N P

Announcements

- Problem Set 7 graded; will be returned at end of lecture.
- Unclaimed problem sets and midterms moved!
  - Now in cabinets in the Gates open area near the drop-off box.

The Complexity Class **P**

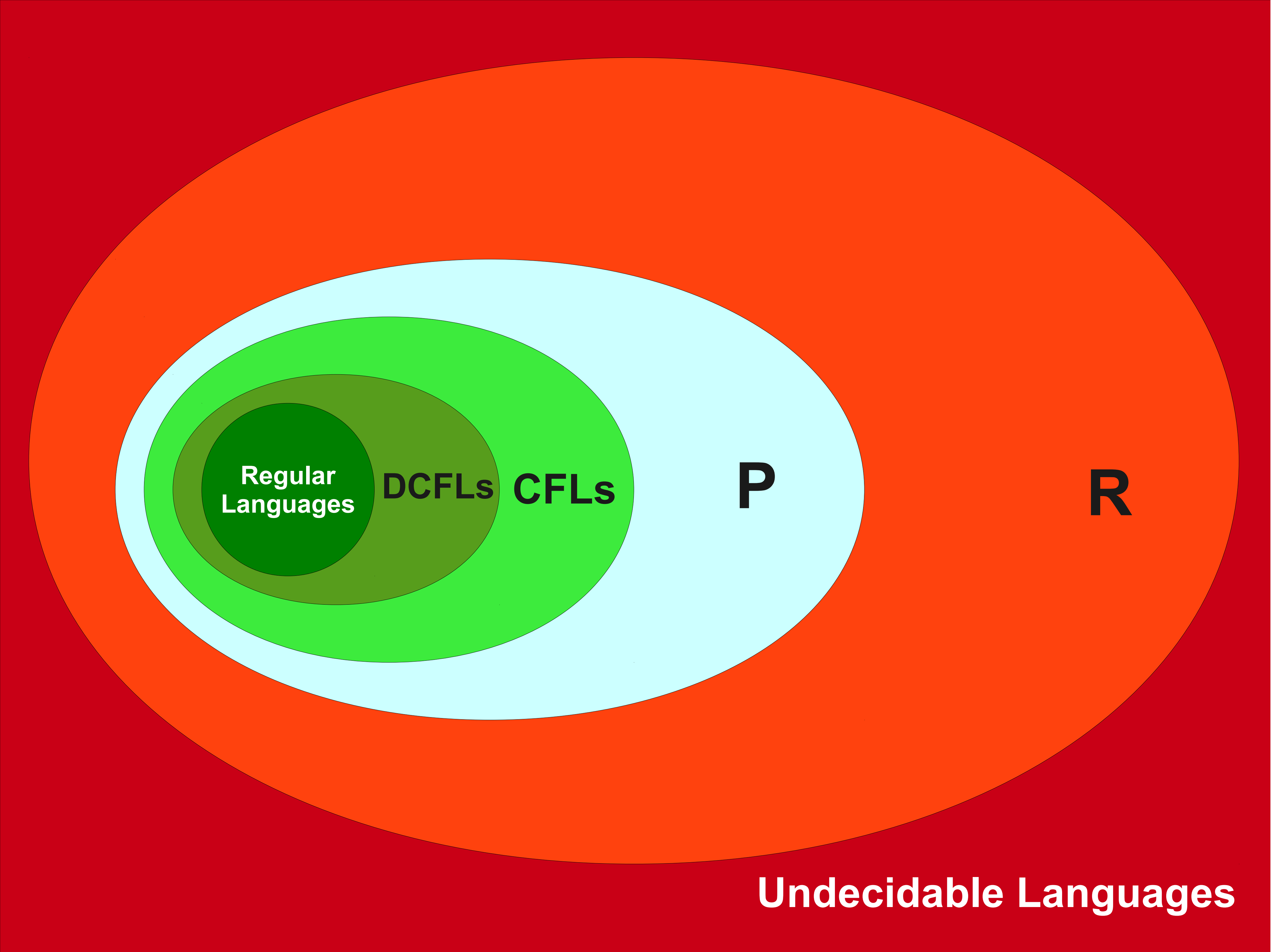
- The complexity class **P** (**p**olynomial time) contains all problems that can be solved in polynomial time.
- Formally:

$$\mathbf{P} = \bigcup_{k=0}^{\infty} \mathrm{TIME}(n^k)$$

- The **Cobham-Edmonds Thesis**: A decision problem can be solved efficiently iff it is in **P**.

Examples of Problems in **P**

- All regular languages are in **P**.
  - Belong to TIME(n).
- All DCFLs are in **P**.
  - Belong to TIME(n^2).
- All CFLs are in **P**.
  - Belong to TIME(n^{18}).
- Many other problems are in **P**:
  - *POWER2*
  - *SEARCH*


Regular Languages
DCFLs
CFLs
P
R
Undecidable Languages

Proving Languages are in P

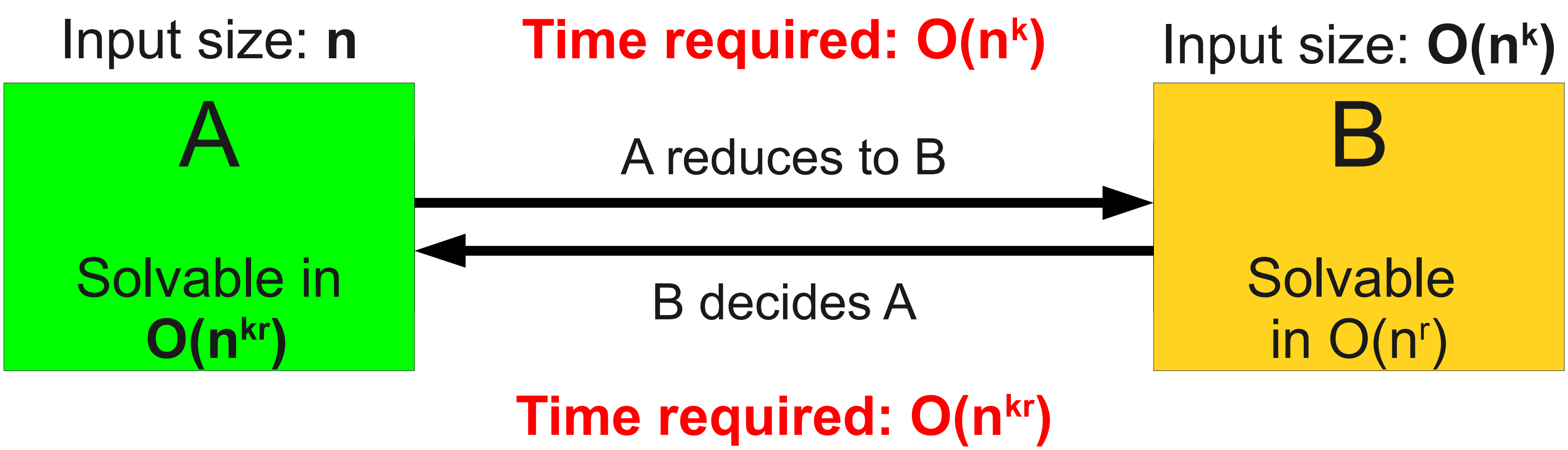
- **Directly prove the language is in P**.
  - Build a decider for the language L.
  - Prove that the decider runs in time $O(n^k)$.
- **Use closure properties**.
  - Prove that the language can be formed by appropriate transformations of languages in **P**.
- **Reduce the language to a language in P**.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

Polynomial-Time Reductions

- Let $A \subseteq \Sigma_1^*$ and $B \subseteq \Sigma_2^*$ be languages.
- A **polynomial-time 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 $f(w) \in B$.
- Notation: $\mathbf{A \leq_P B}$.
- Informally:
  - A way of turning inputs to *A* into inputs to *B*
  - that can be computed in polynomial time
  - that preserves the correct answer.

Polynomial-Time Reductions

- Suppose that we know that $B \in \mathbf{P}$.
- Suppose that $A \leq_P B$.
- Then $A \in \mathbf{P}$ as well.



