Complexity Theory

Part II

Outline for Today

- **Recap from Last Time**
  - Where are we, again?
- **P and NP**
  - Two fundamental complexity classes.
- **The P versus NP Question**
  - We know that **R** ≠ **RE** – what about **P** and **NP**?
- **Polynomial-Time Reducibility**
  - Connecting problems together.

Recap from Last Time

Nondeterministic TMs

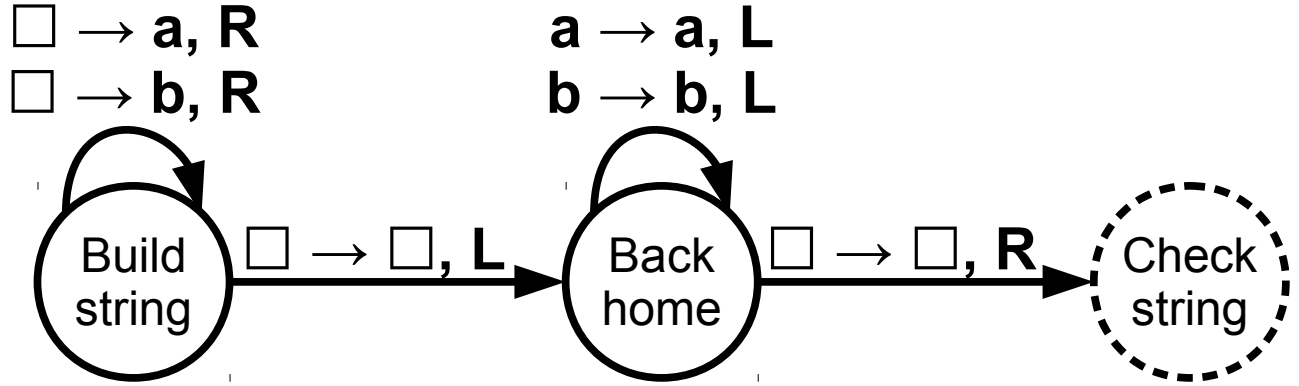
- A ***nondeterministic Turing machine*** (or ***NTM***) is a Turing machine in which there can be zero or multiple transitions defined at each state.
- As with NFAs, NTMs accept if any path accepts. In other words, an NTM for a language L is one where

  $w \in L$ iff there is some series of choices N can make that causes N to accept w.

- In particular, if $w \in L$, N only needs to accept w along one branch. The rest can loop infinitely or reject.
- Unlike NFAs, you don't just end up with an NTM in multiple states at the same time. Each time an NTM makes a nondeterministic choice, the entire computation branches.

Guessing an Arbitrary String

- It's common to build NTMs that nondeterministically guess some arbitrary finite, discrete object, then deterministically check that the object has some properties.



- As a high-level description:

> N = "On input w:
> Nondeterministically guess a string $x \in \Sigma^*$.
> Deterministically check whether [...]"

Characterizing **RE**

- ***Theorem:*** The following are equivalent:
  - L is an **RE** language.
  - There is a recognizer for L.
  - There is a verifier for L.
  - There is an NTM for L.

Time Complexity

- We can measure the ***time complexity*** of a TM as the number of steps the TM takes to accept some input.
- We'll look at the worst-case runtime of an algorithm to determine how efficient it is.

Brute Force vs. Creativity

- **Longest increasing subsequence:**
  - Naive: $O(n \cdot 2^n)$
  - Fast: $O(n^2)$
- **Shortest path problem:**
  - Naive: $O(n^2 \cdot n!)$
  - Fast: $O(n + m)$, where n is the number of nodes and m the number of edges. (Take CS161 for details!)

The Cobham-Edmonds Thesis

A language L can be ***decided efficiently*** if 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 Cobham-Edmonds Thesis

- Efficient runtimes:
  - $4n + 13$
  - $n^3 - 2n^2 + 4n$
  - $n \log \log n$
- "Efficient" runtimes:
  - $n^{1,000,000,000,000}$
  - 10^{500}
- Inefficient runtimes:
  - 2^n
  - $n!$
  - n^n
- "Inefficient" runtimes:
  - $n^{0.0001 \log n}$
  - 1.000000001^n

The Complexity Class **P**

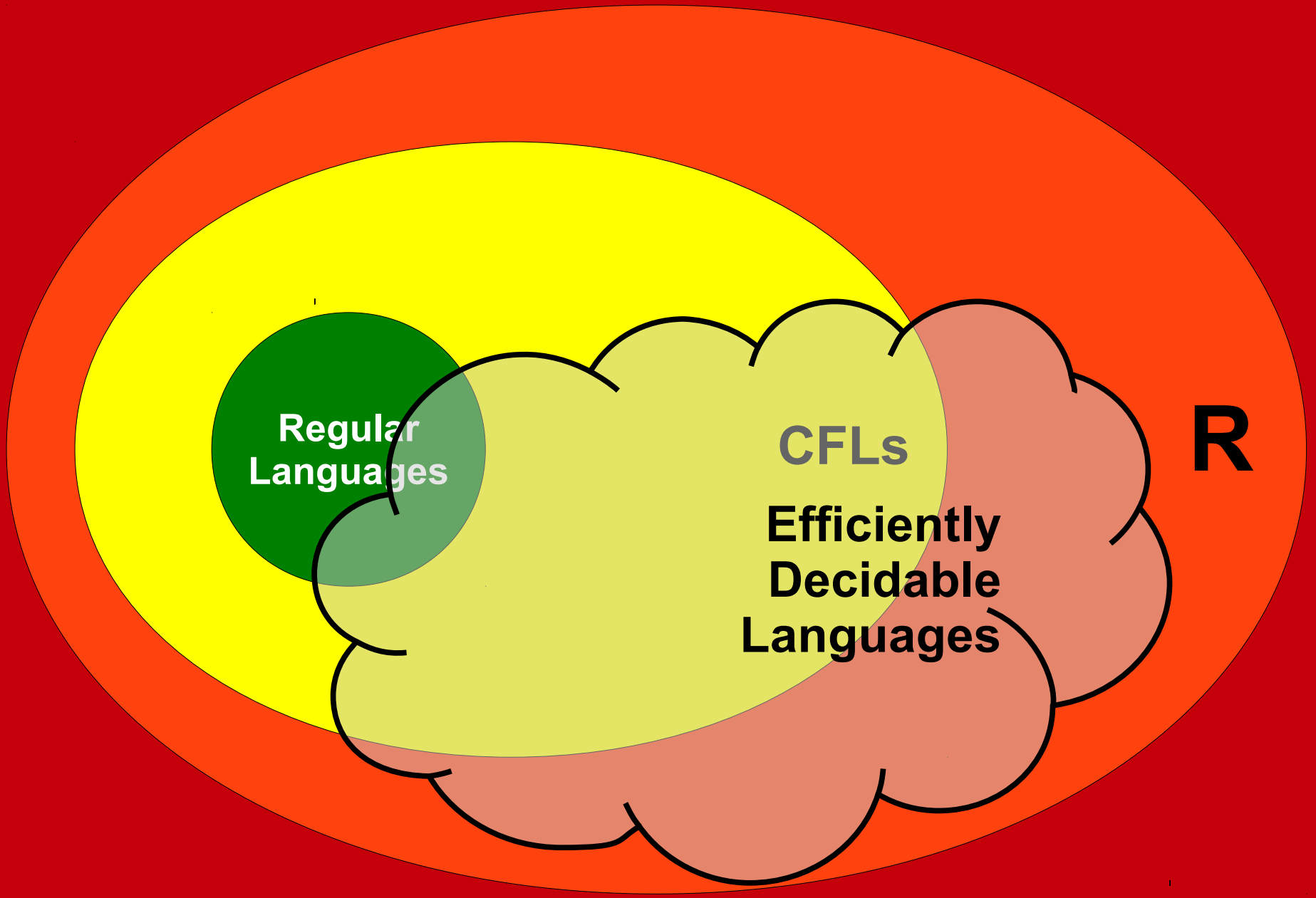
- The ***complexity class* P** (for ***p***olynomial time) contains all problems that can be solved in polynomial time.
- Formally:

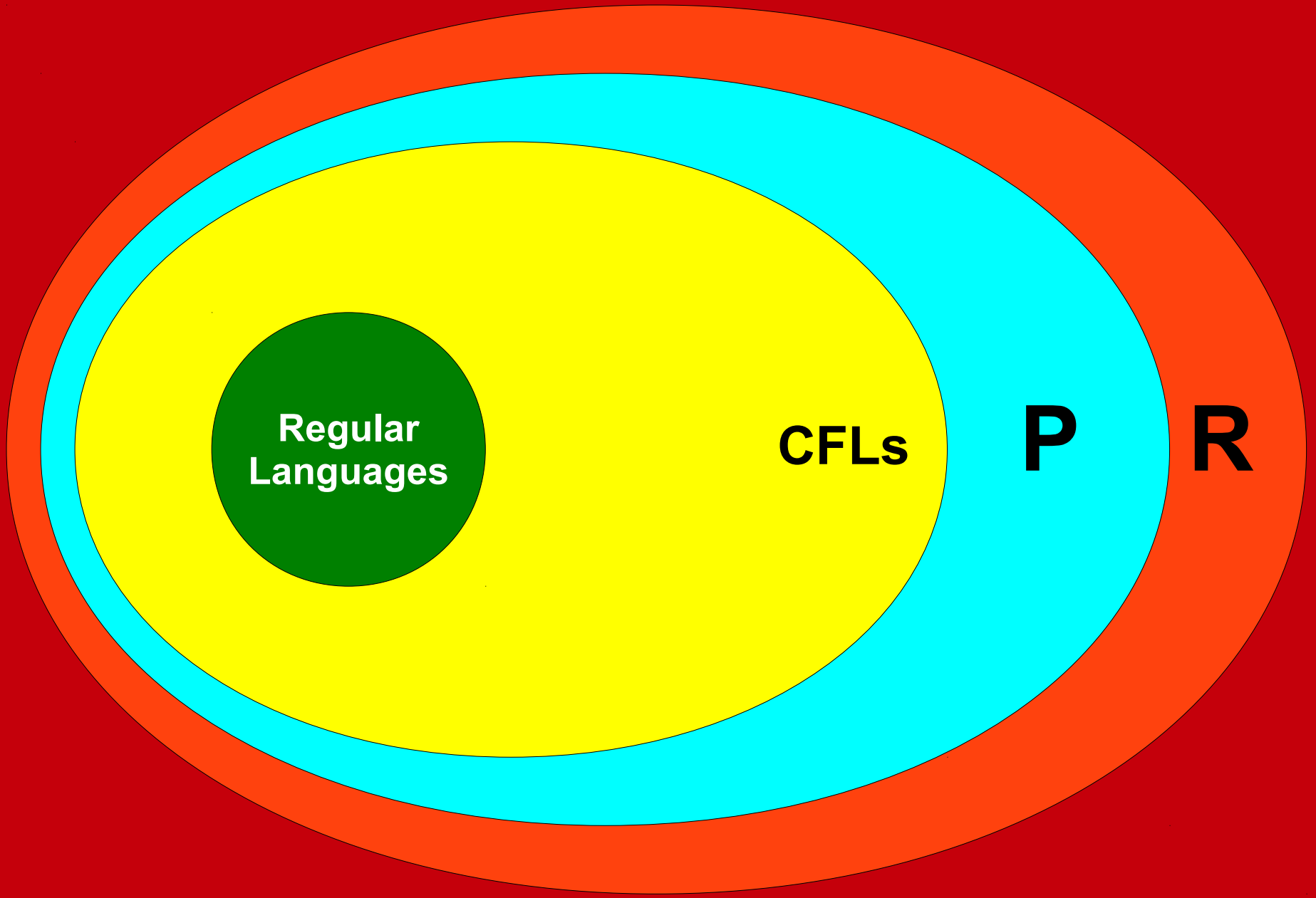
$$\mathbf{P} = \{ L \mid \text{There is a polynomial-time decider for } L \}$$

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

Examples of Problems in **P**

- All regular languages are in **P**.
  - All have linear-time TMs.
- All CFLs are in **P**.
  - Requires a more nuanced argument (the *CYK algorithm* or *Earley's algorithm*.)
- Many other problems are in **P**.
  - More on that in a second.


Regular Languages
CFLs
Efficiently Decidable Languages
R
Undecidable Languages


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 $O(n^{37})$.)

- **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?

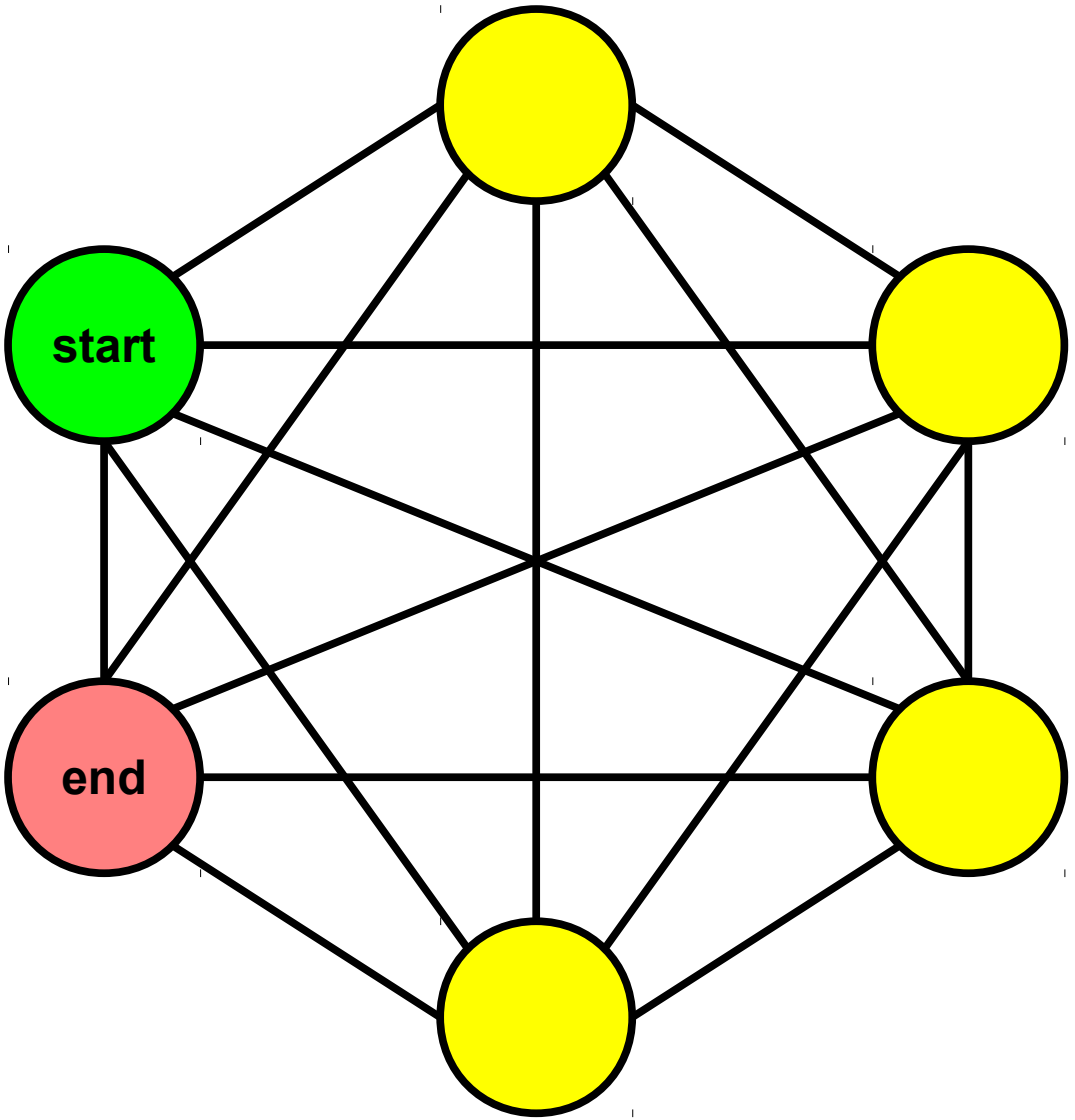
- **Linear programming:**

  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?

