all the squares.
    - **Deterministically** check whether the guess is correct.
    - If so, accept; if not, reject."

For an arbitrary $n^2 \times n^2$ grid:

Total number of cells in the grid: $\boldsymbol{n^4}$

Total time to fill in the grid: $\boldsymbol{O(n^4)}$

Total number of rows, columns, and boxes to check: $\boldsymbol{O(n^2)}$

2	5	7	9	6	4	1	8	3
4	9	1	8	7	3	6	5	2
3	8	6	1	2	5	9	4	7
6	4	5	7	3	2	8	1	9
7	1	9	5	4	8	3	2	6
8	3	2	6	1	9	5	7	4
1	6	3	2	5	7	4	9	8
5	7	8	4	9	6	2	3	1
9	2	4	3	8	1	7	6	5

A Problem in **NP**

- Does a Sudoku grid have a solution?
  - M = "On input $\langle S \rangle$, an encoding of a Sudoku puzzle:
    - **Nondeterministically** guess how to fill in all the squares.
    - **Deterministically** check whether the guess is correct.
    - If so, accept; if not, reject."

For an arbitrary $n^2 \times n^2$ grid:

Total number of cells in the grid: $\boldsymbol{n^4}$

Total time to fill in the grid: $\boldsymbol{O(n^4)}$

Total number of rows, columns, and boxes to check: $\boldsymbol{O(n^2)}$

Total time required to check each row/column/box:

2	5	7	9	6	4	1	8	3
4	9	1	8	7	3	6	5	2
3	8	6	1	2	5	9	4	7
6	4	5	7	3	2	8	1	9
7	1	9	5	4	8	3	2	6
8	3	2	6	1	9	5	7	4
1	6	3	2	5	7	4	9	8
5	7	8	4	9	6	2	3	1
9	2	4	3	8	1	7	6	5

A Problem in **NP**

- Does a Sudoku grid have a solution?
  - M = "On input $\langle S \rangle$, an encoding of a Sudoku puzzle:
    - **Nondeterministically** guess how to fill in all the squares.
    - **Deterministically** check whether the guess is correct.
    - If so, accept; if not, reject."

For an arbitrary $n^2 \times n^2$ grid:

Total number of cells in the grid: $\boldsymbol{n^4}$

Total time to fill in the grid: $\boldsymbol{O(n^4)}$

Total number of rows, columns, and boxes to check: $\boldsymbol{O(n^2)}$

Total time required to check each row/column/box: $\boldsymbol{O(n^2)}$

2	5	7	9	6	4	1	8	3
4	9	1	8	7	3	6	5	2
3	8	6	1	2	5	9	4	7
6	4	5	7	3	2	8	1	9
7	1	9	5	4	8	3	2	6
8	3	2	6	1	9	5	7	4
1	6	3	2	5	7	4	9	8
5	7	8	4	9	6	2	3	1
9	2	4	3	8	1	7	6	5

A Problem in **NP**

- Does a Sudoku grid have a solution?
  - M = "On input $\langle S \rangle$, an encoding of a Sudoku puzzle:
    - **Nondeterministically** guess how to fill in all the squares.
    - **Deterministically** check whether the guess is correct.
    - If so, accept; if not, reject."

For an arbitrary $n^2 \times n^2$ grid:

Total number of cells in the grid: $\boldsymbol{n^4}$

Total time to fill in the grid: $\boldsymbol{O(n^4)}$

Total number of rows, columns, and boxes to check: $\boldsymbol{O(n^2)}$

Total time required to check each row/column/box: $\boldsymbol{O(n^2)}$

Total runtime:

2	5	7	9	6	4	1	8	3
4	9	1	8	7	3	6	5	2
3	8	6	1	2	5	9	4	7
6	4	5	7	3	2	8	1	9
7	1	9	5	4	8	3	2	6
8	3	2	6	1	9	5	7	4
1	6	3	2	5	7	4	9	8
5	7	8	4	9	6	2	3	1
9	2	4	3	8	1	7	6	5

A Problem in **NP**

- Does a Sudoku grid have a solution?
  - M = "On input $\langle S \rangle$, an encoding of a Sudoku puzzle:
    - **Nondeterministically** guess how to fill in all the squares.
    - **Deterministically** check whether the guess is correct.
    - If so, accept; if not, reject."