Theorem: If $B \in \mathbf{P}$ and $A \leq_{\mathrm{P}} B$, 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 $f(w) \in$ A. Since f is a polynomial-time reduction, $f(w) \in B$ iff $w \in A$. Thus M accepts w iff $w \in A$, so $\mathscr{L}(M) = A$. ■

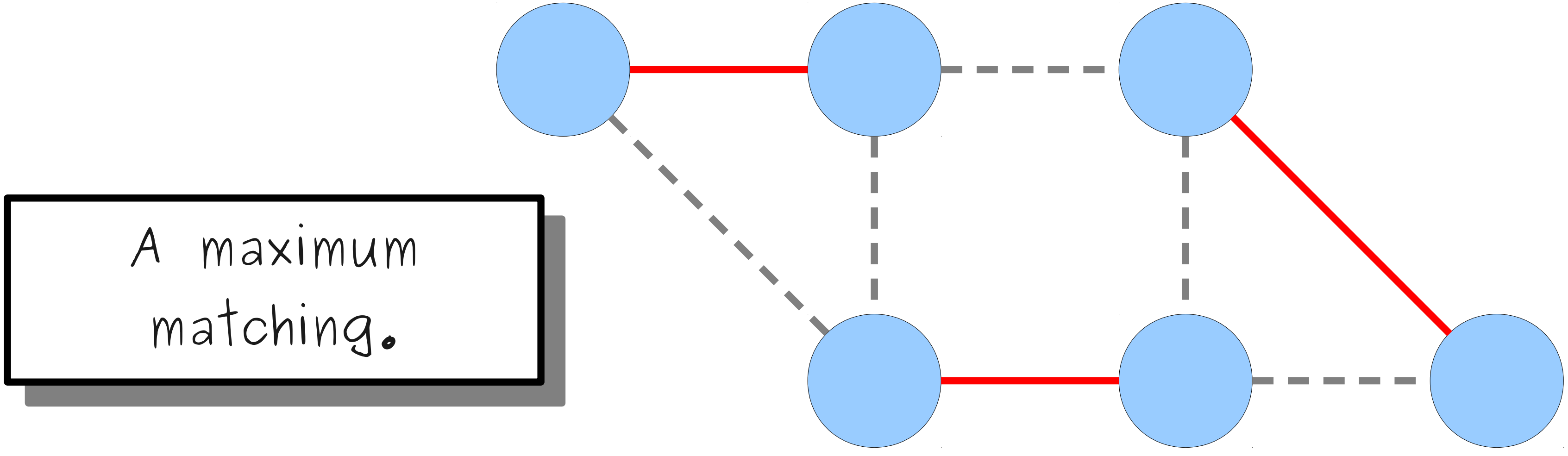
A Sample Reduction

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

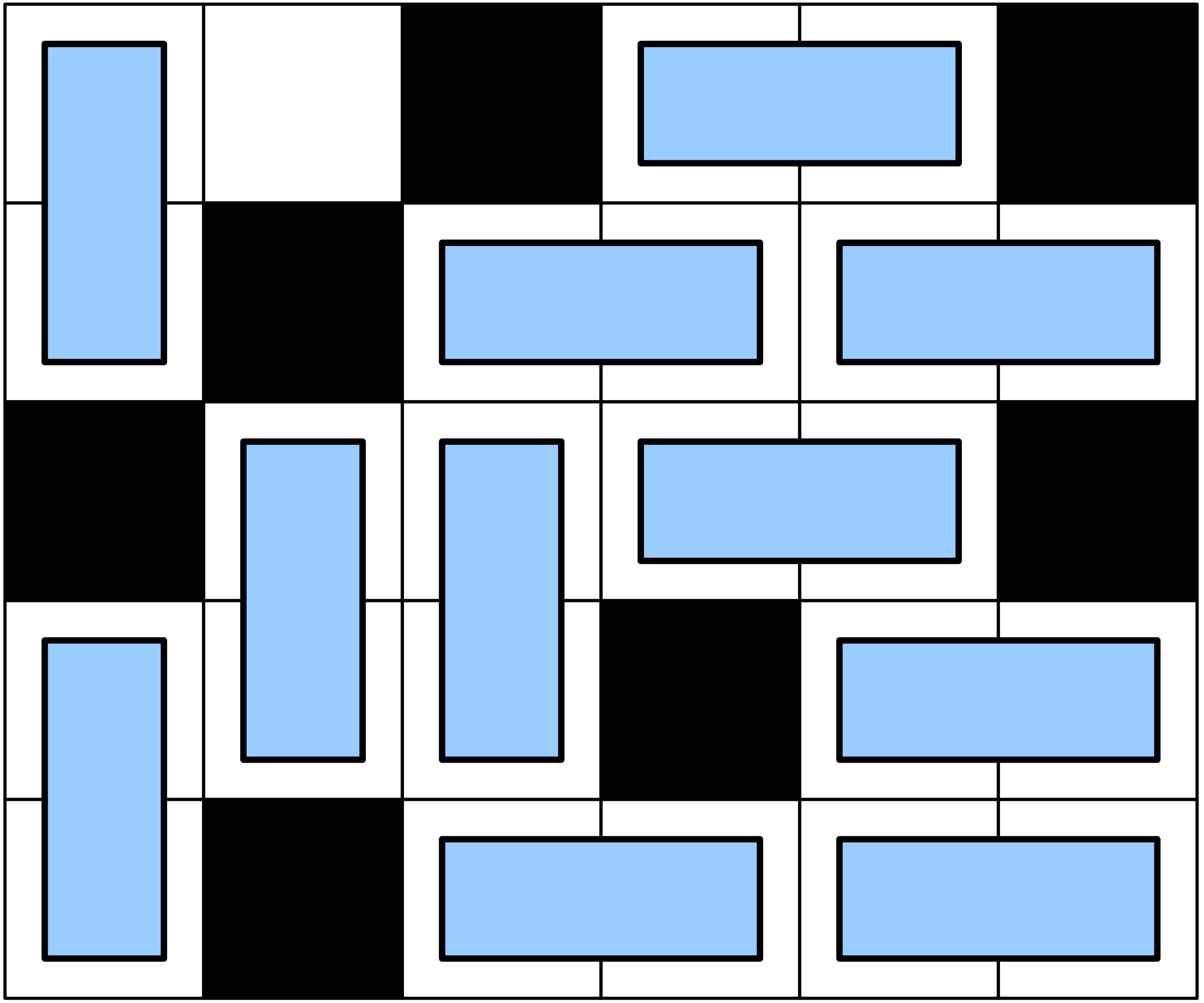
- Given an undirected graph G, a **matching** in G is a set of edges such that no two edges share an endpoint.
- A **maximum matching** is a matching with the largest number of edges.



Maximum Matching

- Jack Edmonds' paper "Paths, Trees, and Flowers" that describes 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



A Domino Tiling Reduction

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

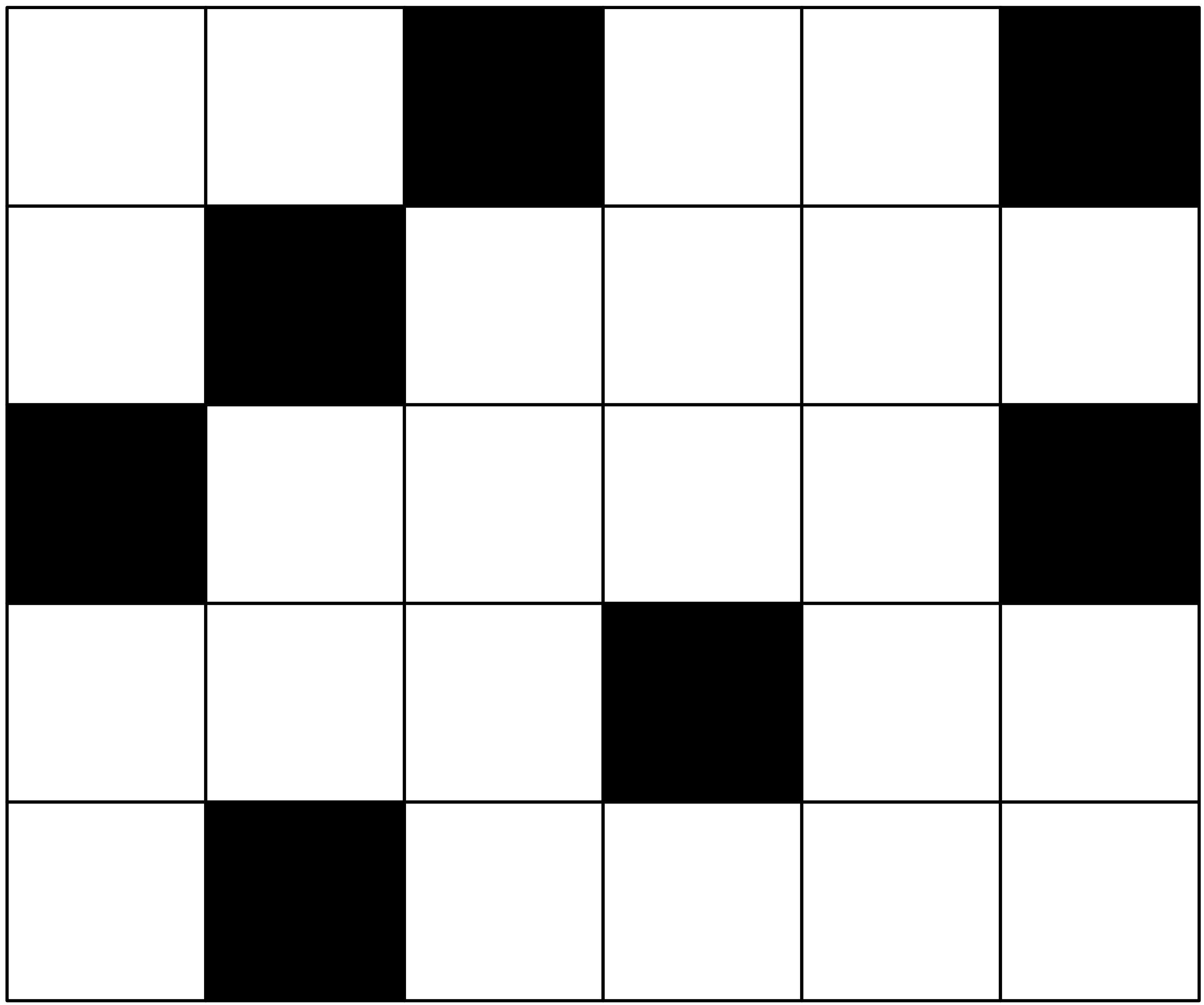
  $MATCHING$ = { ⟨G, k⟩ | G is an undirected graph with a matching of size at least k }