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

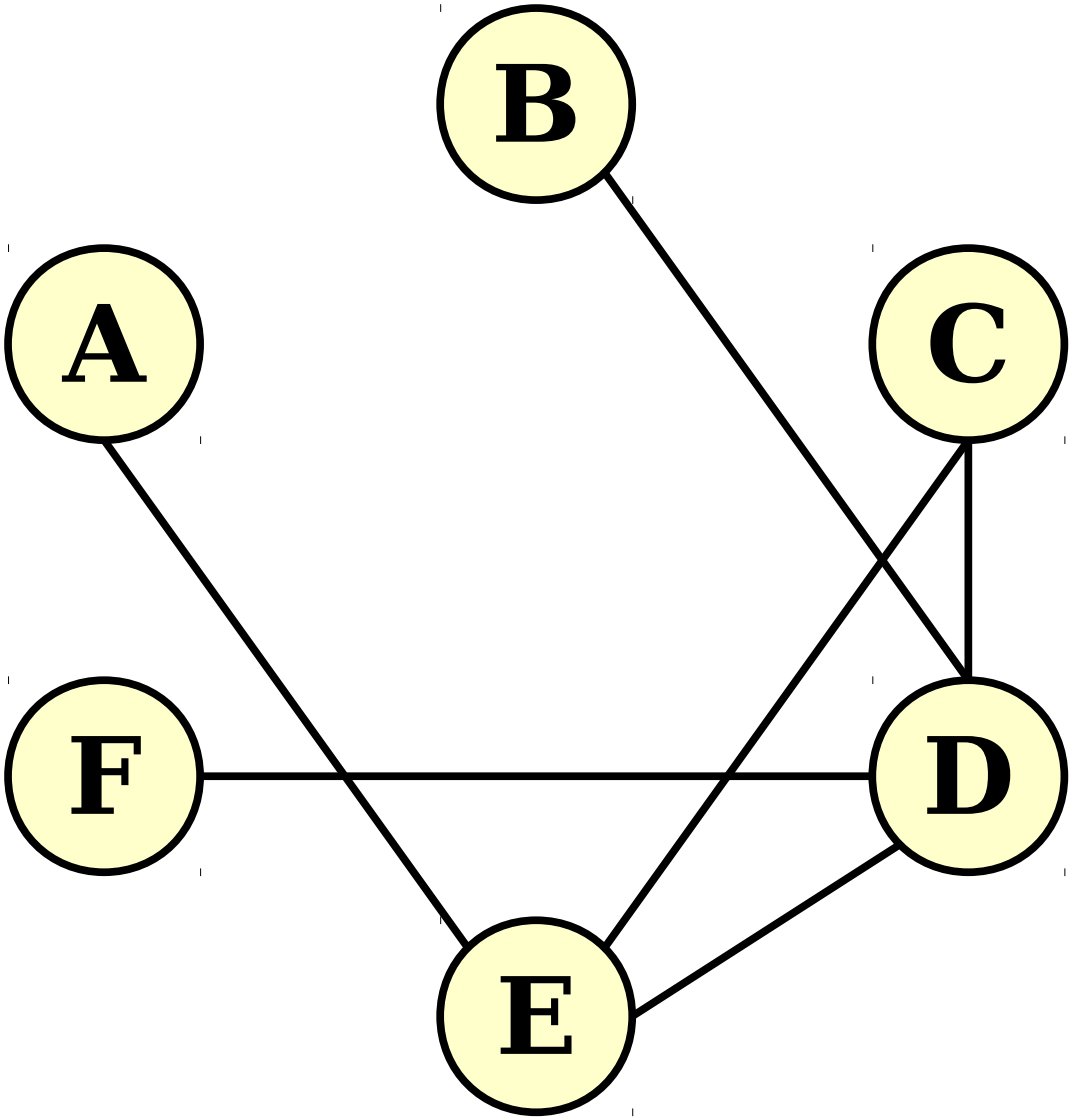
A Sample Problem

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

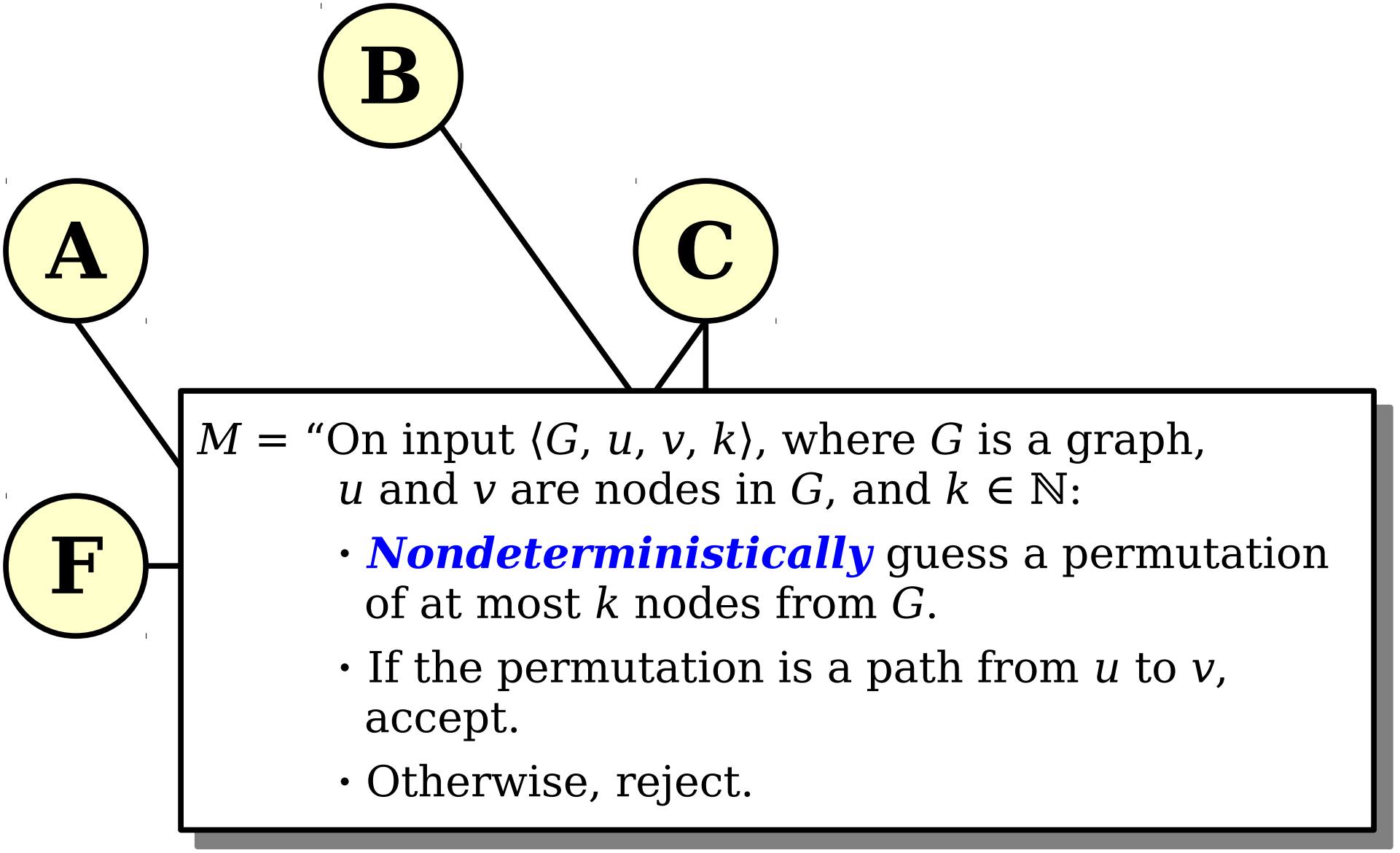
M = "On input $\langle S, k \rangle$, where S is a sequence of numbers and k is a natural number:

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

Another Problem



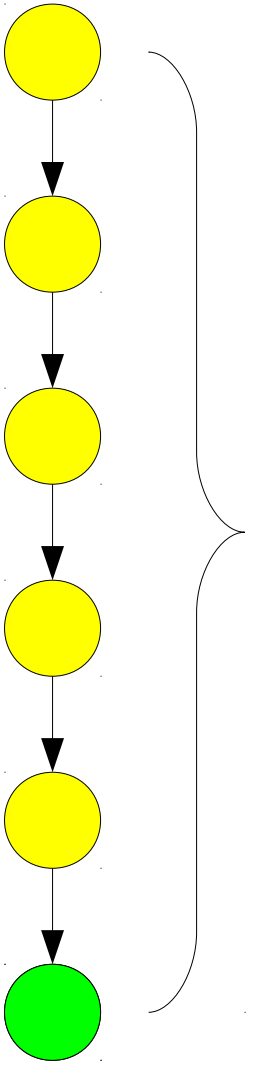
Another Problem



How do we measure NTM efficiency?

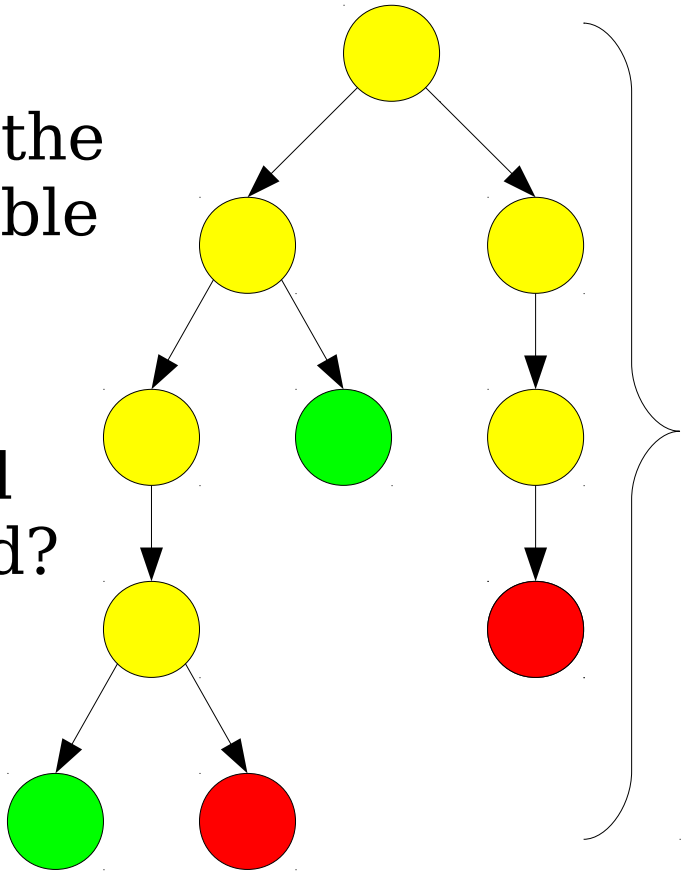
Analyzing NTMs

- 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.

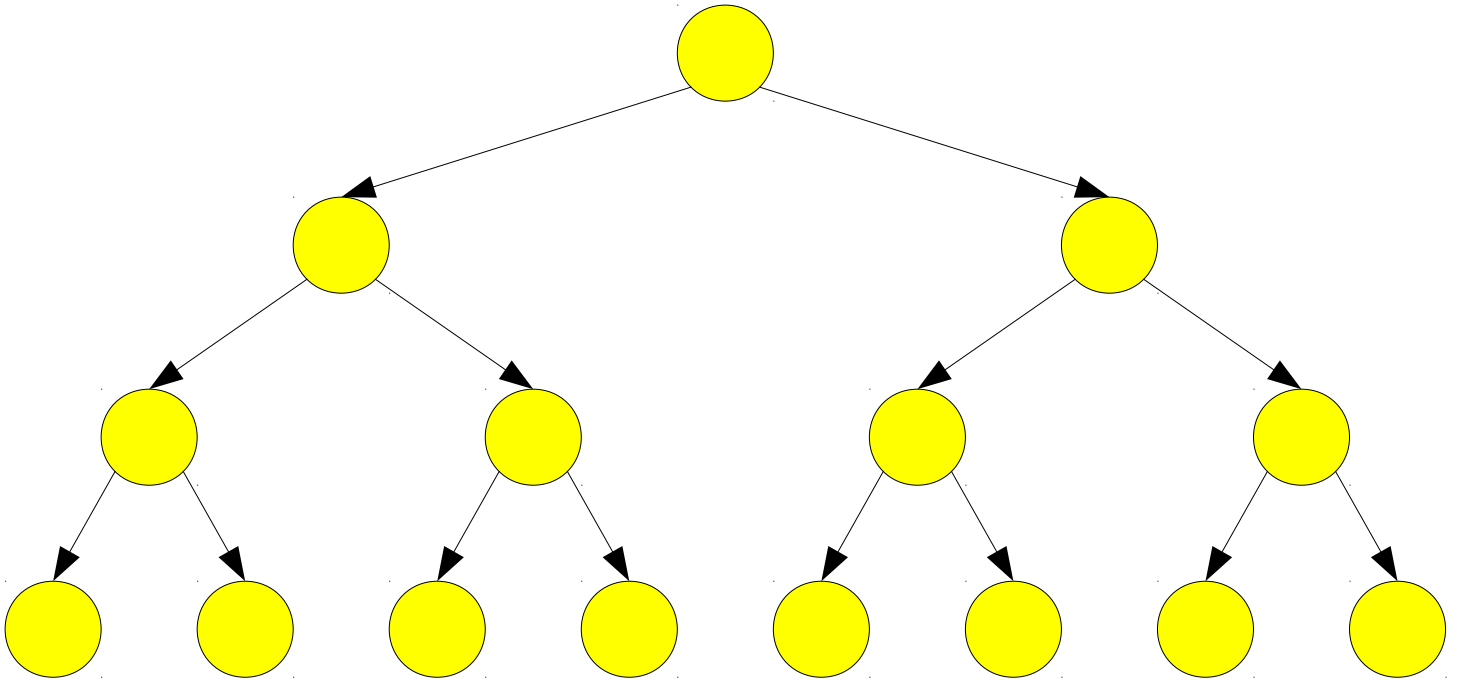


Analyzing NTMs

- 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 an $n^2 \times n^2$ 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 an $n^2 \times n^2$ 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 an $n^2 \times n^2$ 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 an $n^2 \times n^2$ 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 an $n^2 \times n^2$ 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 an $n^2 \times n^2$ 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 an $n^2 \times n^2$ 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 an $n^2 \times n^2$ 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 an $n^2 \times n^2$ 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 an $n^2 \times n^2$ 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 an $n^2 \times n^2$ 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 an $n^2 \times n^2$ 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 an $n^2 \times n^2$ 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: $\boldsymbol{O(n^2)}$

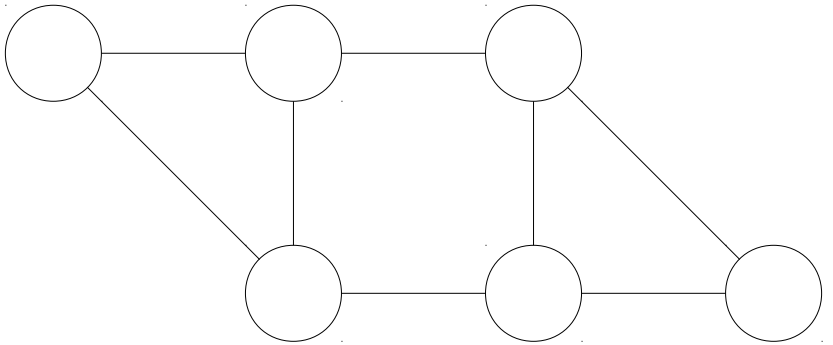
Total time required to check each row/column/box: $\boldsymbol{O(n^2)}$

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

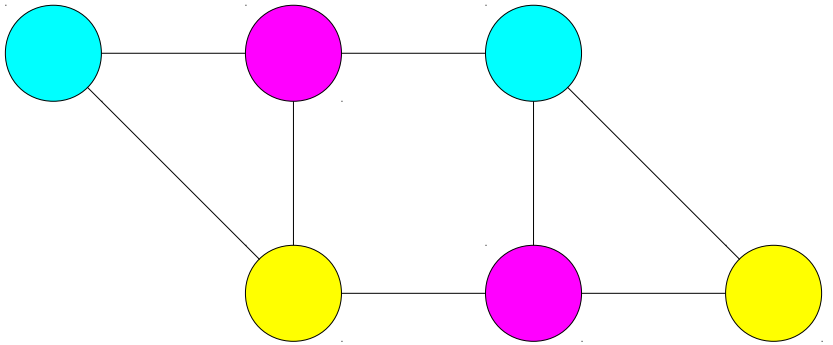
A Problem in **NP**

- A ***k-coloring*** of an undirected graph G is a way of assigning one if k colors to each node in G such that no two nodes joined by an edge have the same color.
  - Applications in compilers, cell phone towers, etc.
- **Question:** Given a graph G and a number k, is graph G k-colorable?