For an arbitrary $n^2 \times n^2$ grid:

Total number of cells in the grid: $\boldsymbol{n^4}$

Total time to fill in the grid: $\boldsymbol{O(n^4)}$

Total number of rows, columns, and boxes to check: $\boldsymbol{O(n^2)}$

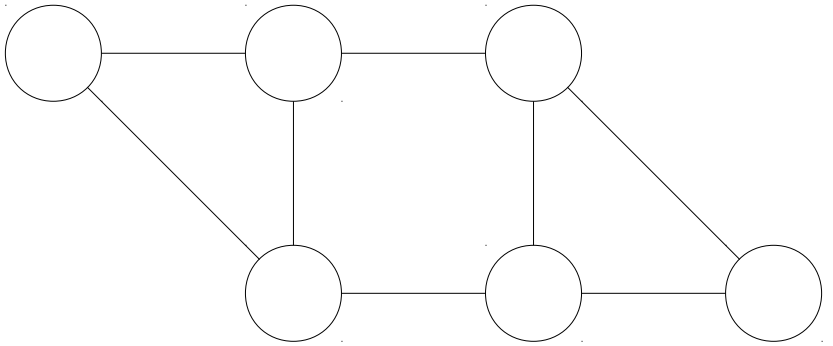
Total time required to check each row/column/box: $\boldsymbol{O(n^2)}$

Total runtime: $\boldsymbol{O(n^4)}$

2	5	7	9	6	4	1	8	3
4	9	1	8	7	3	6	5	2
3	8	6	1	2	5	9	4	7
6	4	5	7	3	2	8	1	9
7	1	9	5	4	8	3	2	6
8	3	2	6	1	9	5	7	4
1	6	3	2	5	7	4	9	8
5	7	8	4	9	6	2	3	1
9	2	4	3	8	1	7	6	5

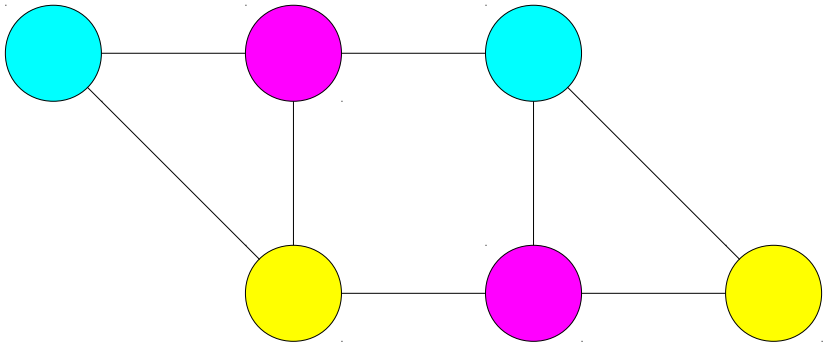
A Problem in **NP**

- A **graph coloring** is a way of assigning colors to nodes in an undirected graph such that no two nodes joined by an edge have the same color.
  - Applications in compilers, cell phone towers, etc.
- Question: Can graph G be colored with at most k colors?



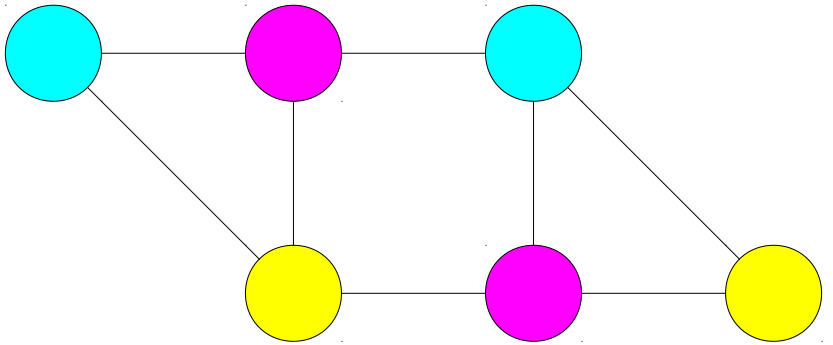
A Problem in **NP**

- A **graph coloring** is a way of assigning colors to nodes in an undirected graph such that no two nodes joined by an edge have the same color.
  - Applications in compilers, cell phone towers, etc.
- Question: Can graph G be colored with at most k colors?