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

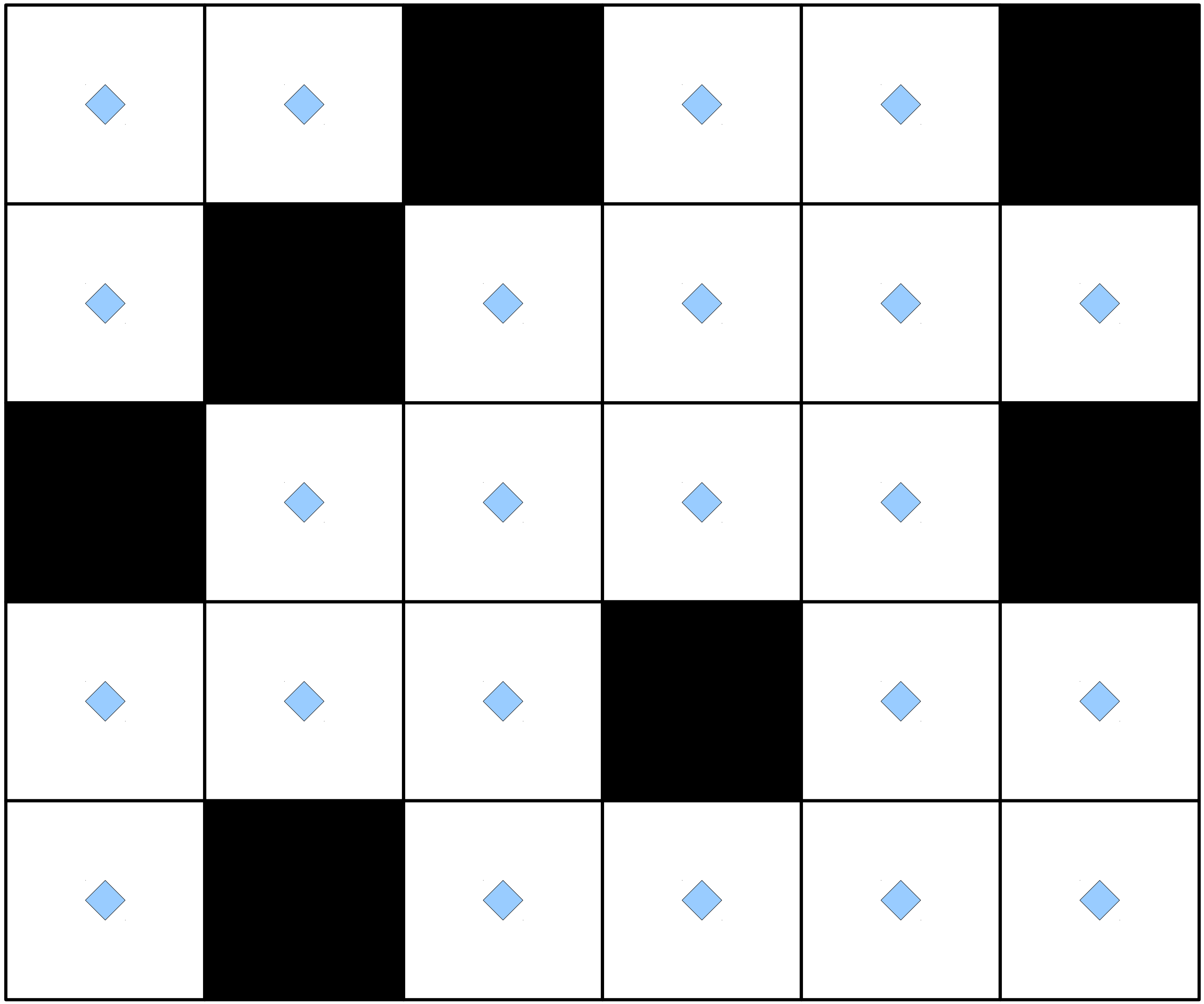
  DOMINO = { ⟨D, k⟩ | D is a grid and k nonoverlapping dominoes can be placed on D. }