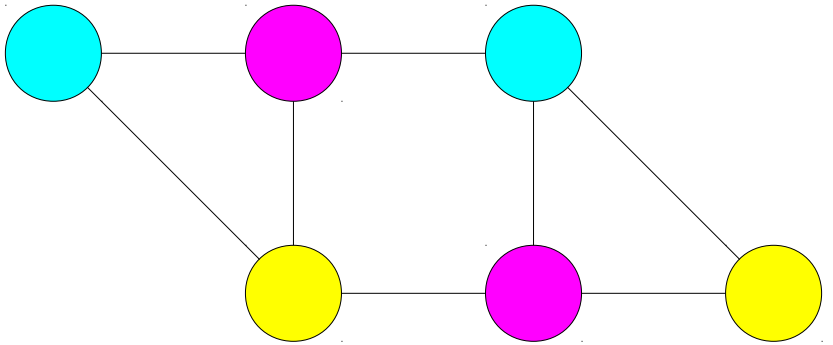
A Problem in **NP**

- A ***k-coloring*** of an undirected graph G is a way of assigning one if k colors to each node in G such that no two nodes joined by an edge have the same color.
  - Applications in compilers, cell phone towers, etc.
- **Question:** Given a graph G and a number k, is graph G k-colorable?



A Problem in **NP**

- A ***k-coloring*** of an undirected graph G is a way of assigning one if k colors to each node in G such that no two nodes joined by an edge have the same color.
  - Applications in compilers, cell phone towers, etc.
- **Question:** Given a graph G and a number k, is graph G k-colorable?
- 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 = { $\langle G, k \rangle$ | G is an undirected graph, $k \in \mathbb{N}$, and G is k-colorable. }

- In other words:

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

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**

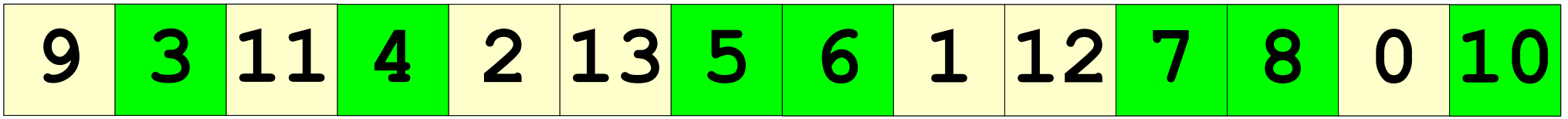
2	5	7	9	6	4	1	8	3
4	9	1	8	7	3	6	5	2
3	8	6	1	2	5	9	4	7
6	4	5	7	3	2	8	1	9
7	1	9	5	4	8	3	2	6
8	3	2	6	1	9	5	7	4
1	6	3	2	5	7	4	9	8
5	7	8	4	9	6	2	3	1
9	2	4	3	8	1	7	6	5

A Problem in **NP**

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

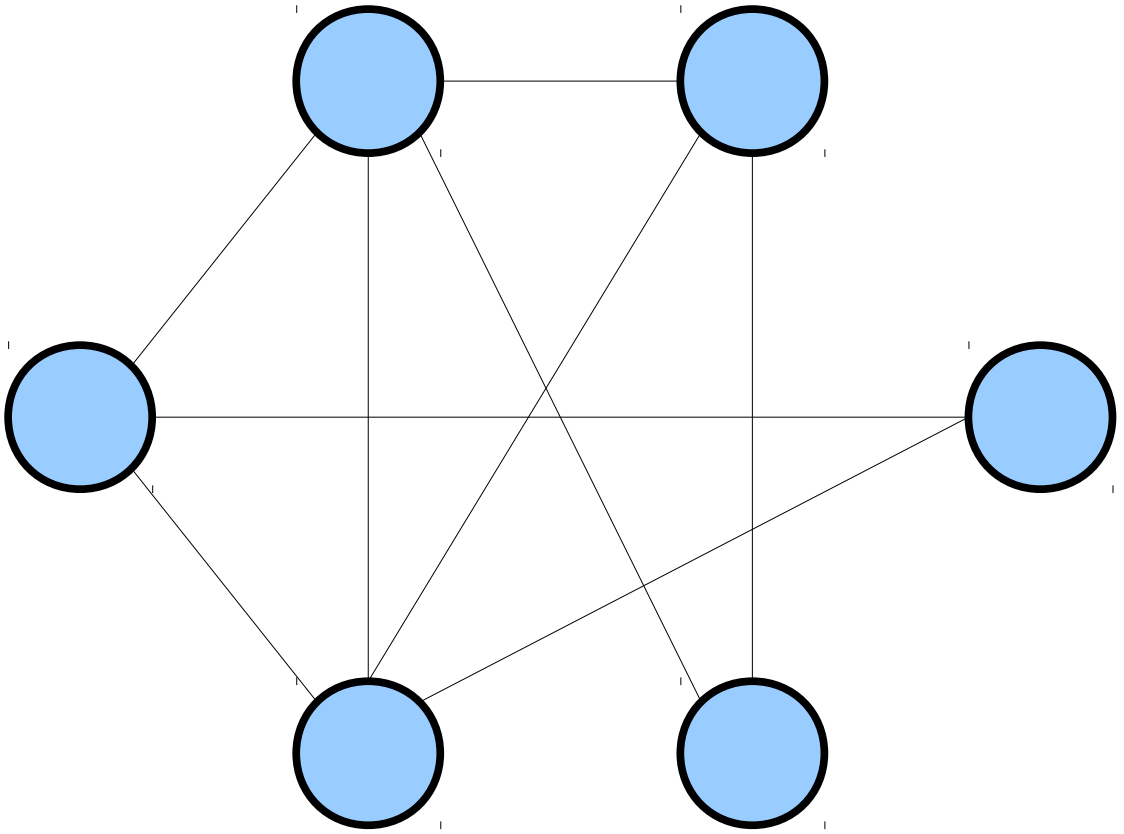
Is there an ascending subsequence of length at least 7?

A Problem in **NP**



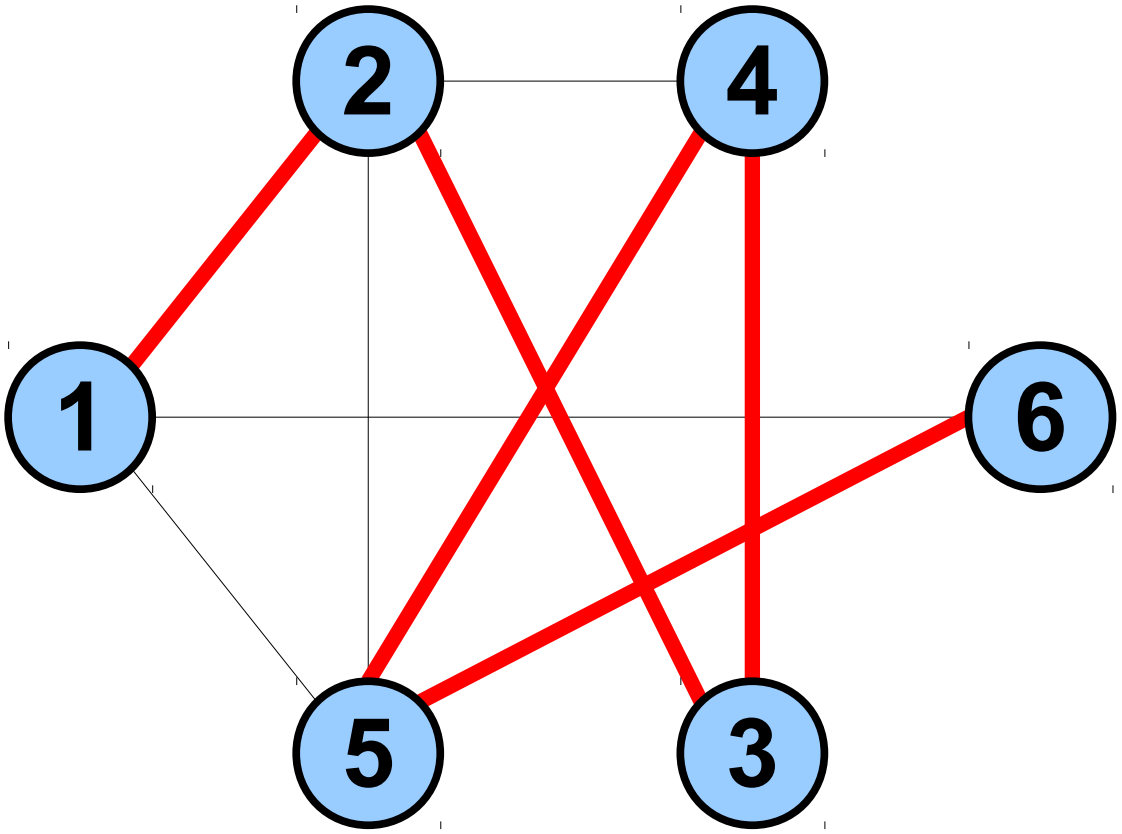
Is there an ascending subsequence of length at least 7?

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?

Verifiers

- Recall that a ***verifier*** for L is a deterministic TM V such that
  - V halts on all inputs.
  - $w \in L$ iff $\exists c \in \Sigma^*$. V accepts $\langle w, c \rangle$.

- **Theorem:** $L \in \mathbf{RE}$ iff there is a verifier for L.

Polynomial-Time Verifiers

- A ***polynomial-time verifier*** for L is a deterministic TM V such that
  - V halts on all inputs.
  - $w \in L$ iff $\exists c \in \Sigma^*$. V accepts $\langle w, c \rangle$.
  - V's runtime is a polynomial in $|w|$.
- **Theorem:** $L \in \mathbf{NP}$ iff there is a polynomial-time verifier for L.

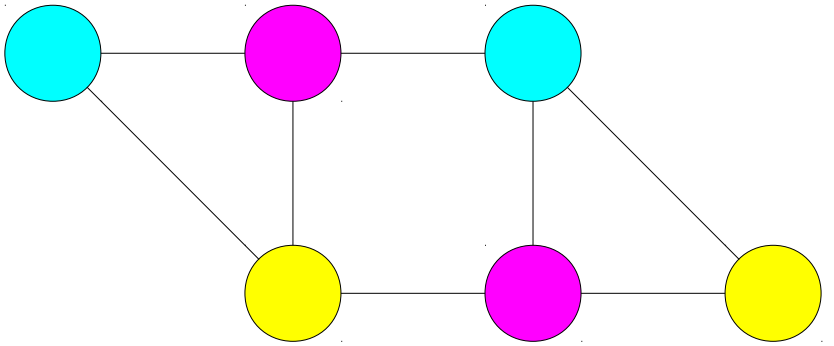
A Problem in **NP**

- 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."



The Verifier Definition of **NP**

- ***Theorem:*** If $L \in$ **NP**, there is a polynomial-time verifier for it.
- ***Proof sketch:*** Use the general construction that turns an TM into a verifier, and argue that the overall construction runs in polynomial time. ■

The Verifier Definition of **NP**

- ***Theorem:*** If there is a polynomial-time verifier V for L, then $L \in \mathbf{NP}$.
- ***Proof idea:*** Build an NTM that nondeterministically guesses a certificate, then deterministically runs the verifier to check it. Then, argue that the NTM runs in nondeterministic polynomial time.

And now…

The
Most Important Question
in
Theoretical Computer Science

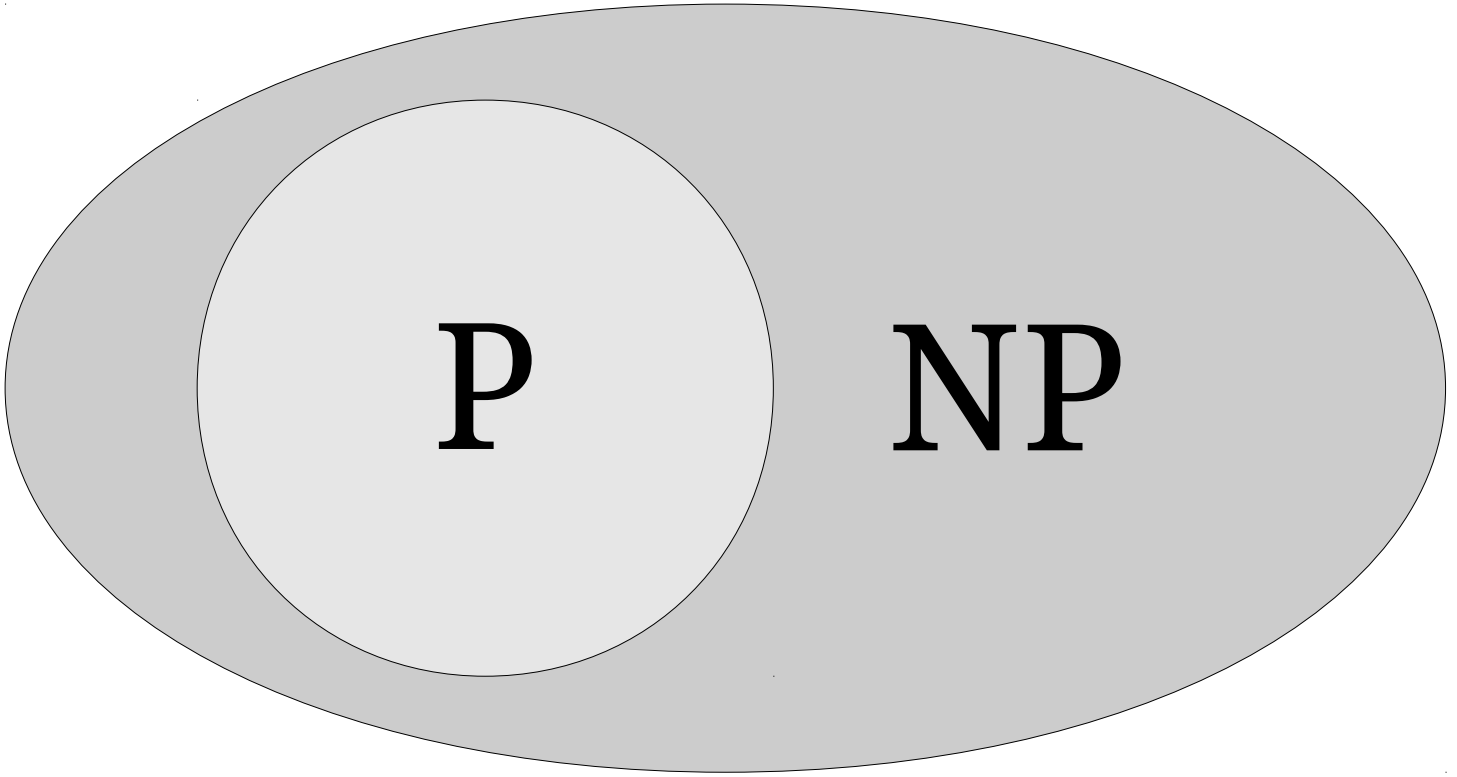
What is the connection between **P** and **NP**?