A Problem in **NP**

- A **graph coloring** is a way of assigning colors to nodes in an undirected graph such that no two nodes joined by an edge have the same color.
  - Applications in compilers, cell phone towers, etc.
- Question: Can graph G be colored with at most k colors?
- M = "On input $\langle G, k \rangle$:
  - **Nondeterministically** guess a k-coloring of the nodes of G.
  - **Deterministically** check whether it is legal.
  - If so, accept; if not, reject."



Other Problems in **NP**

- **Subset sum:**

  Given a set S of natural numbers and a target number n, is there a subset of S that sums to n?

- **Longest path:**
  - Given a graph G, a pair of nodes u and v, and a number k, is there a simple path from u to v of length at least k?

- **Job scheduling:**
  - Given a set of jobs J, a number of workers k, and a time limit t, can the k workers, working in parallel complete all jobs in J within time t?

Problems and Languages

- Abstract question: does a Sudoku grid have a solution?
- Formalized as a language:

SUDOKU = { ⟨*S*⟩ | *S* is a solvable Sudoku grid. }

- In other words:

S is solvable iff $\langle S \rangle \in SUDOKU$

Problems and Languages

- Abstract question: can a graph be colored with k colors?
- Formalized as a language:

  $COLOR = \{ \langle G, k \rangle \mid$ G is an undirected graph, $k \in \mathbb{N}$, and G is k-colorable. $\}$

- In other words:

  G is k-colorable iff $\langle G, k \rangle \in COLOR$

A General Pattern

- The NTMs we have seen so far always follow this pattern:
  - M = "On input w:
    - **Nondeterministically** guess some object.
    - **Deterministically** check whether this was the right guess.
    - If so, accept; otherwise, reject."
- Intuition: The NTM is searching for some *proof* that w belongs to some language L.
  - If $w \in L$, it can guess the proof.
  - If $w \notin L$, it will never guess the proof.

An Intuition for **NP**

- Intuitively, a language L is in **NP** iff there is an easy way of proving strings in L actually belong to L.
- If $w \in L$, there is some information that can easily be used to convince someone that $w \in L$.

A Problem in **NP**

		7		6		1		
					3		5	2
3			1		5	9		7
6		5		3		8		9
	1						2	
8		2		1		5		4
1		3	2		7			8
5	7		4					
		4		8		7		

A Problem in **NP**

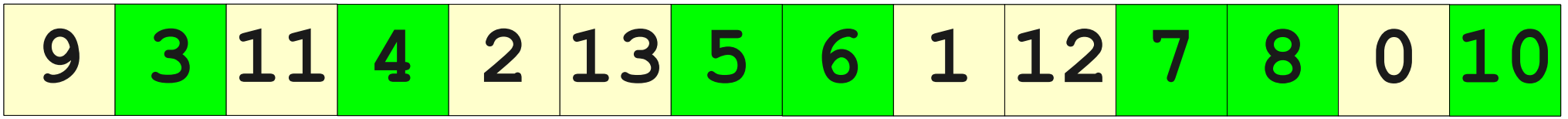
2	5	7	9	6	4	1	8	3
4	9	1	8	7	3	6	5	2
3	8	6	1	2	5	9	4	7
6	4	5	7	3	2	8	1	9
7	1	9	5	4	8	3	2	6
8	3	2	6	1	9	5	7	4
1	6	3	2	5	7	4	9	8
5	7	8	4	9	6	2	3	1
9	2	4	3	8	1	7	6	5