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

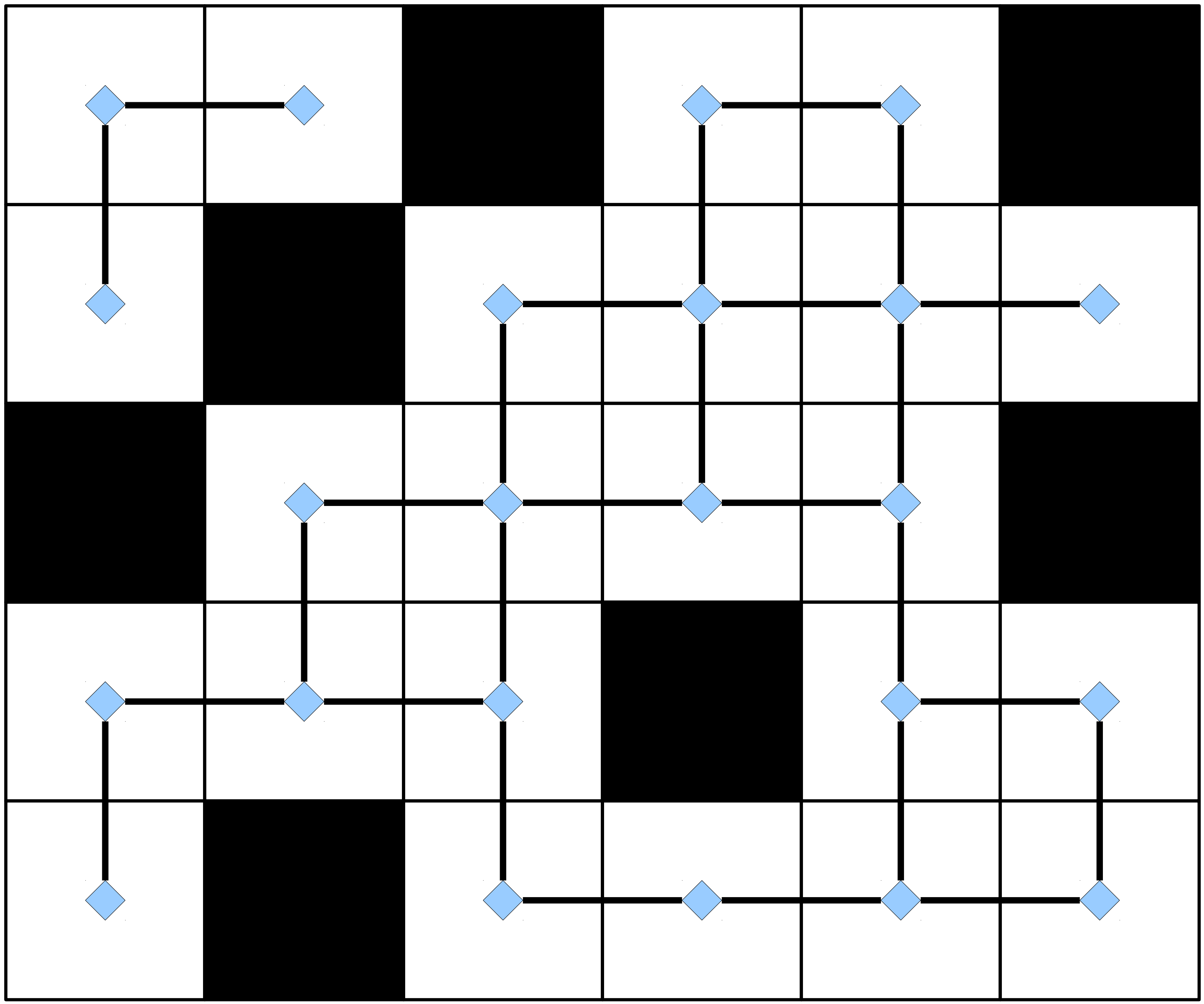
Solving Domino Tiling



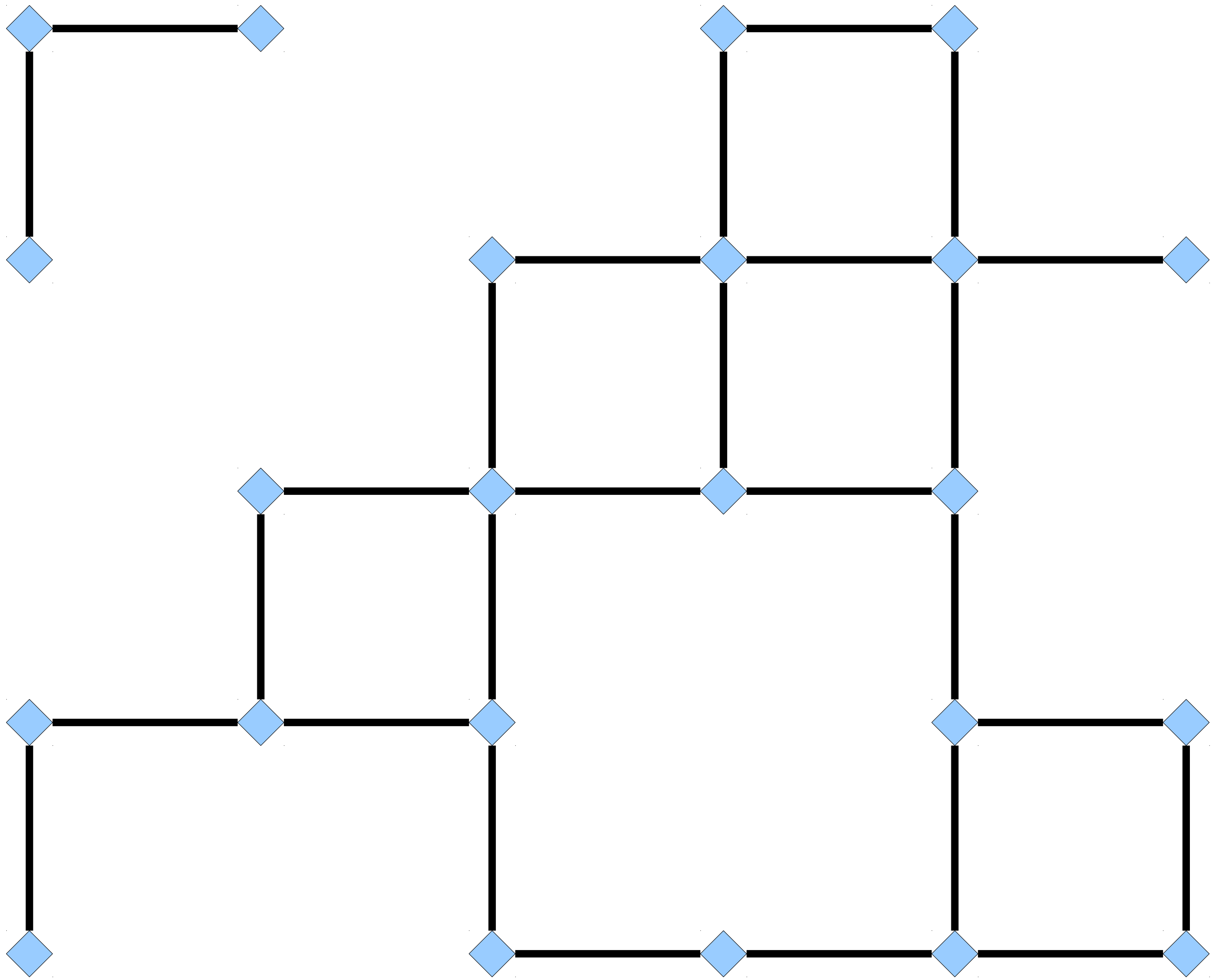
Solving Domino Tiling



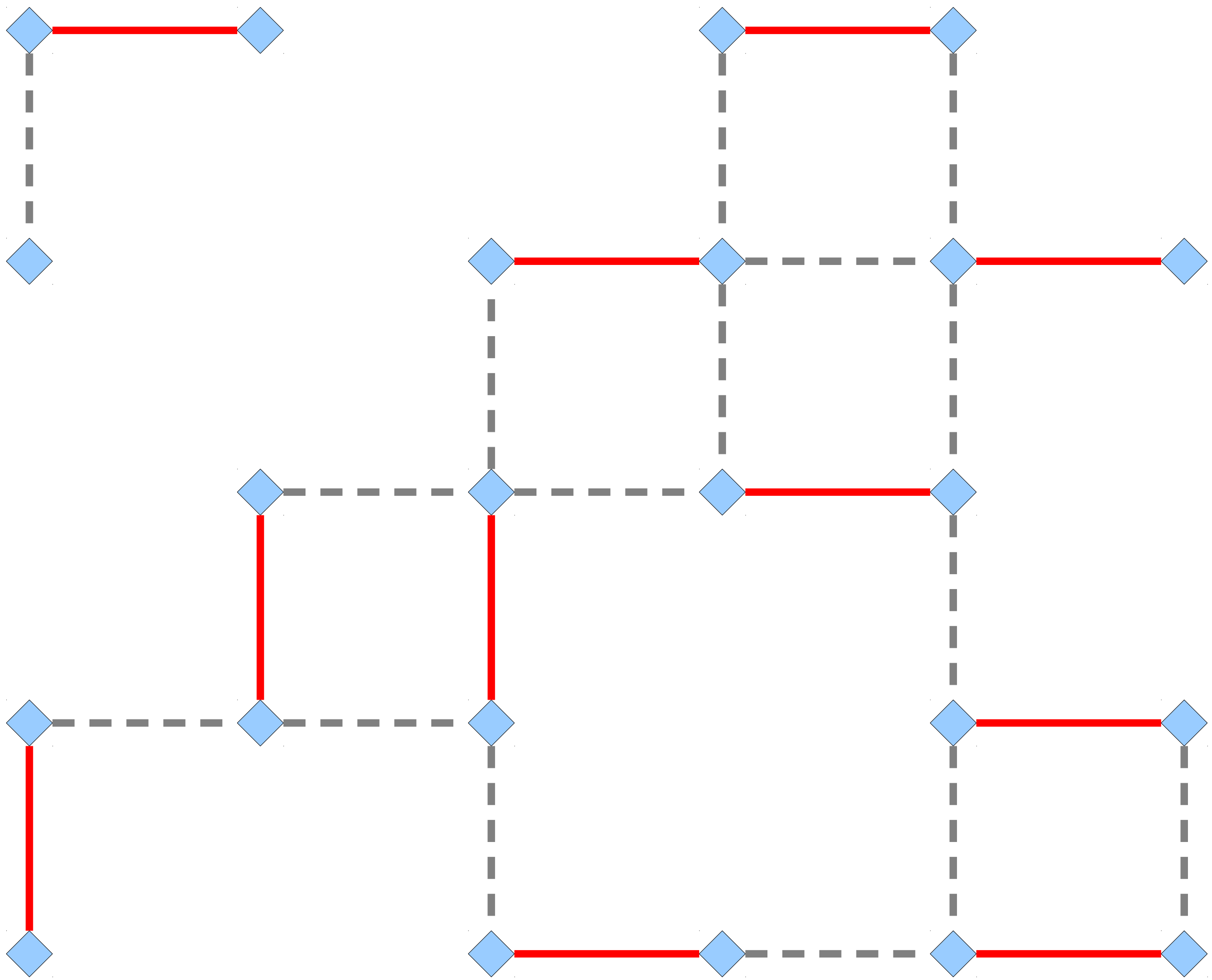
Solving Domino Tiling



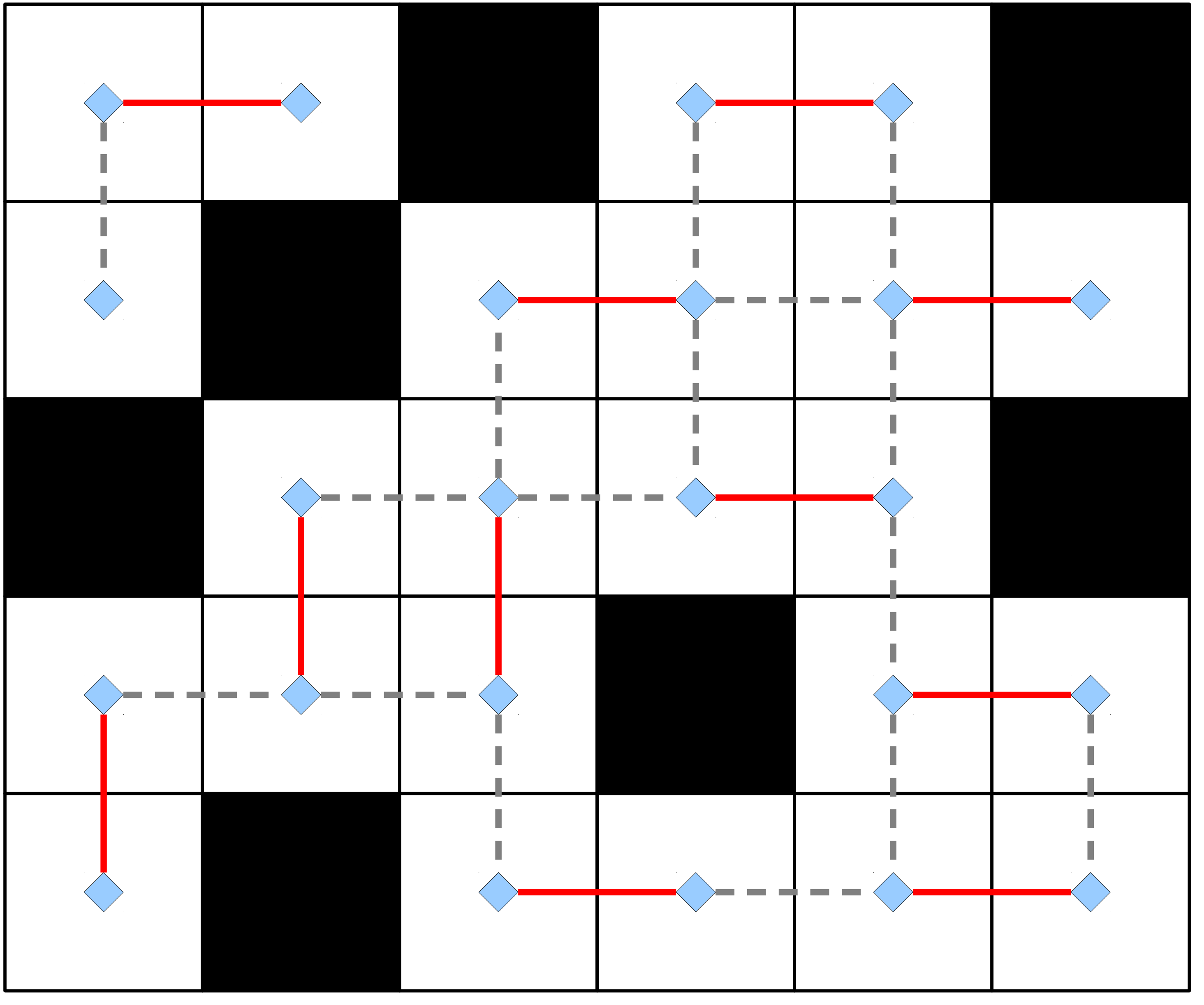
Solving Domino Tiling



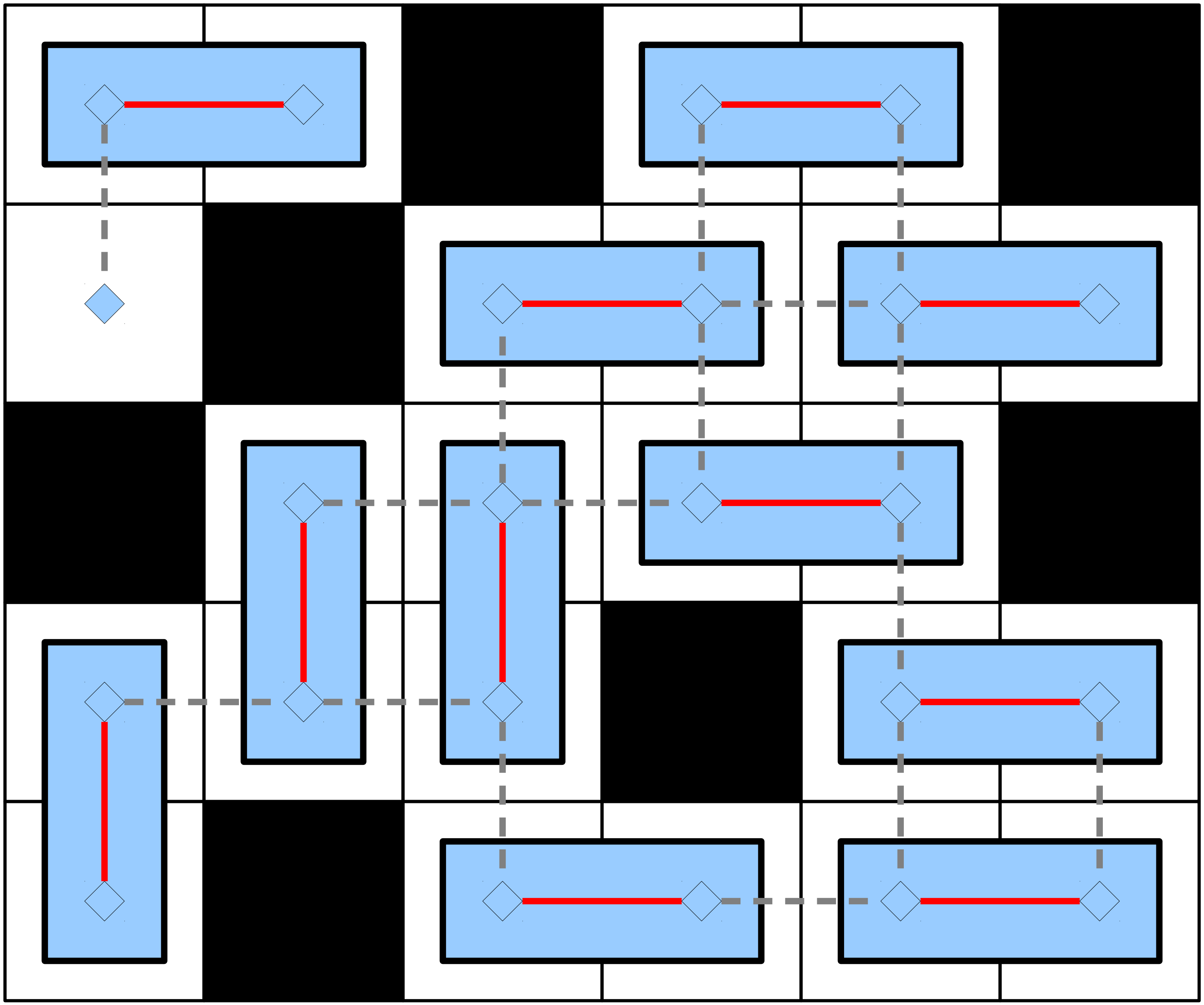
Solving Domino Tiling



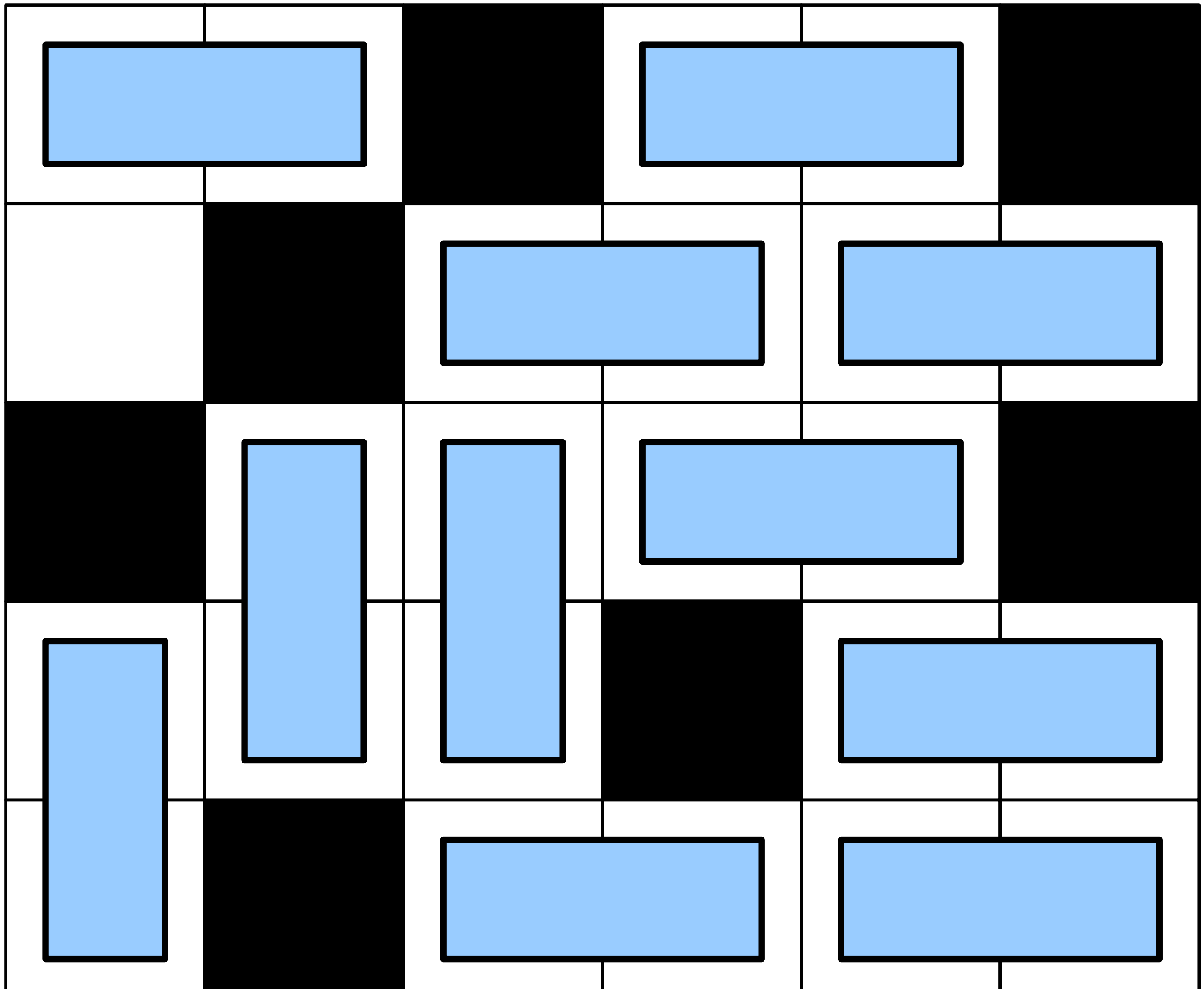
Solving Domino Tiling



Solving Domino Tiling

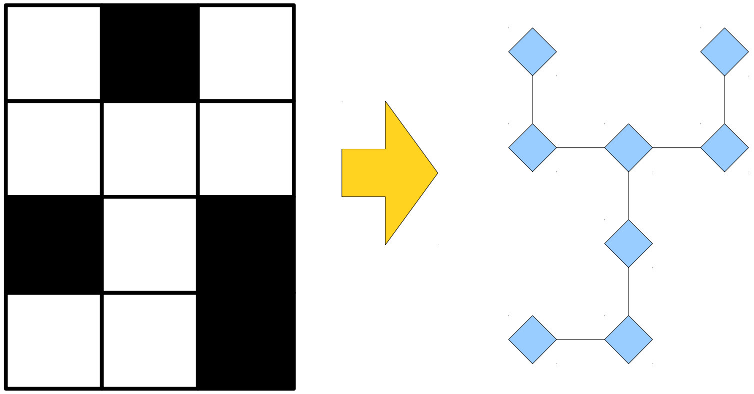


Solving Domino Tiling



Our Reduction

- Given as input $\langle D, k \rangle$, construct the graph G as follows:
  - For each empty cell x_i, construct a node v_i.
  - For each pair of adjacent empty cells x_i and x_j, construct an edge (v_i, v_j)