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

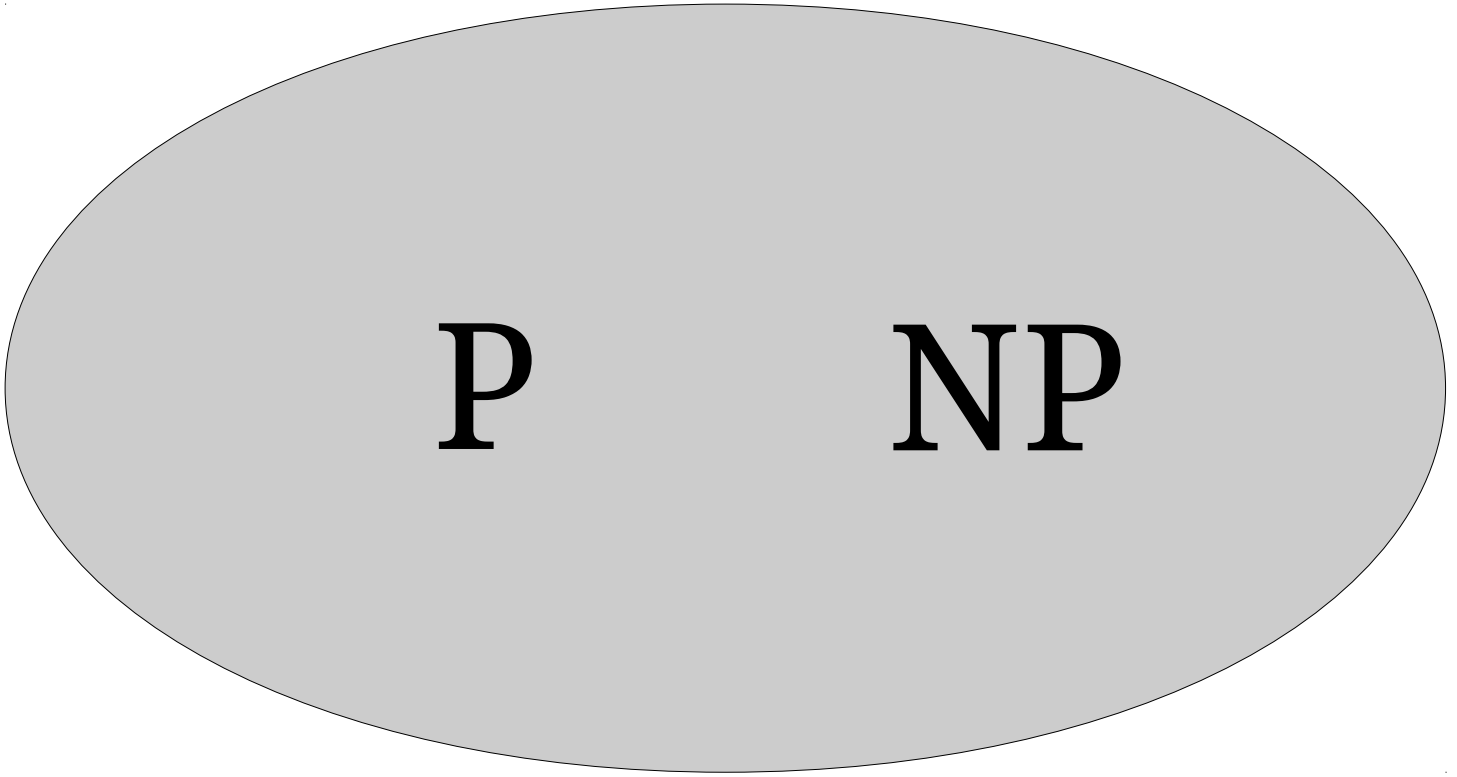
$$\mathbf{NP} = \{ L \mid \text{There is a nondeterministic polynomial-time decider for } L \}$$

$$\mathbf{P} \subseteq \mathbf{NP}$$

Which Picture is Correct?



Which Picture is Correct?



Does **P** = **NP**?

$\mathbf{P \stackrel{?}{=} NP}$

- The $\mathbf{P \stackrel{?}{=} NP}$ question is the most important question in theoretical computer science.
- With the verifier definition of **NP**, one way of phrasing this question is

If a solution to a problem can be ***checked*** efficiently, can that problem be ***solved*** efficiently?

- An answer either way will give fundamental insights into the nature of computation.

Why This Matters

- The following problems are known to be efficiently verifiable, but have no known efficient solutions:
  - Determining whether an electrical grid can be built to link up some number of houses for some price (Steiner tree problem).
  - Determining whether a simple DNA strand exists that multiple gene sequences could be a part of (shortest common supersequence).
  - Determining the best way to assign hardware resources in a compiler (optimal register allocation).
  - Determining the best way to distribute tasks to multiple workers to minimize completion time (job scheduling).
  - ***And many more***.
- If **P = NP**, ***all*** of these problems have efficient solutions.
- If **P ≠ NP**, ***none*** of these problems have efficient solutions.

Why This Matters

- If **P = NP**:
  - A huge number of seemingly difficult problems could be solved efficiently.
  - Our capacity to solve many problems will scale well with the size of the problems we want to solve.
- If **P ≠ NP**:
  - Enormous computational power would be required to solve many seemingly easy tasks.
  - Our capacity to solve problems will fail to keep up with our curiosity.

What We Know

- Resolving **P** $\stackrel{?}{=}$ **NP** has proven ***extremely difficult***.
- In the past 35 years:
  - Not a single correct proof either way has been found.
  - Many types of proofs have been shown to be insufficiently powerful to determine whether **P** $\stackrel{?}{=}$ **NP**.
  - A majority of computer scientists believe **P** ≠ **NP**, but this isn't a large majority.
- Interesting read: Interviews with leading thinkers about **P** $\stackrel{?}{=}$ **NP**:
  - http://web.ing.puc.cl/~jabaier/iic2212/poll-1.pdf

The Million-Dollar Question

The Clay Mathematics Institute has offered a **$1,000,000 prize** to anyone who proves or disproves **P = NP**.

The Million-Dollar Question

The Clay Mathematics Institute has offered a **$1,000,000 prize** to anyone who proves or disproves **P = NP**.

Time-Out for Announcements!

Problem Set Logistics

- Problem Set 8 is due the Monday we get back from break.
- Problem Set 9 (weighted at only 2% of your grade) will go out on that Monday and be due on the last day of class.
  - Due to university policies, ***no late submissions*** will be accepted, even if you still have your late period.
- Problem Set 6 was just graded and should be returned soon.
- Problem Set 7 will be returned by Wednesday of the break.

ACM Richard Tapia Conference

- “This year, the CS department is trying to send a group of students to the ACM Richard Tapia Celebration of Diversity in Computing (Richard Tapia Conference). It's a conference designed to allow underrepresented minorities in CS to network and build skills. We're trying to get a sense of how many people would be interested in attending. It's not set in stone yet, but registration, flights, and lodging would be covered.”
- Interested? Fill out the interest form at http://goo.gl/forms/pyS5H4EpuL, or contact Khalil Griffin (khalilsg@stanford.edu).

Your Questions!

“Do you believe that **P = NP**? How would either result impact your life?”

“Vi or Emacs?”

“I love CS103! What other courses are similar in topic that you would recommend?”

“In your opinion, what's the worst runtime?”

Back to CS103!

What do we know about $\mathbf{P} \stackrel{?}{=} \mathbf{NP}$?

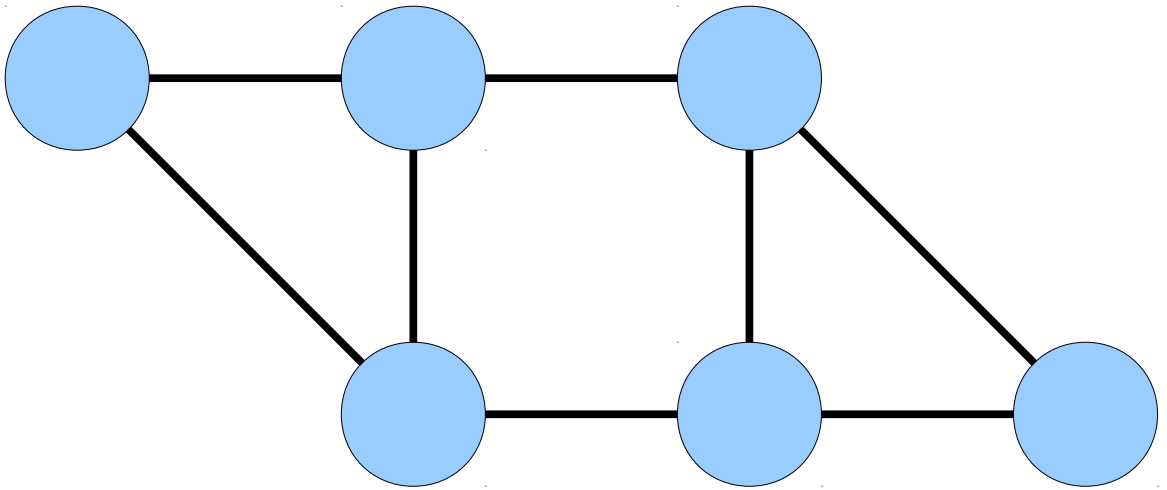
Reducibility

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.

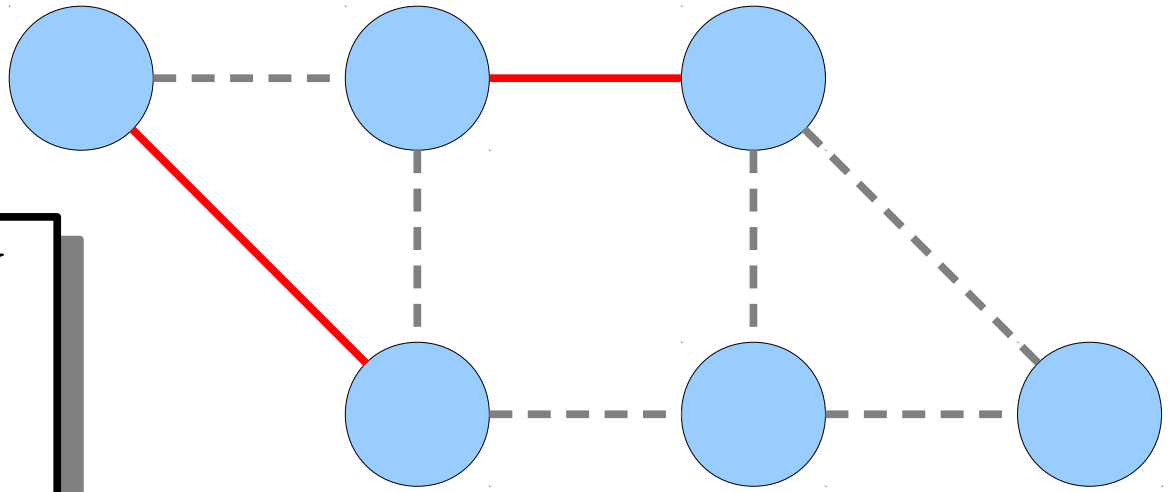
Maximum Matching

- 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

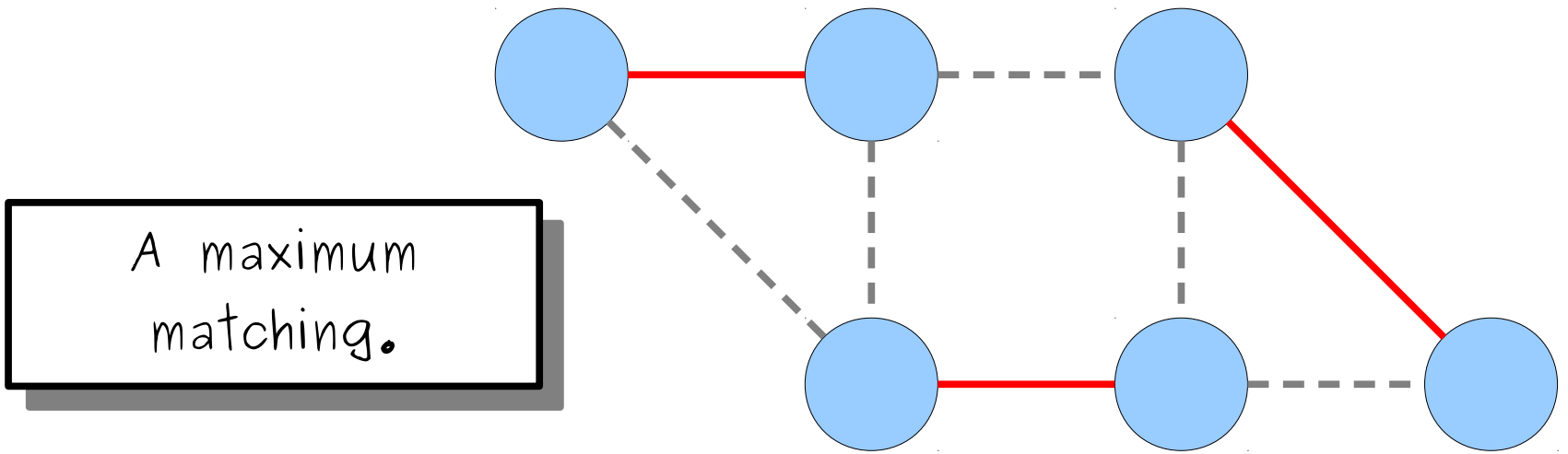
- Given an undirected graph G, a ***matching*** in G is a set of edges such that no two edges share an endpoint.
- A ***maximum matching*** is a matching with the largest number of edges.



A matching, but not a maximum matching.

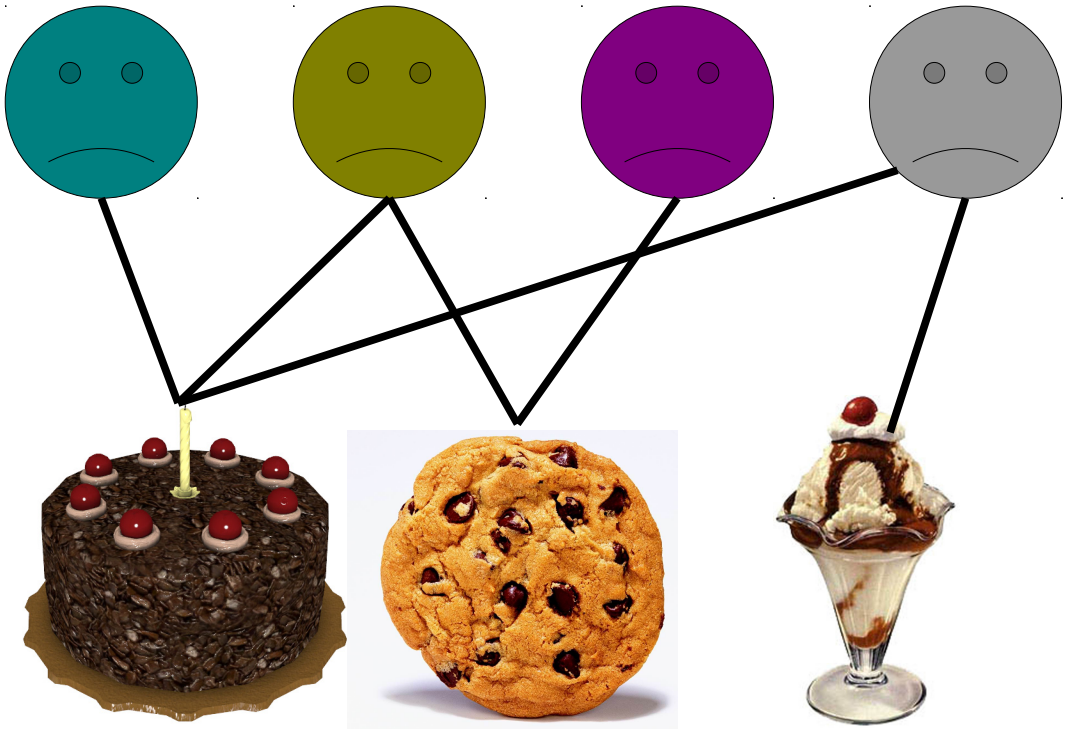
Maximum Matching

- 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.