A Problem in **NP**

9	3	11	4	2	13	5	6	1	12	7	8	0	10

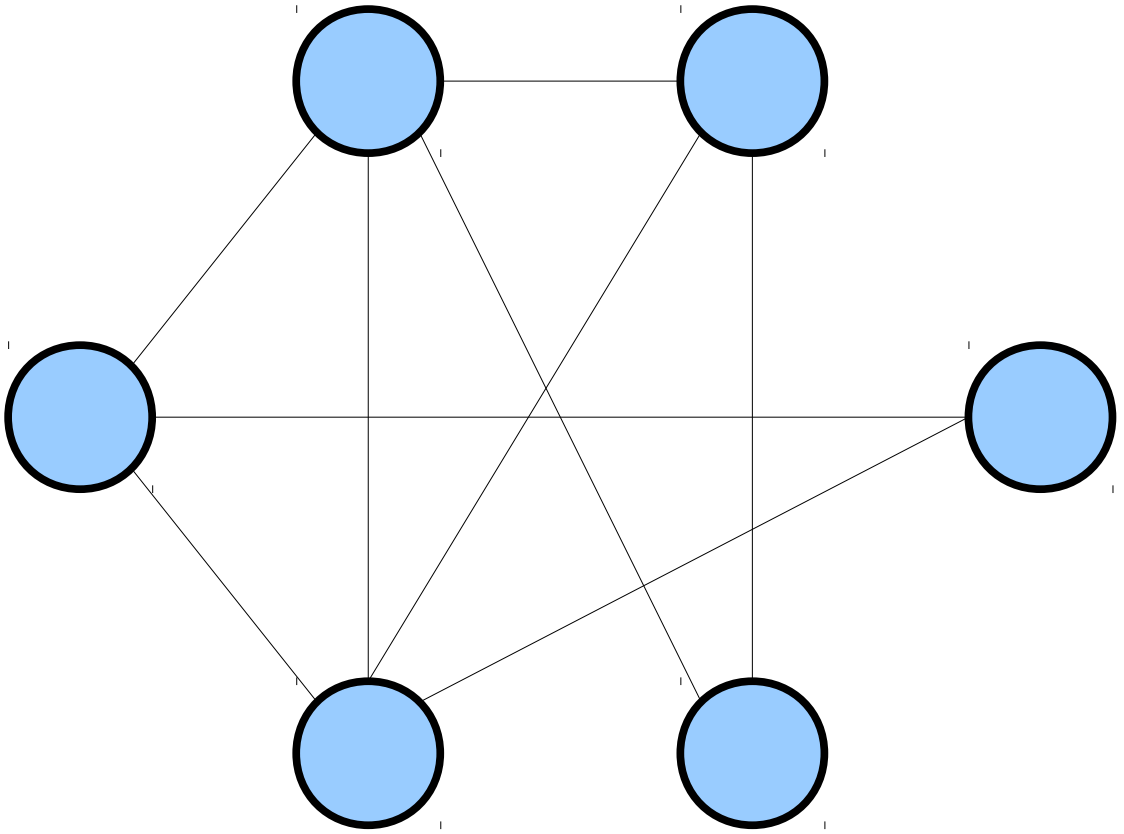
Is there an ascending subsequence of length at least 7?