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

A Polynomial-Time Reduction

- To prove that f is a polynomial-time reduction, we will show that the size of $f(w)$ is a polynomial in the size of w.
  - Technically, this is **not** sufficient to prove that f runs in polynomial time.
  - However, most reductions that construct a polynomially-large object take polynomial time.
  - We will gloss over the fact that the polynomial-size object can be constructed in polynomial time; barring very unusual reductions, this is almost always true.

A Polynomial-Time Reduction

- Given a grid D and a number k, how large is the constructed graph G?
  - One node per empty cell in D.
  - One edge per pair of adjacent empty cells in D.
- There are $O(|D|)$ empty cells in D.
- Each empty cell may have up to four neighbors.
- So there are at most $O(|D|)$ constructed edges.
- Each node and edge can be built in polynomial time, so the overall reduction takes polynomial time.

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 $\langle D, k\rangle$.

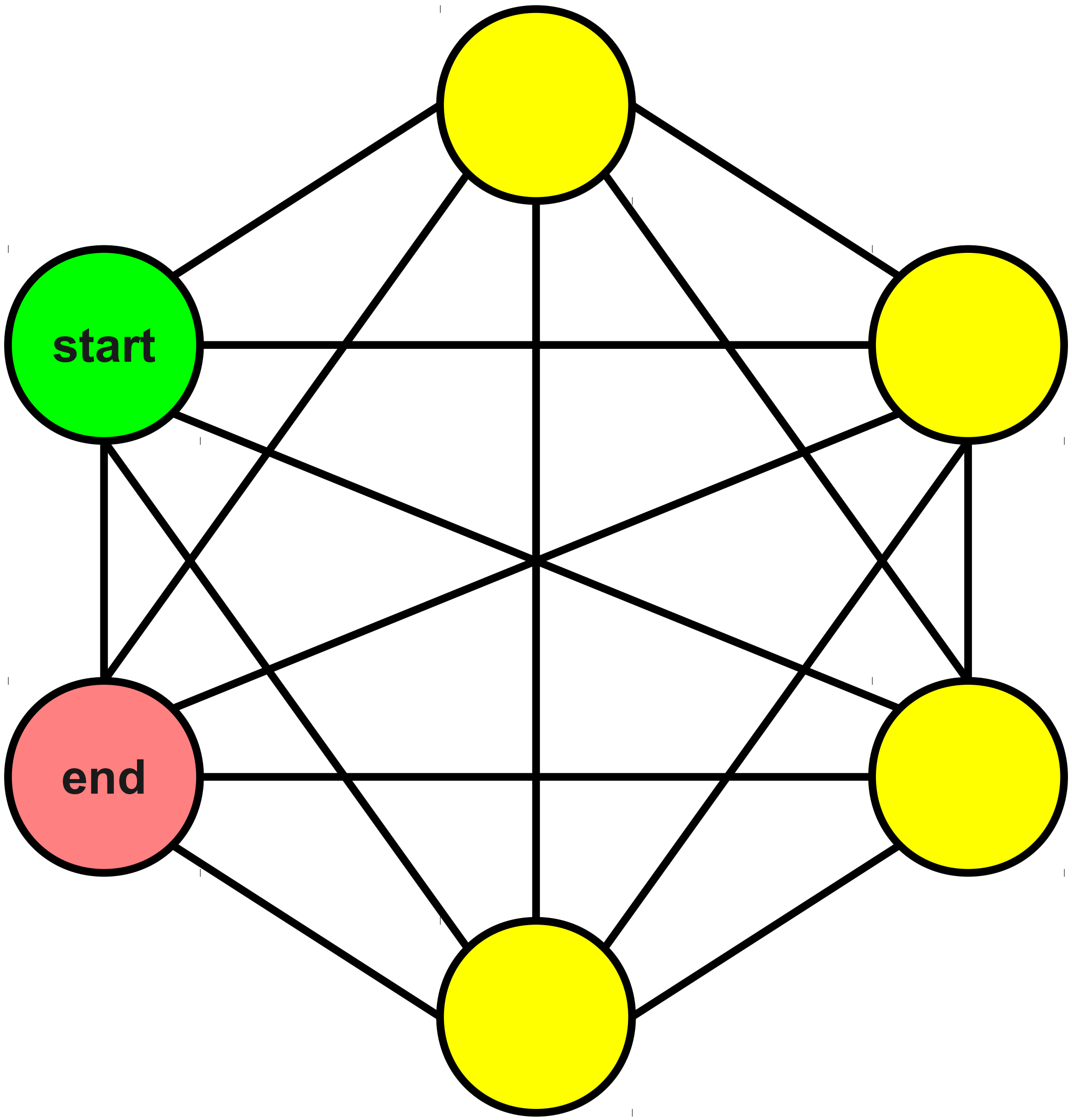
For each empty cell x_i in D, we construct a single node v_i in G. Since there are $O(|D|)$ cells, there are $O(|D|)$ nodes in the graph. For each pair of adjacent, empty cells x_i and x_j in D, we add the edge (x_i, x_j). Since each cell in D has four neighbors, the maximum number of edges we could 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. ■

Summary of **P**

- **P** is the complexity class of yes/no questions that can be solved in polynomial time.
- **P** is closed under polynomial-time reductions.