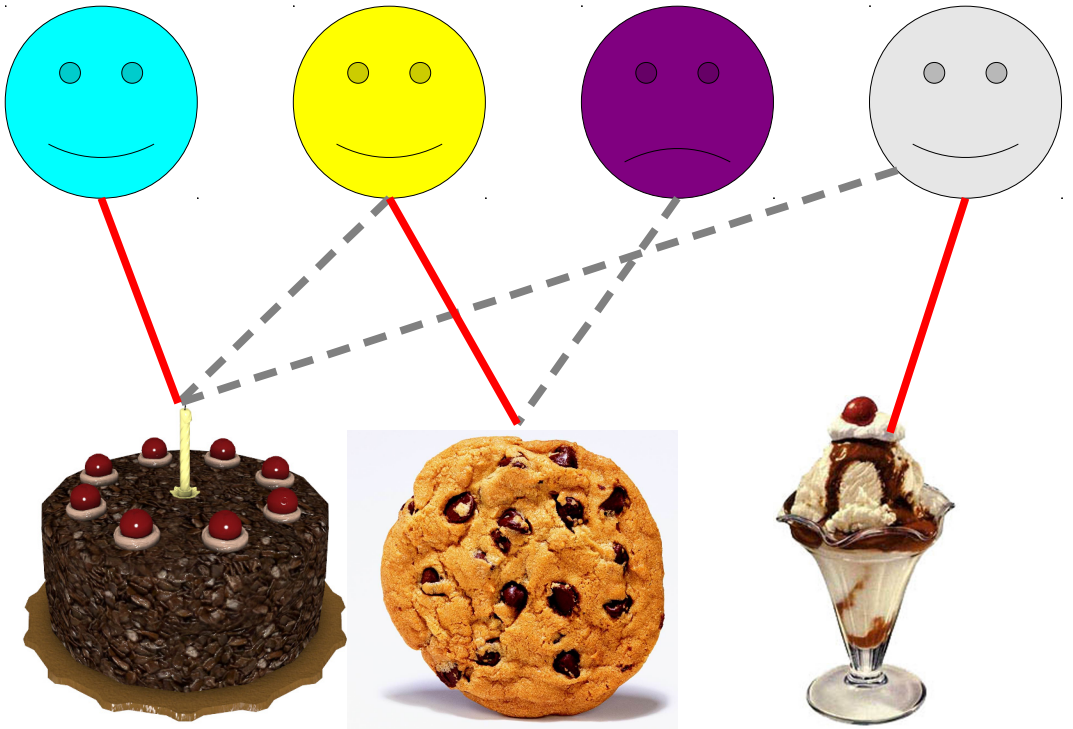
Maximum Matching

- 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.



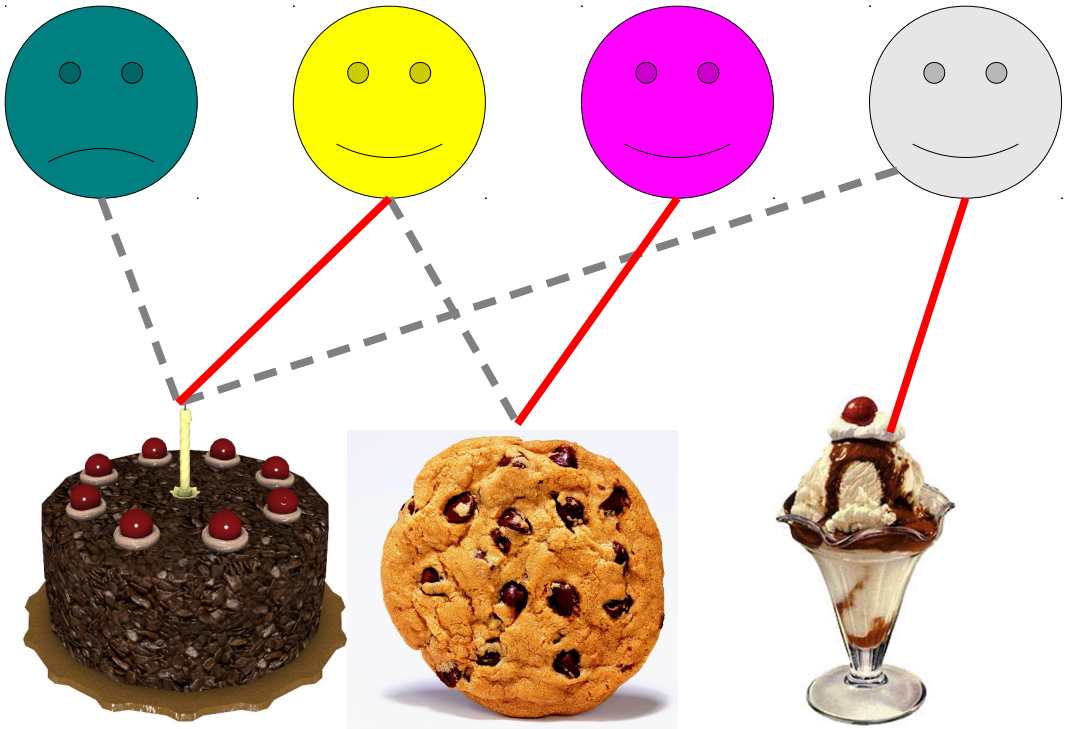
Maximum Matching

- 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

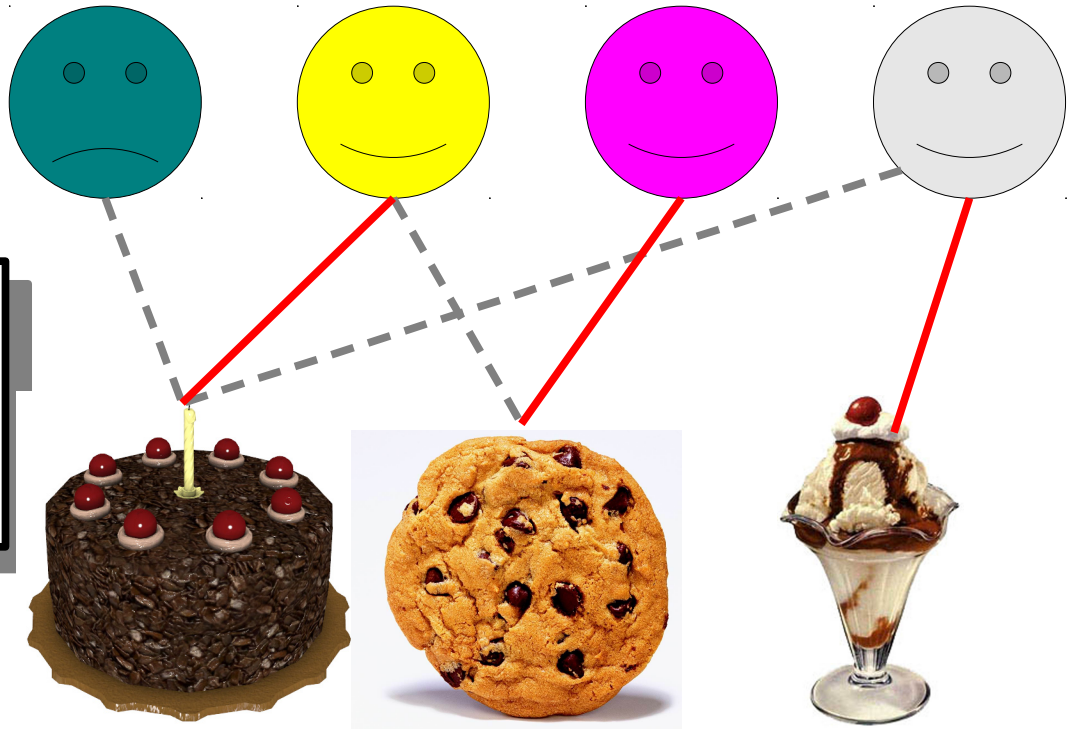
- Given an undirected graph G, a ***matching*** in G is a set of edges such that no two edges share an endpoint.
- A ***maximum matching*** is a matching with the largest number of edges.



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.

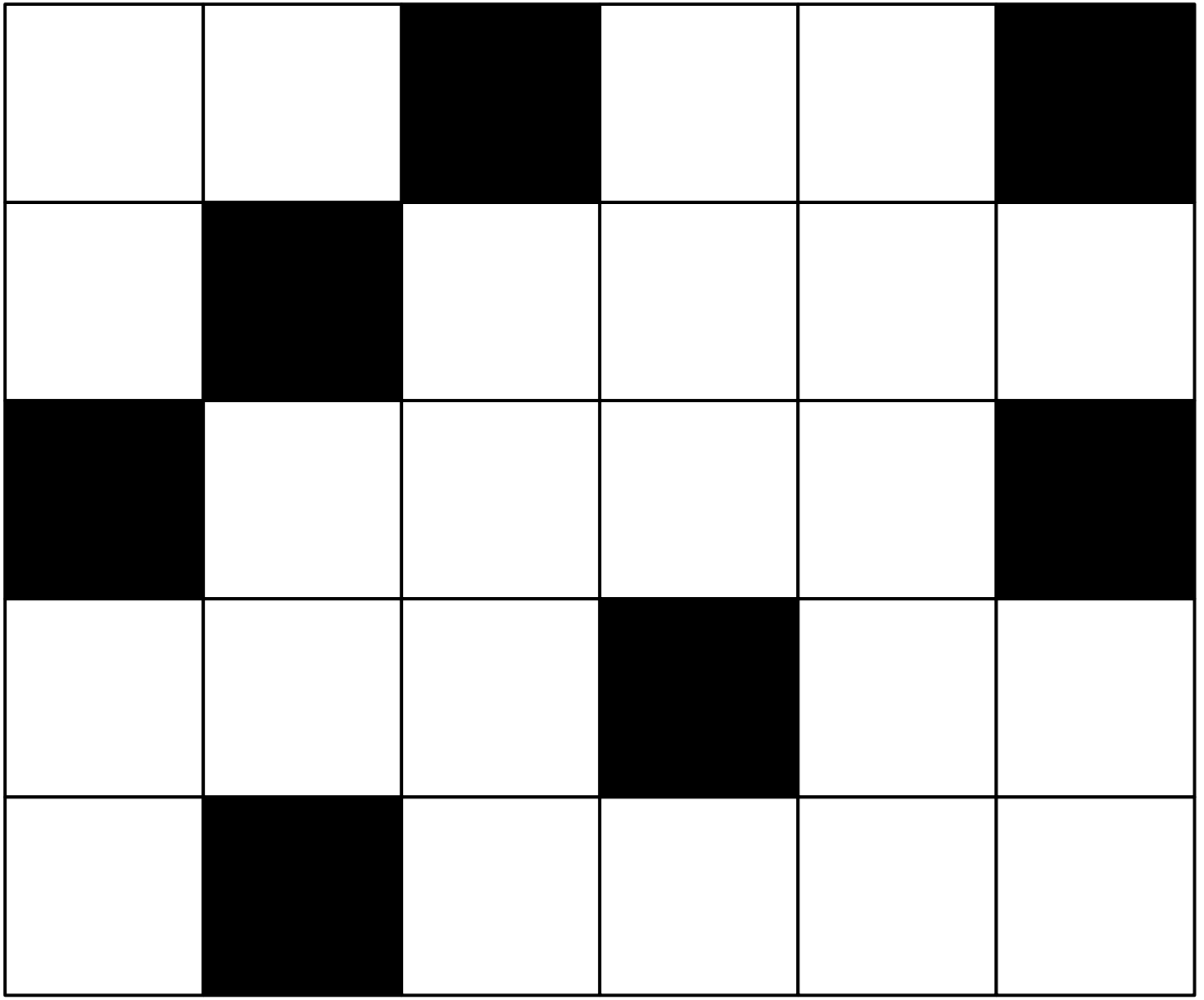
Maximum matchings are not necessarily unique.



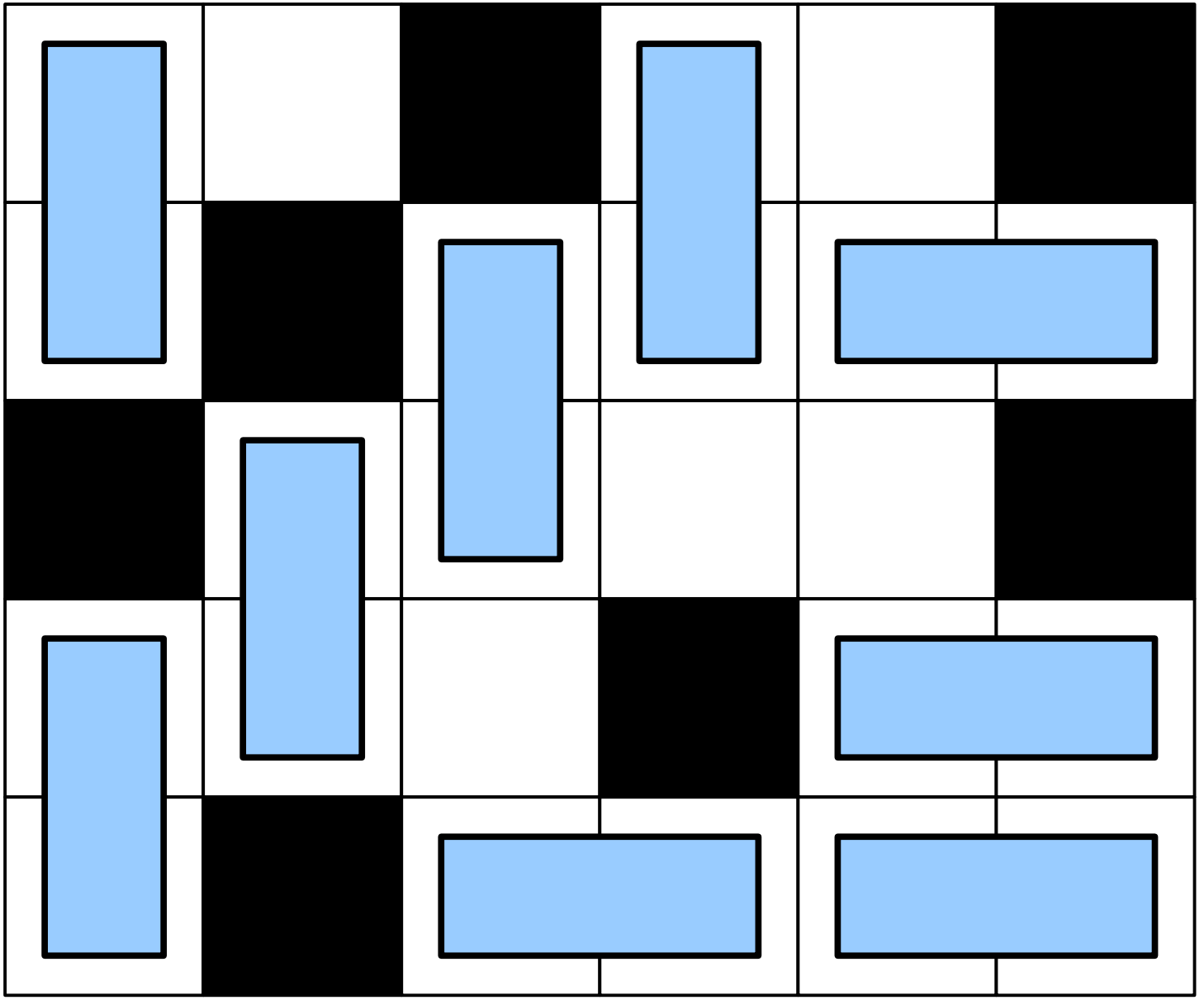
Maximum Matching

- 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?