A Problem in **NP**



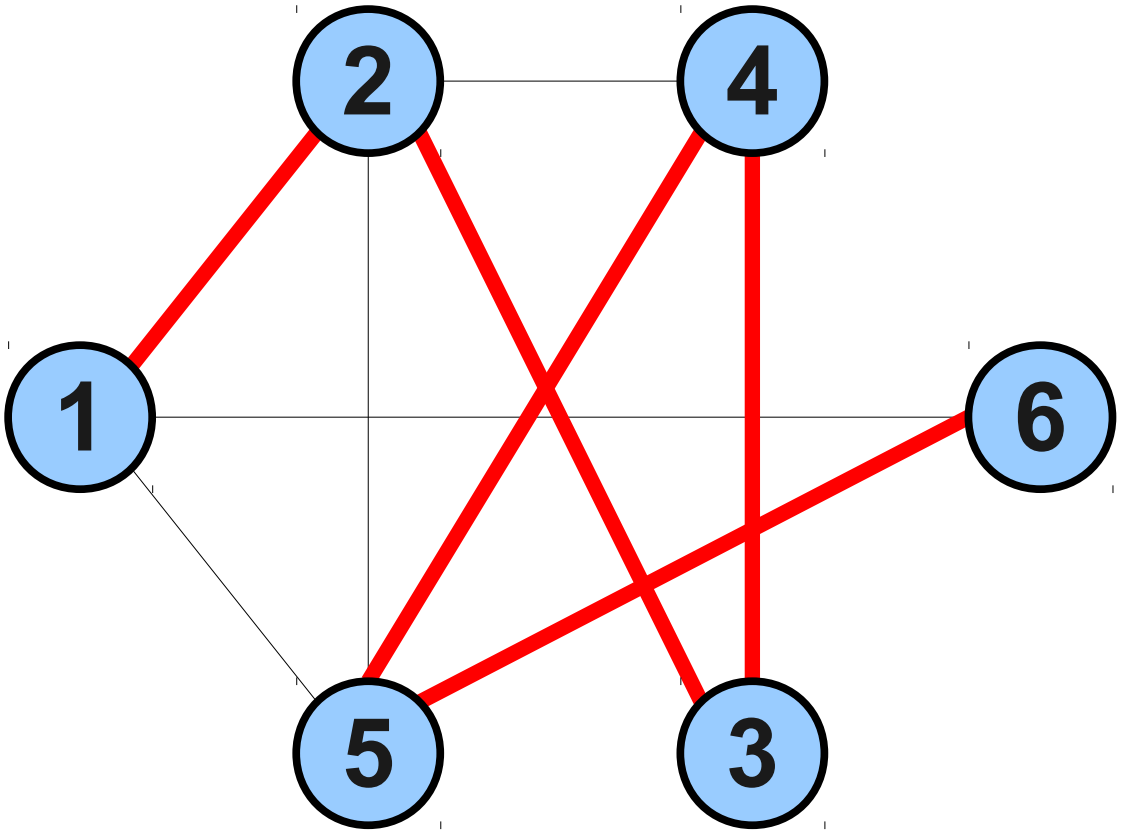
Is there an ascending subsequence of length at least 7?

A Problem in **NP**



Is there a simple path that goes through every node exactly once?

A Problem in **NP**



Is there a simple path that goes through every node exactly once?

Another View of **NP**

- ***Theorem:*** $L \in$ **NP** iff there is a *deterministic* TM V with the following properties:
  - $w \in L$ iff there is some $c \in \Sigma^*$ such that V accepts $\langle w, c \rangle$.
  - V runs in time polynomial in $|w|$.