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



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

$$\{1, 2, 3, 4, 5, 6, 7, 8\}$$

How many subsets of this set are there?

1 2 3 4 5 6 7 8

How many binary search trees can you form from these numbers?

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.
  - Each binary search tree made from some elements has exactly one node per element.
- This brings us to our next topic…

N P

NTMs

- A **nondeterministic Turing machine** (NTM) is a generalization of the Turing machine.
- An NTM may have multiple transitions defined for a given state/symbol combination.
- The NTM accepts if **any** choice of transitions enters an accepting state.
- The NTM rejects if **all** choices of transitions enter a rejecting state.
- Otherwise, the NTM loops.

Nondeterminism Revisited

- If we add nondeterminism to the DFA, we get the NFA.
  - NFAs are no more powerful than DFAs.
- If we add nondeterminism to the DPDA, we get the PDA.
  - PDAs are more powerful than DPDAs.
- Adding nondeterminism to a TM produces the equivalently powerful NTM.
  - NTMs are no more powerful than TMs.

Nondeterminism Revisited

- Converting an NFA to a DFA might introduce exponentially more space.
- It is sometimes impossible to convert an NPDA to a DPDA.
- Converting an NTM to a TM might dramatically slow down the TM.

Designing NTMs

- Nondeterminism is a **very** powerful tool for solving problems.
- Many problems can be solved simply with nondeterminism using the following template:
  - **Nondeterministically** guess some important piece of information.
  - **Deterministically** check that the guess was correct.

Nondeterministic Algorithms

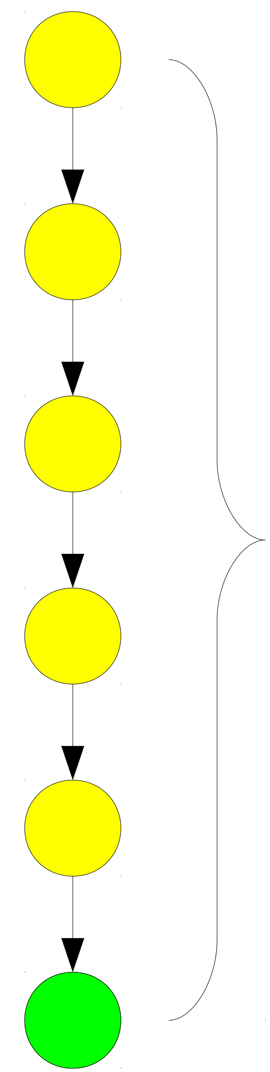
- Recall: a number $n > 1$ is composite if it is not prime.
- Let $\Sigma = \{ \mathtt{1} \}$ and consider the language

$$COMPOSITE = \{ \mathtt{1}^n \mid n \text{ is composite } \}$$

- We can build a **multitape, nondeterministic TM** for *COMPOSITE* as follows:
- M = "On input $\mathtt{1}^n$:
  - **Nondeterministically** write out q $\mathtt{1}$s on a second tape ($2 \le q < n$)
  - **Nondeterministically** write out r $\mathtt{1}$s on a third tape ($2 \le r < n$)
  - **Deterministically** check if $qr = n$.
  - If so, accept.
  - Otherwise, reject"

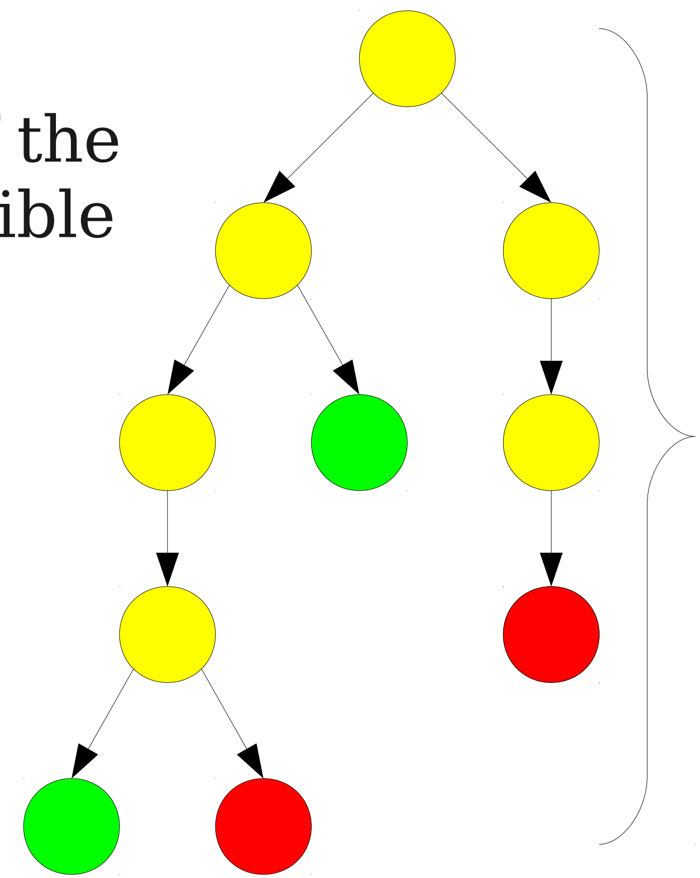
Analyzing NTMs

- 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).



Analyzing NTMs

- M = "On input 1^n:
  - **Nondeterministically** write out q 1s on a second tape ($2 \leq q < n$)
  - **Nondeterministically** write out r 1s on a third tape ($2 \leq r < n$)
  - **Deterministically** check if $qr = n$.
  - If so, accept.
  - Otherwise, reject"