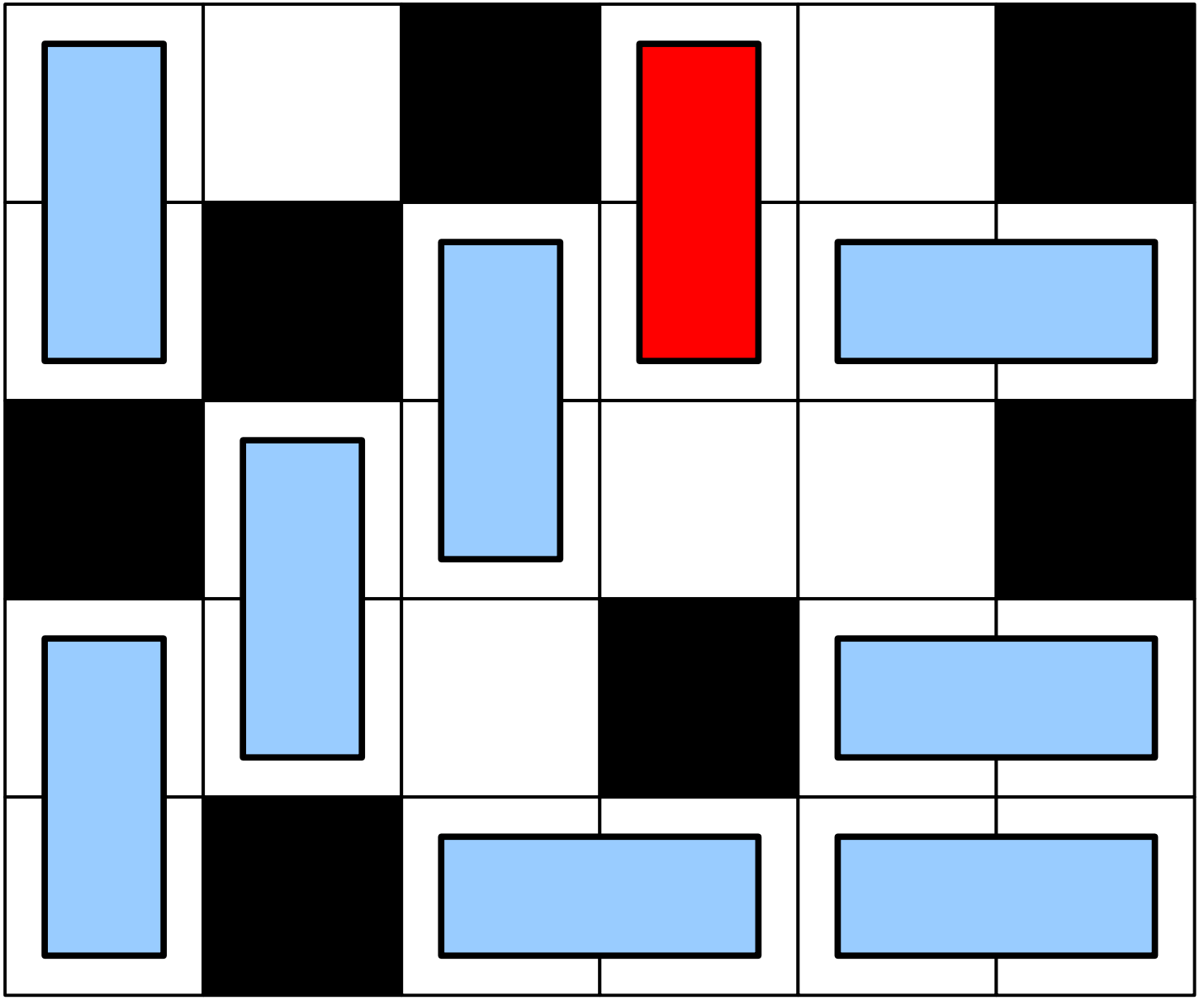
Domino Tiling



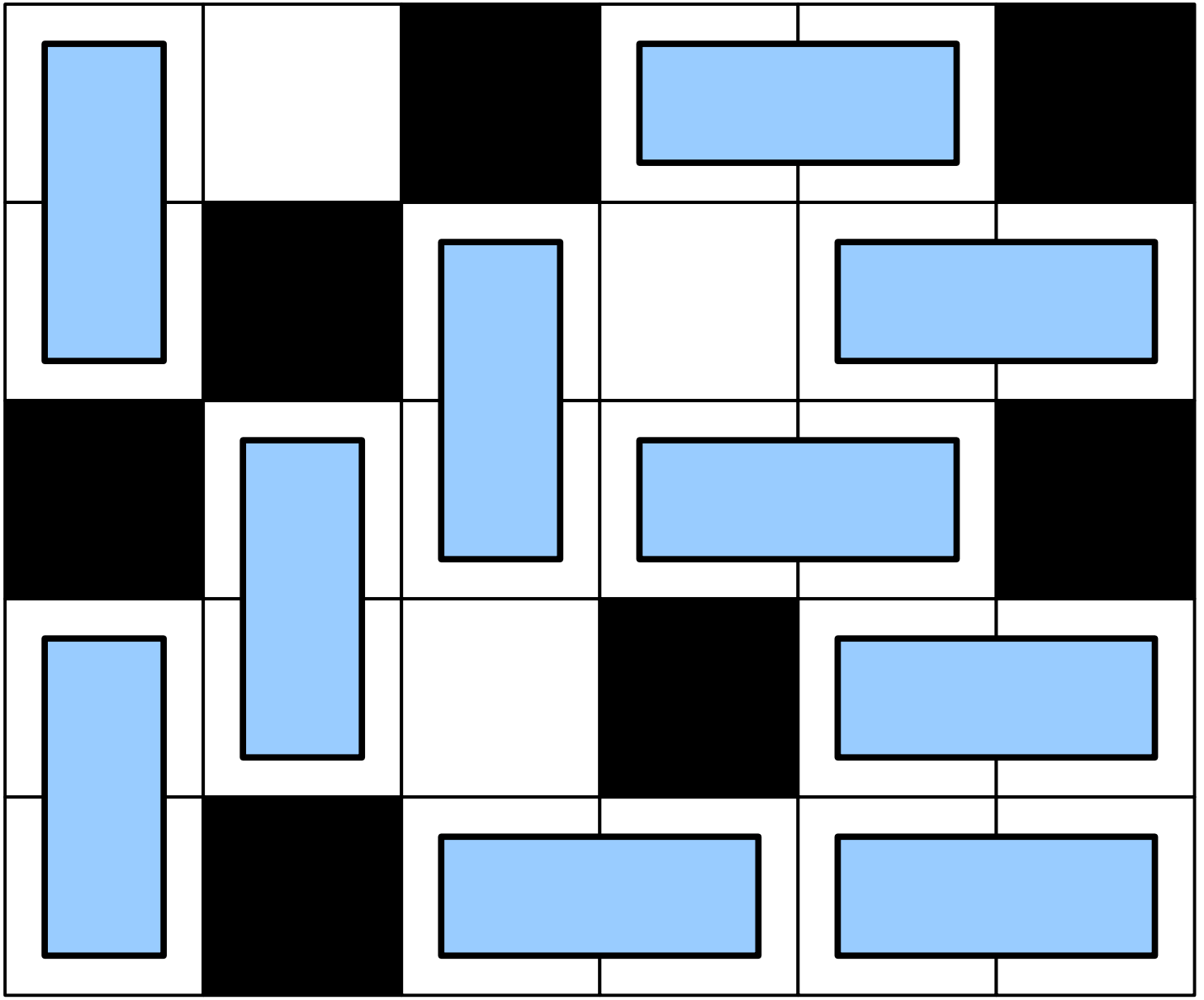
Domino Tiling



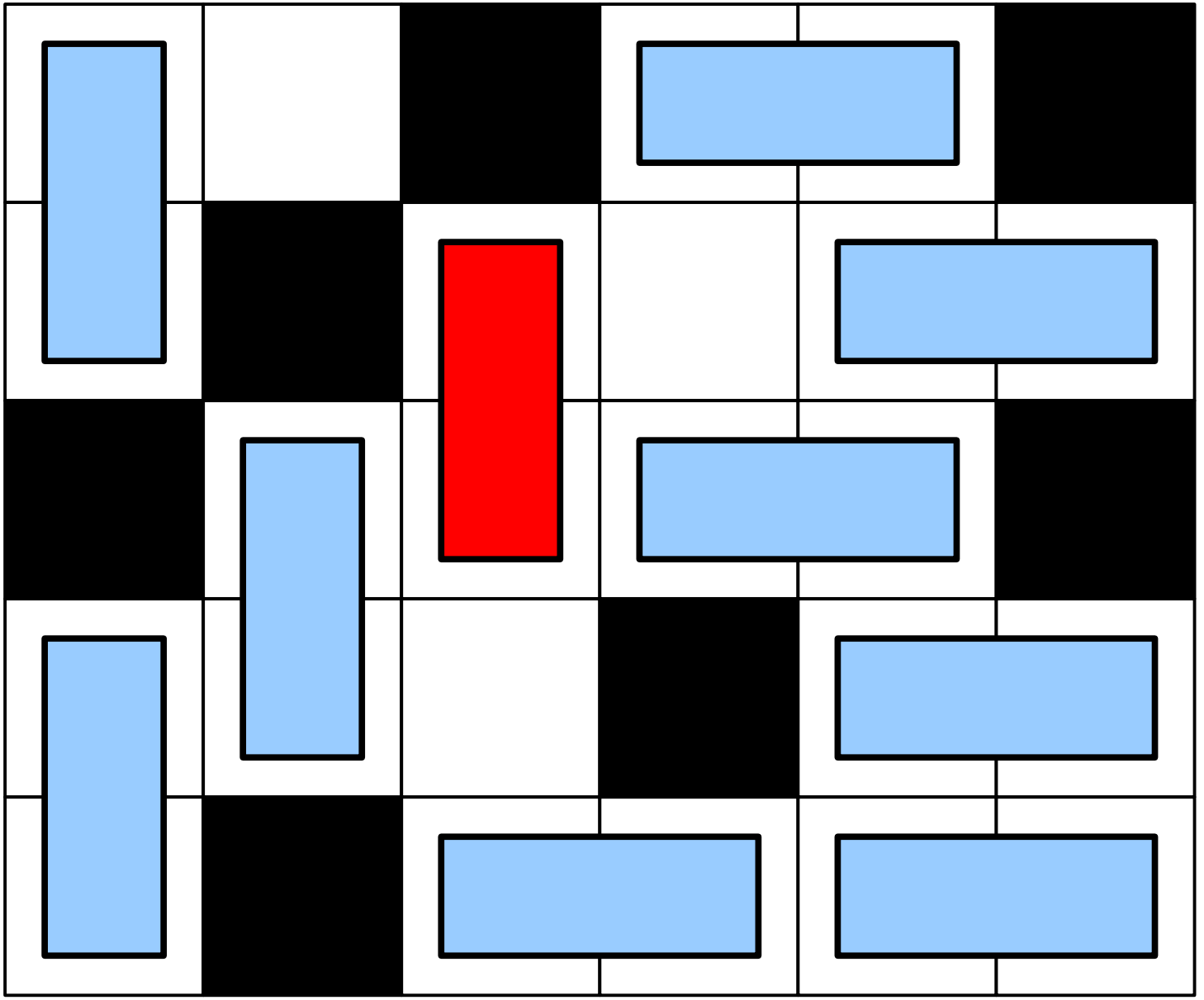
Domino Tiling



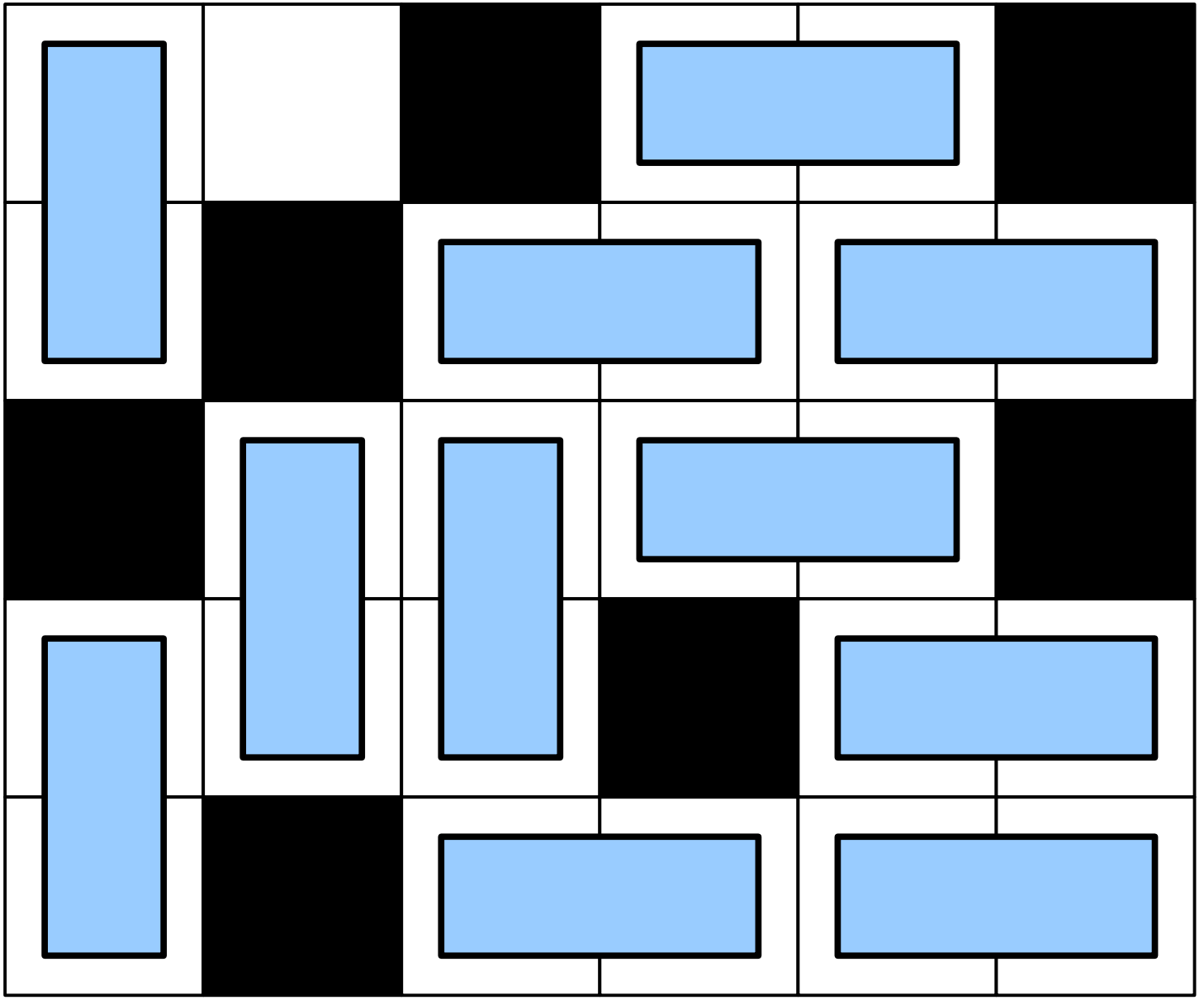
Domino Tiling



Domino Tiling



Domino Tiling



A Domino Tiling Reduction

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

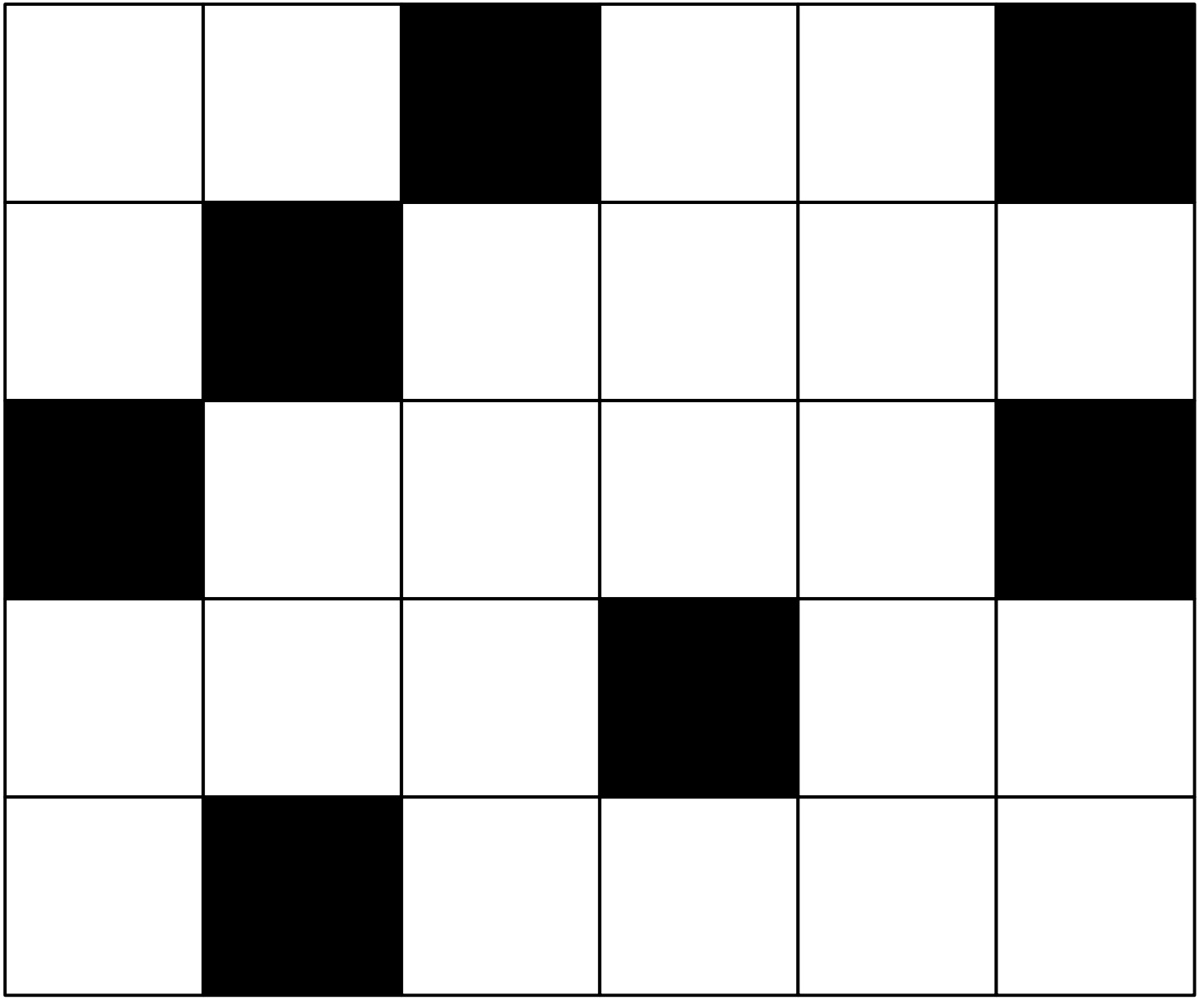
  $MATCHING = \{ \langle G, k \rangle \mid G$ is an undirected graph with a matching of size at least $k \}$

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

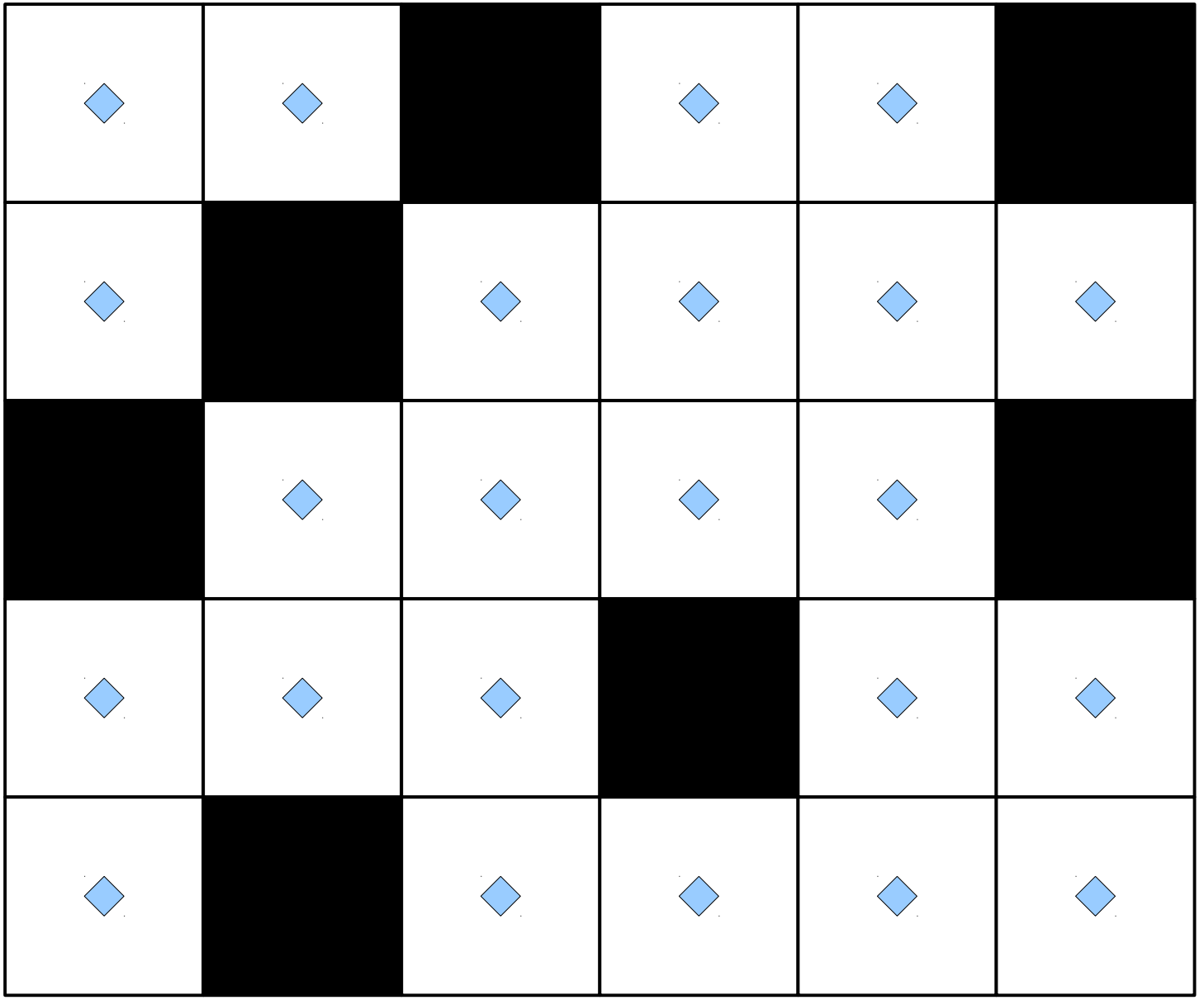
  $DOMINO = \{ \langle D, k \rangle \mid D$ is a grid and k nonoverlapping dominoes can be placed on D. $\}$

- We'll use the fact that $MATCHING \in \mathbf{P}$ to prove that $DOMINO \in \mathbf{P}$.