Another View of **NP**

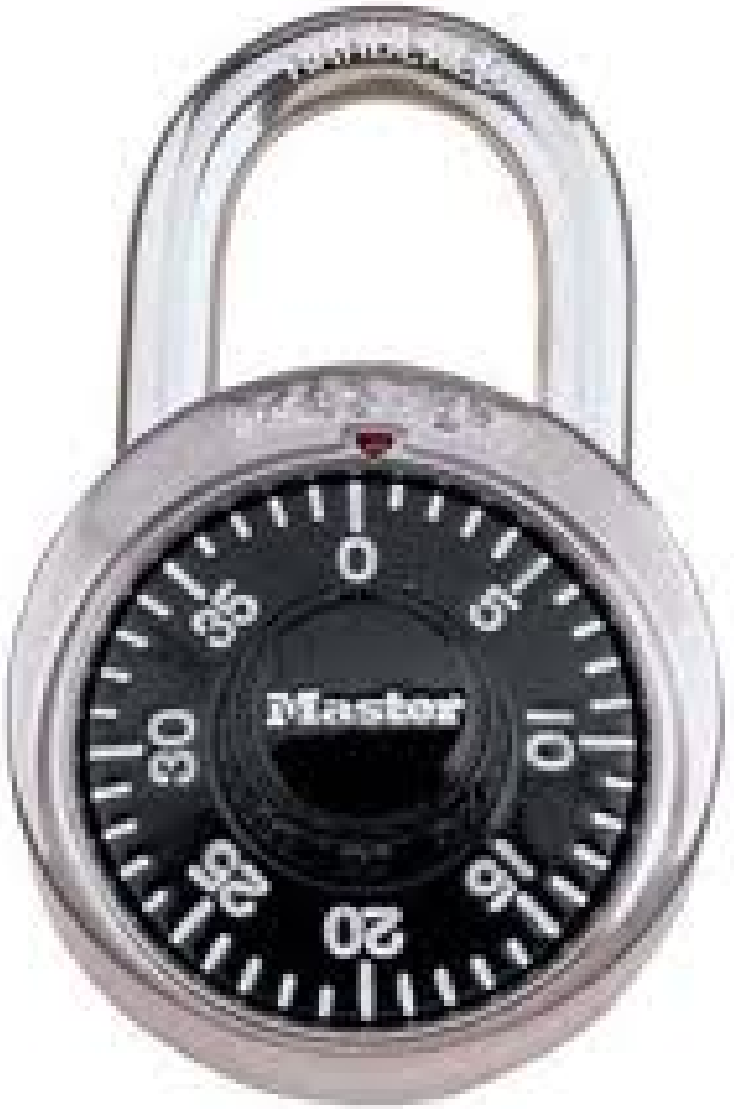
- ***Theorem:*** $L \in$ **NP** iff there is a *deterministic* TM V with the following properties:
  - $w \in L$ iff there is some $c \in \Sigma^*$ such that V accepts $\langle w, c \rangle$.
  - V runs in time polynomial in $|w|$.
- ***Intuition:*** Think about how you would convince someone what a string w belongs to an **NP** language L.
  - If $w \in L$, there is some information you can provide to easily convince someone that $w \in L$.
  - If $w \notin L$, then no information you provide can convince someone that $w \in L$.

Another View of **NP**

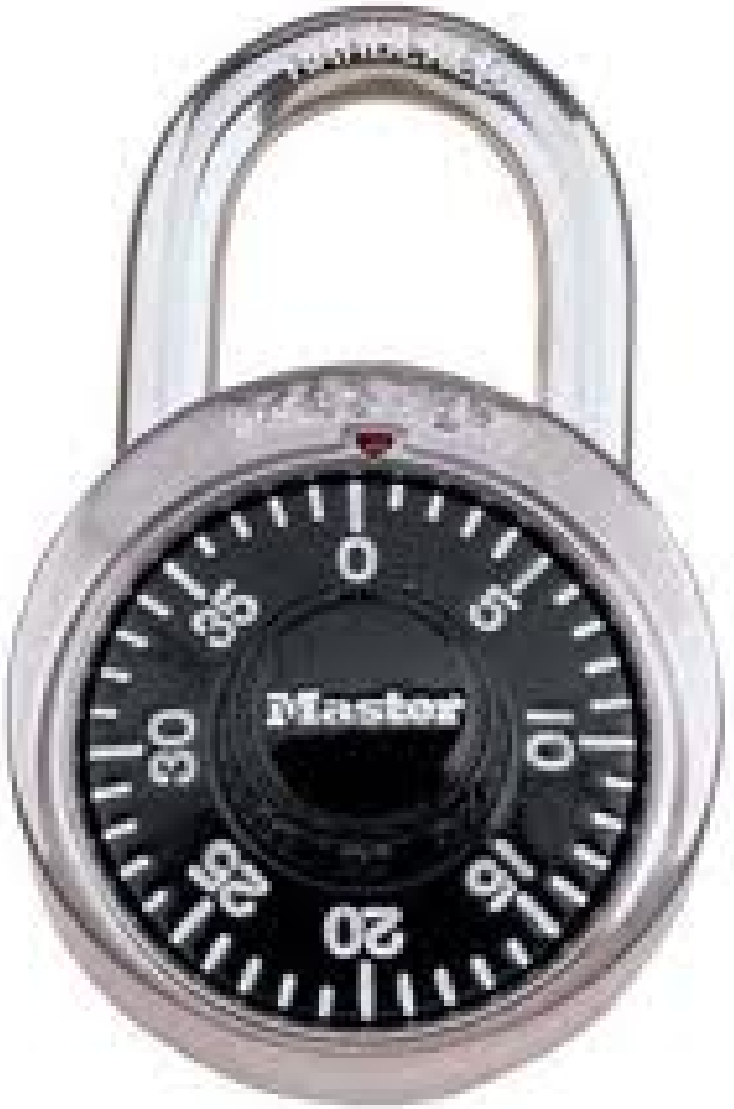
- ***Theorem:*** $L \in$ **NP** iff there is a *deterministic* TM V with the following properties:
  - $w \in L$ iff there is some $c \in \Sigma^*$ such that V accepts $\langle w, c \rangle$.
  - V runs in time polynomial in $|w|$.
- Some terminology:
  - A TM V with the above property is called a **polynomial-time verifier for *L***.
  - The string c is called a **certificate** for w.
  - You can think of V as checking the certificate that proves $w \in L$.