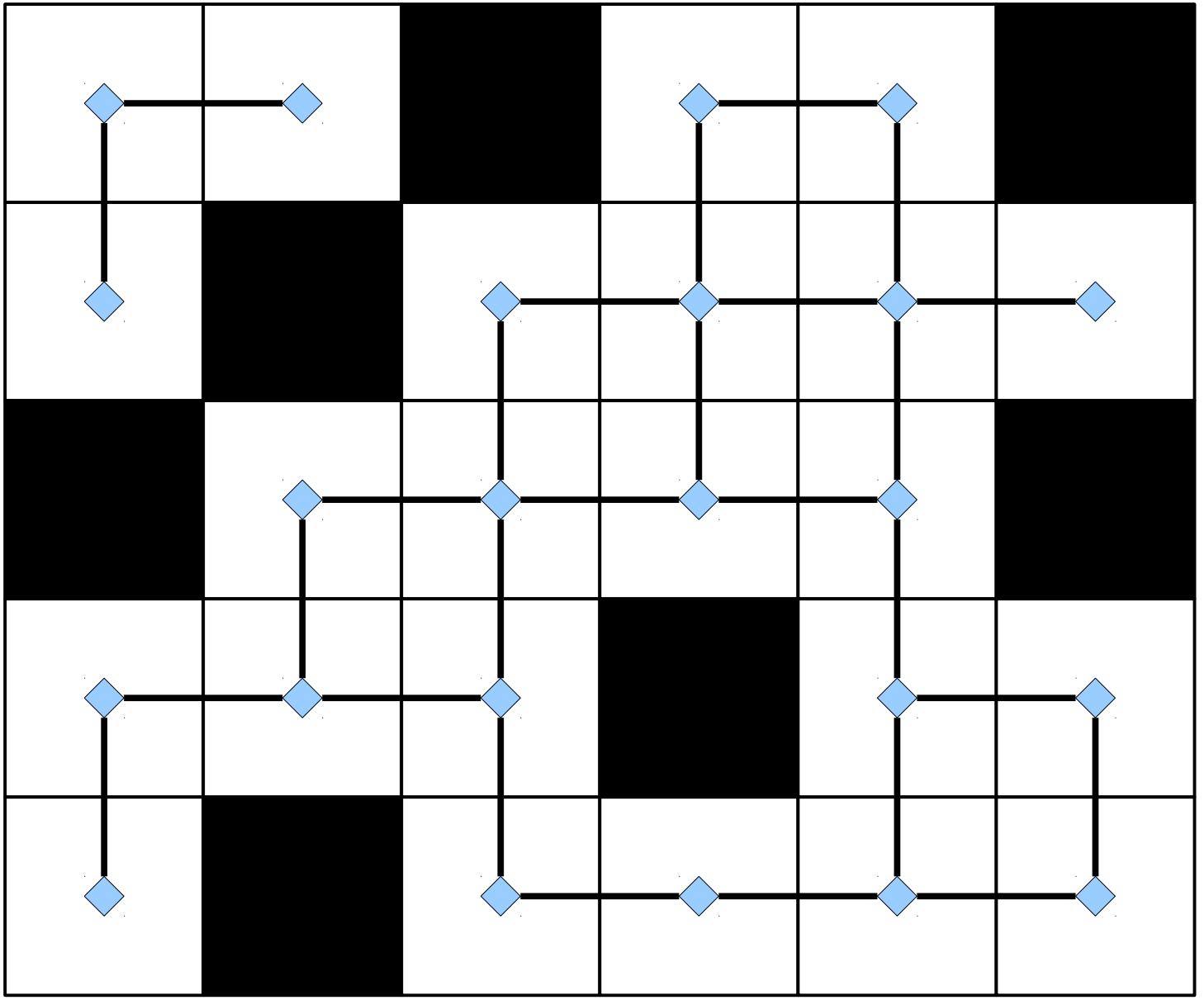
Solving Domino Tiling



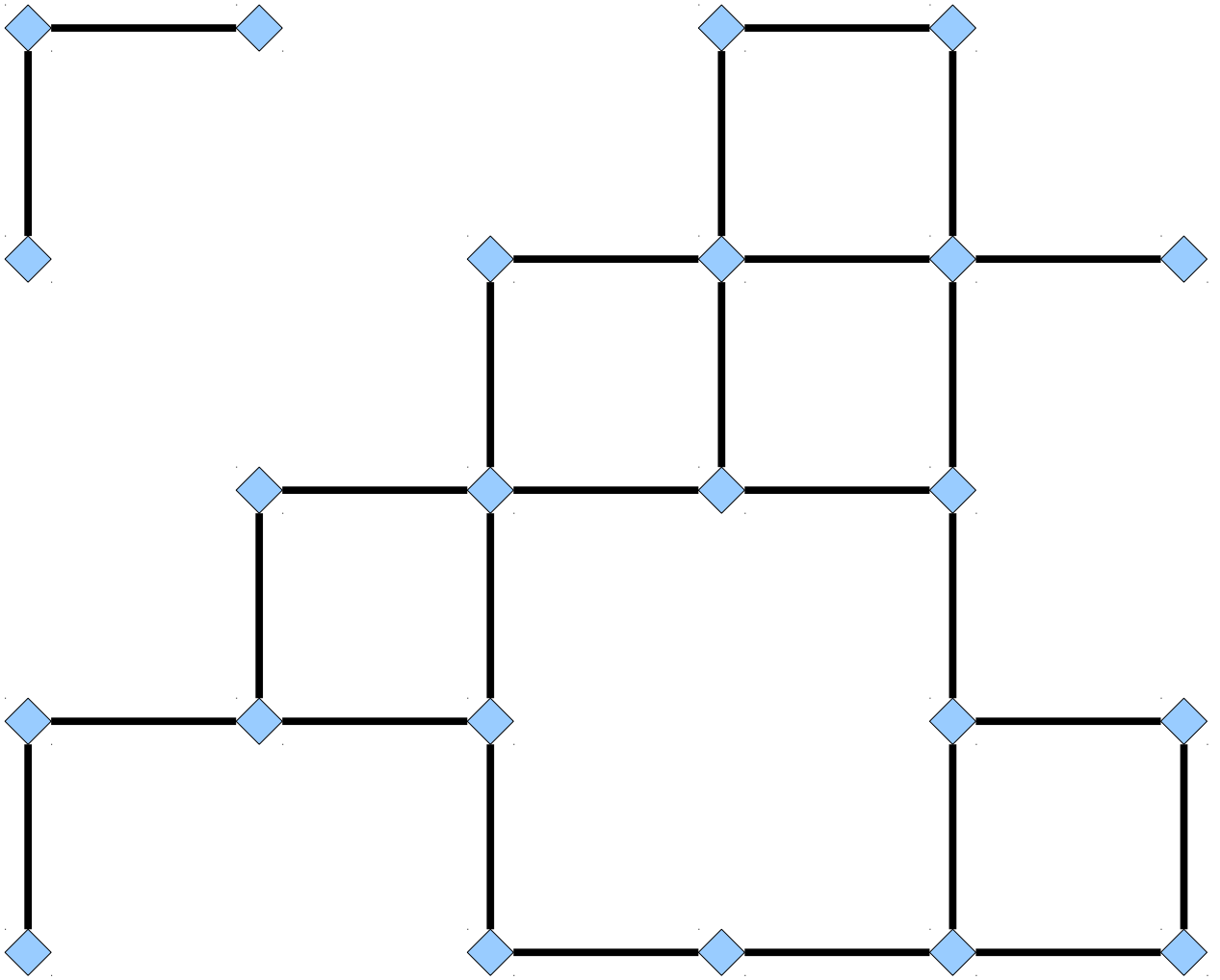
Solving Domino Tiling



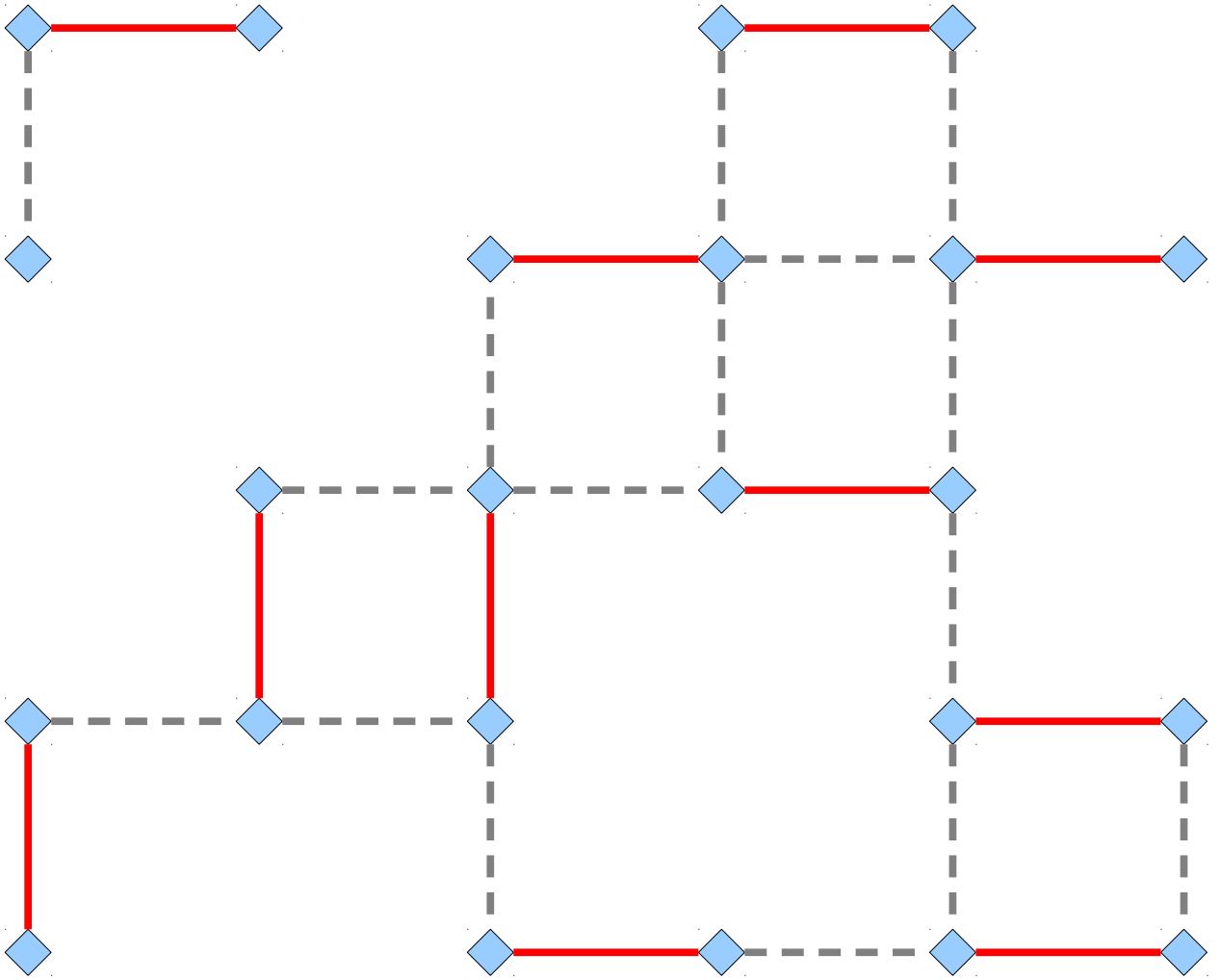
Solving Domino Tiling



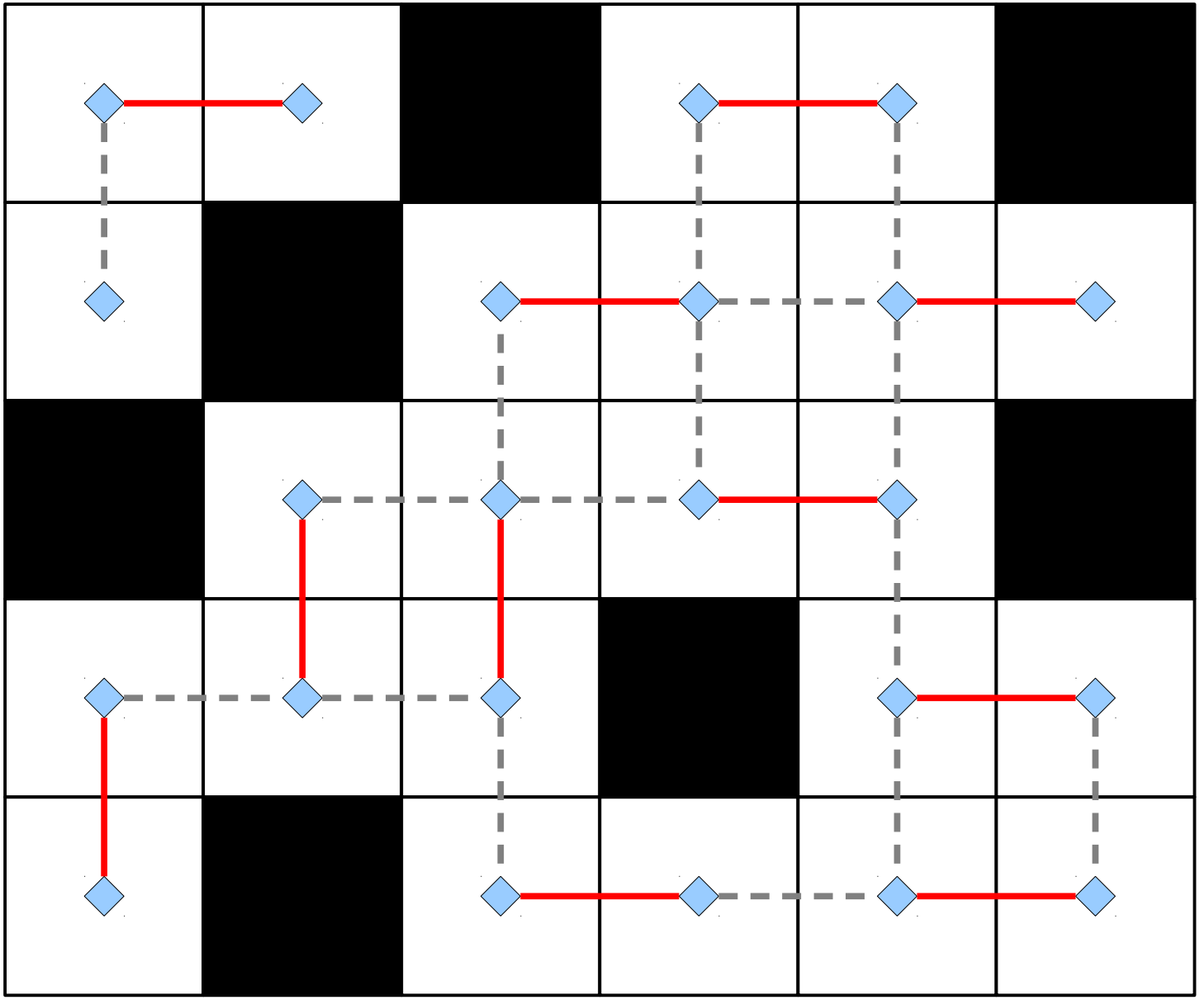
Solving Domino Tiling



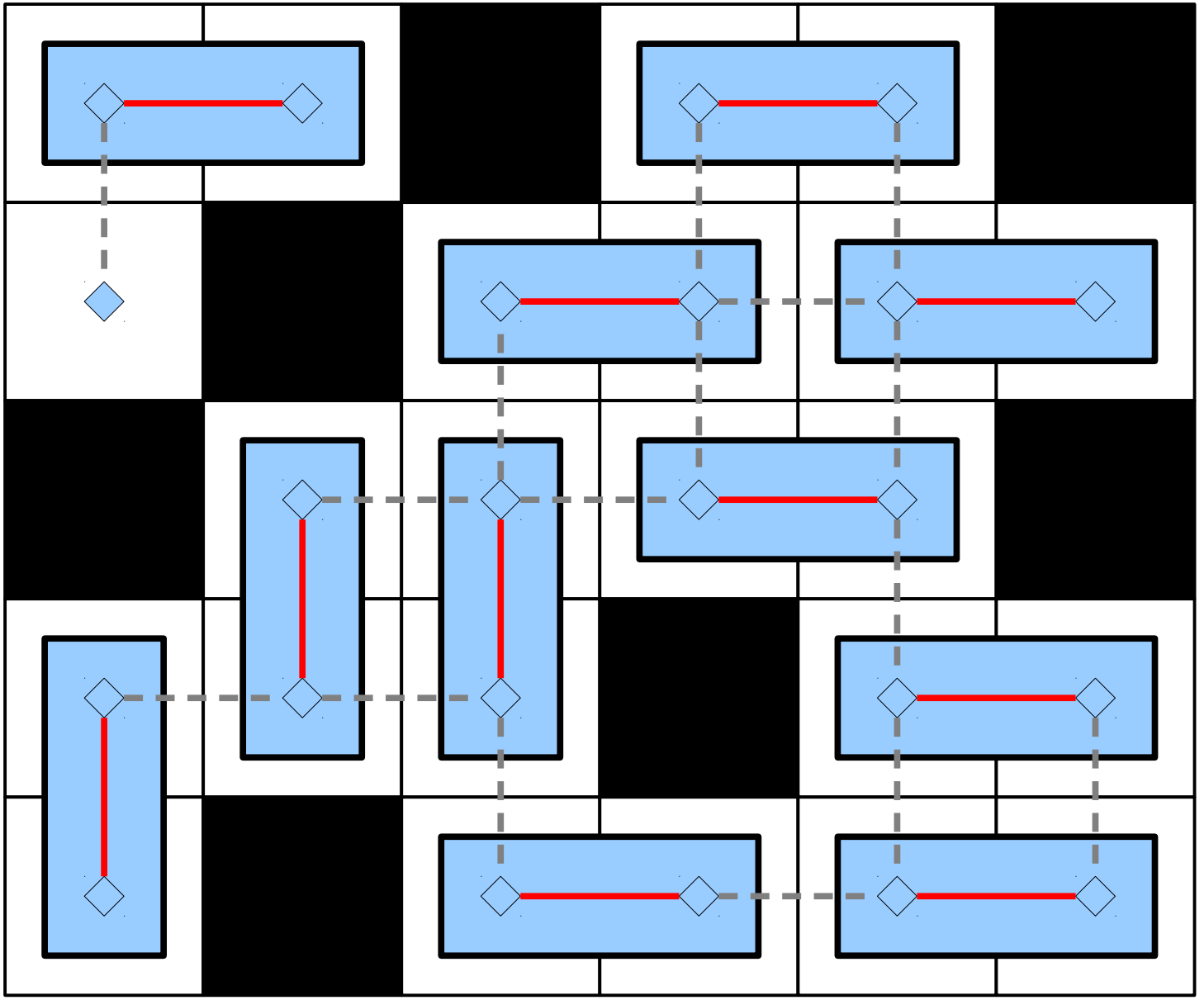
Solving Domino Tiling



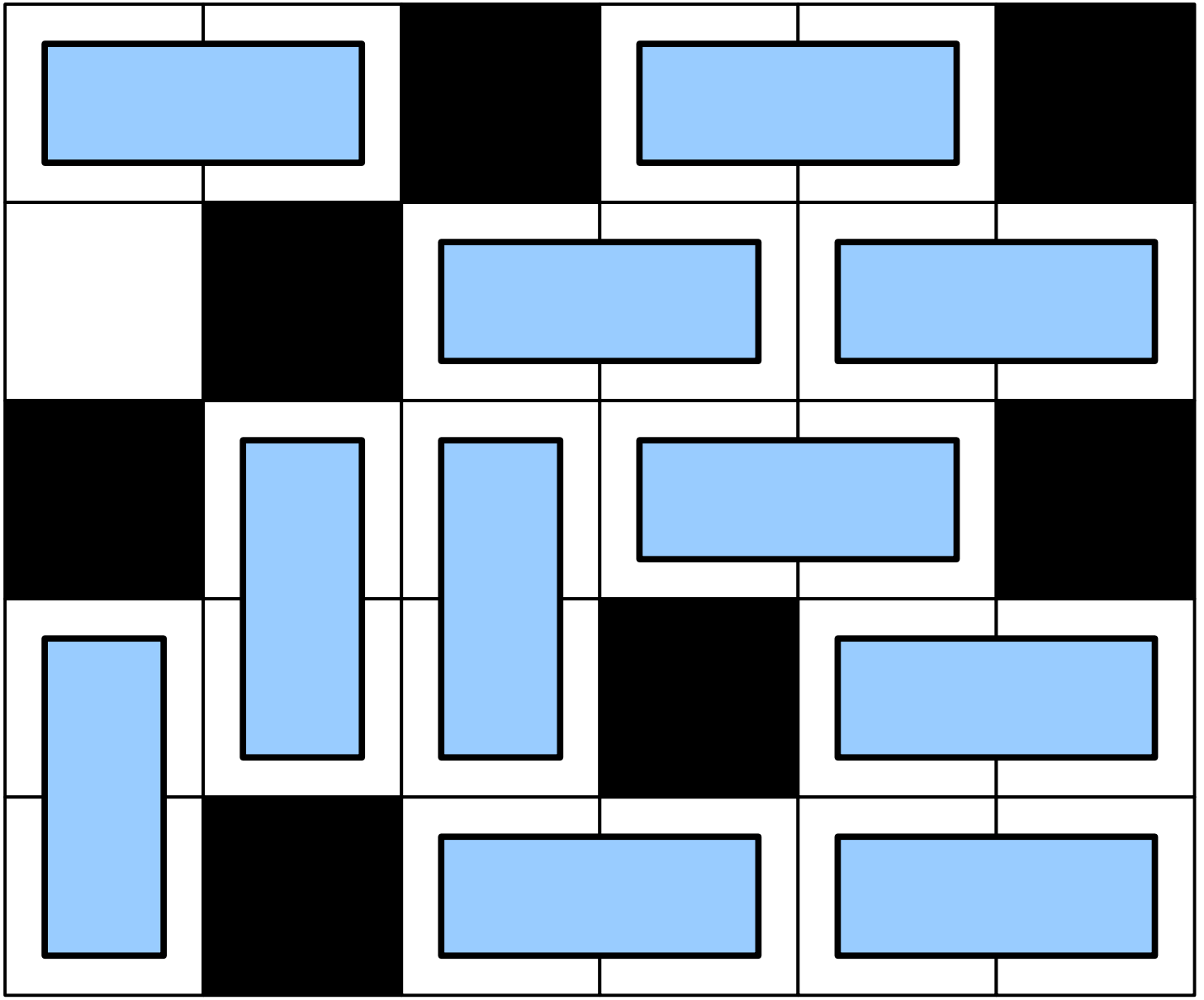
Solving Domino Tiling



Solving Domino Tiling



Solving Domino Tiling



The Setup

- To determine whether you can place at least k dominoes on a crossword grid, do the following:
  - Convert the grid into a graph: each empty cell is a node, and any two adjacent empty cells have an edge between them.
  - Ask whether that graph has a matching of size k or greater.
  - Return whatever answer you get.

In Pseudocode

```
boolean canPlaceDominos(Grid G, int k) {
   return hasMatching(gridToGraph(G), k);
}
```

Why This Works

- This overall construction gives a polynomial-time algorithm for the domino tiling problem.
- It takes on polynomial time to convert the grid into a graph (we'll hand-wave these details away.)
- Once we have that new graph, it takes only polynomial time to check if there's a sufficiently large matching.
- Overall, this only requires polynomial time.