An Efficiently Verifiable Puzzle

An Efficiently Verifiable Puzzle



An Efficiently Verifiable Puzzle



Question: Can this lock be opened?

Another View of **NP**

- ***Theorem:*** $L \in$ **NP** iff there is a *deterministic* TM V with the following properties:
  - $w \in L$ iff there is some $c \in \Sigma^*$ such that V accepts $\langle w, c \rangle$.
  - V runs in time polynomial in $|w|$.
- Important properties of V:
  - If V accepts $\langle w, c \rangle$, then we're guaranteed $w \in L$.
  - If V does not accept $\langle w, c \rangle$, then either
    - $w \in L$, but you gave the wrong c, or
    - $w \notin L$, so no possible c will work.

Another View of **NP**

- ***Theorem:*** $L \in$ **NP** iff there is a *deterministic* TM V with the following properties:
  - $w \in L$ iff there is some $c \in \Sigma^*$ such that V accepts $\langle w, c \rangle$.
  - V runs in time polynomial in $|w|$.
- Important observations:
  - $\mathscr{L}(V)$ is **not** the language L.
  - L is the set of strings in the language, while $\mathscr{L}(V)$ is a set of strings in the language paired with certificates.
  - V **must** be deterministic.

Another View of **NP**

- ***Theorem:*** $L \in$ **NP** iff there is a *deterministic* TM V with the following properties:
  - $w \in L$ iff there is some $c \in \Sigma^*$ such that V accepts $\langle w, c \rangle$.
  - V runs in time polynomial in $|w|$.
- ***Proof sketch:***
  - If there is a verifier V for L, we can build a poly-time NTM for L by nondeterministically guessing a certificate c, then running V on w.
  - If there is a poly-time NTM for L, we can build a verifier for it. The certificate is the sequence of choices the NTM should make, and V checks that this sequence accepts.

A Problem in **NP**

- Does a Sudoku grid have a solution?
  - M = "On input $\langle S, A \rangle$, an encoding of a Sudoku puzzle and an alleged solution to it:
    - **Deterministically** check whether A is a solution to S.
    - If so, accept; if not, reject."

		7		6		1		
					3		5	2
3			1		5	9		7
6		5		3		8		9
	1						2	
8		2		1		5		4
1		3	2		7			8
5	7		4					
		4		8		7		

A Problem in **NP**

- A **graph coloring** is a way of assigning colors to nodes in an undirected graph such that no two nodes joined by an edge have the same color.
  - Applications in compilers, cell phone towers, etc.
- Question: Can G be colored with at most k colors?
- M = "On input $\langle\langle G, k\rangle, C\rangle$, where C is an alleged coloring:
  - **Deterministically** check whether C is a legal k-coloring of G.
  - If so, accept; if not, reject."