O(n) steps

O(n) steps

O(n^2) steps

\+ O(1) steps

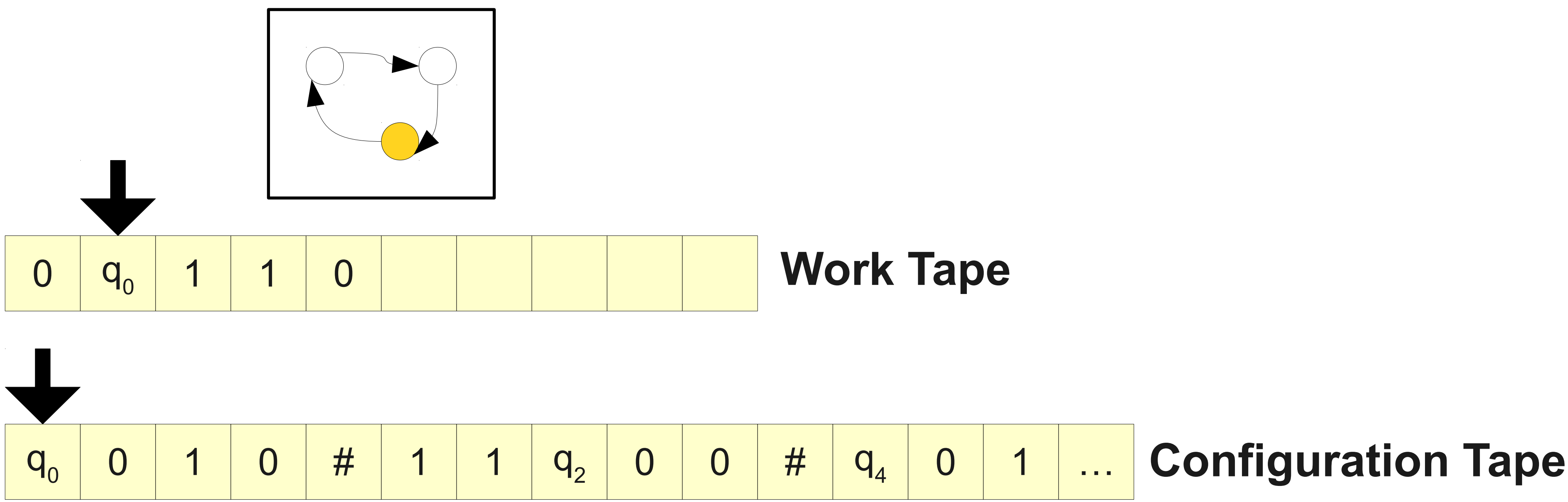
O(n^2) steps

Analyzing NTMs

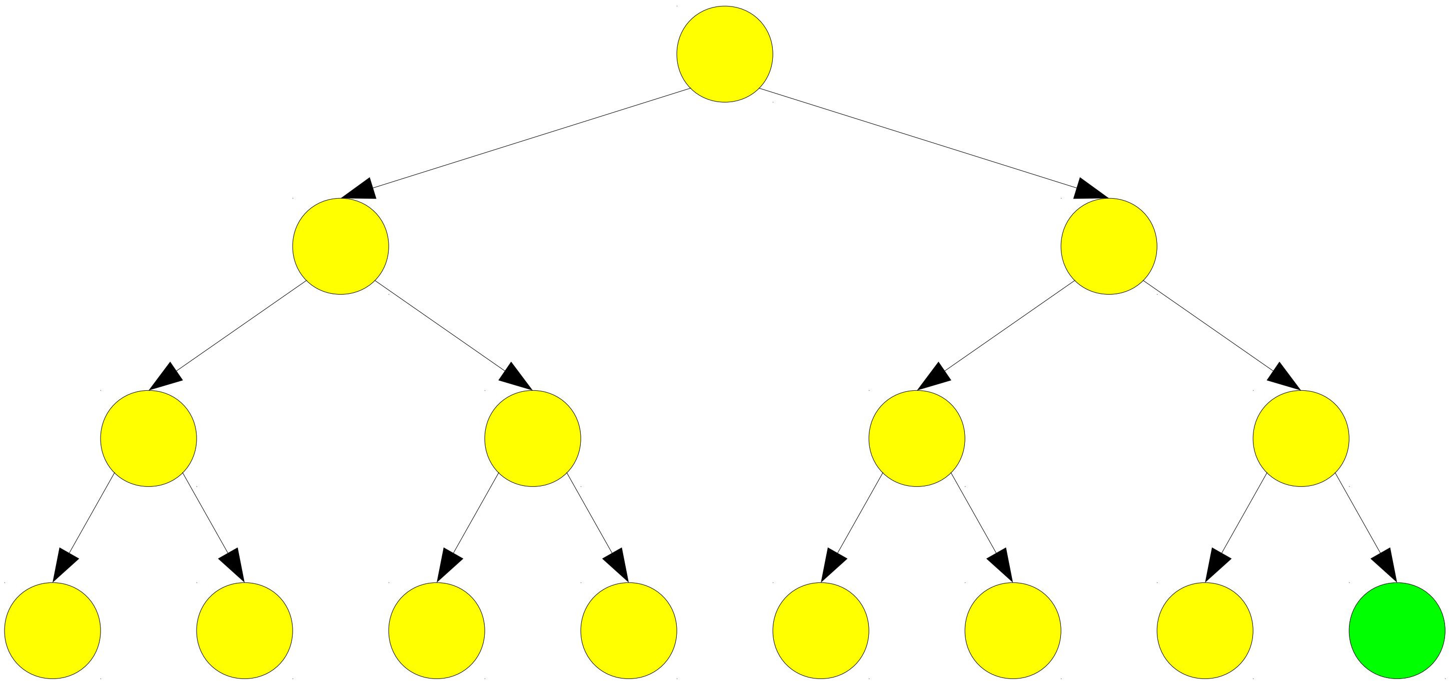
- Our multitape NTM can decide *COMPOSITE* in time $O(n^2)$.
- Using a similar construction to the deterministic case, a single-tape NTM can decide *COMPOSITE* in $O(n^4)$.
- The best known deterministic algorithm for deciding *COMPOSITE* runs *much* more slowly.
  - Runs in time around $O(n^8)$.
- Just how much more powerful are NTMs?

From NTMs to TMs

- NTMs are at least as powerful as TMs.
  - Just don't use any nondeterminism!
- TMs are at least as powerful as NTMs.
  - Idea: Simulate the NTM with a multitape TM.
  - Run a breadth-first search on possible options.



From NTMs to TMs



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.

TIME and NTIME

- **Recall:** TIME($f(n)$) is the class of languages that can be decided in O($f(n)$) time by a single-tape TM.
- **NTIME**($f(n)$) is the class of languages that can be decided in O($f(n)$) time by a single-tape NTM.
  - All possible options terminate in O($f(n)$) steps.
- For any $f(n)$, TIME($f(n)$) $\subseteq$ NTIME($f(n)$).
  - Can always convert a TM to an NTM.

The Complexity Class **NP**

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

$$\mathbf{NP} = \bigcup_{k=0}^{\infty} \mathrm{NTIME}(n^k)$$

- 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."

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

If we allow for a generalized Sudoku board of arbitrary size:

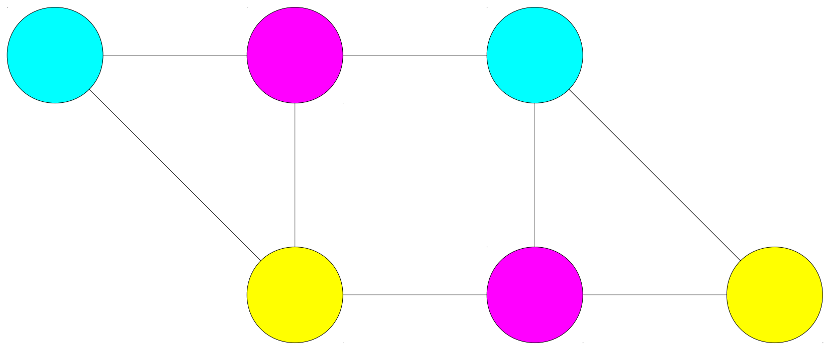
There are polynomially many grid cells to fill in.

Checking the grid takes polynomial time.

Overall algorithm takes polynomial time.

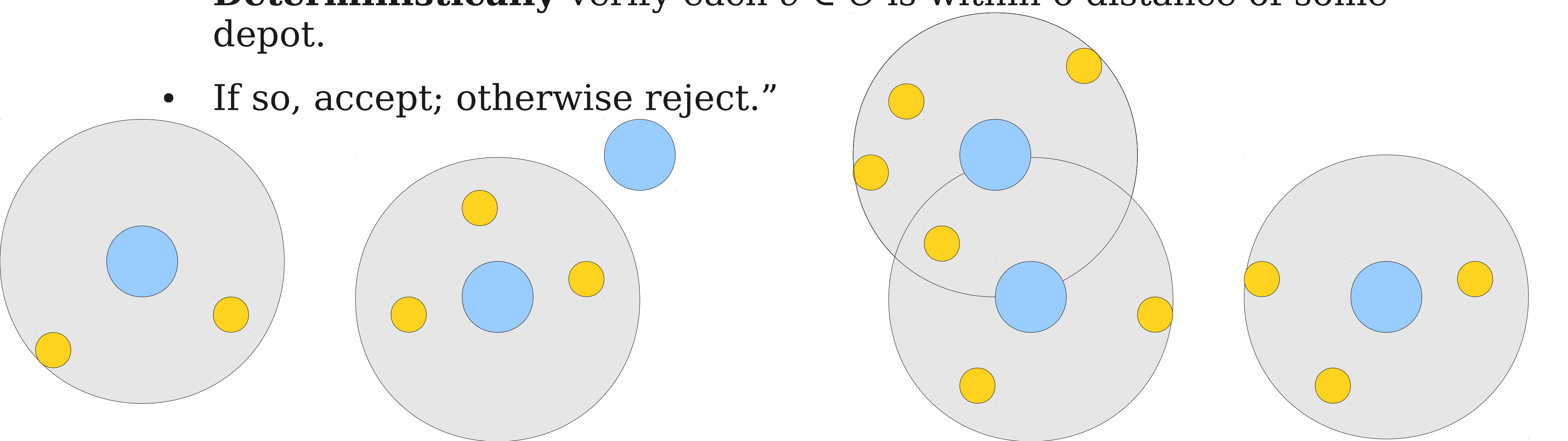
A Problem in **NP**

- 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."



A Problem in **NP**

- Suppose you want to start a delivery service. You want to place depots such that each customer is within some distance of the depot.
- Given a set of candidate locations for depots, can you place k depots and guarantee that each customer is covered?
- M = "On input $\langle D, C, \delta, k\rangle$ (depot locations, customer locations, minimum distance required, and number of depots desired):
    - **Nondeterministically** guess k depots from D.
    - **Deterministically** verify each $c \in C$ is within δ distance of some depot.
    - If so, accept; otherwise reject."



A General Pattern

- The NTMs we have seen so far always follow this pattern:
  - M = "On input w:
    - **Nondeterministically** guess some string x.
    - **Deterministically** check whether x solves w.
    - If so, accept; otherwise, reject."
- Is there a different way of characterizing **NP**?

Polynomial-Time Verifiers

- A **polynomial-time verifier** is a deterministic TM of the form
  - M = "On input $\langle w, x \rangle$:
    - **Deterministically** check whether x solves w.
    - If so, accept; otherwise, reject."

  such that M runs in time polynomial in the length of w (not the length of x).
- The string x is called a **certificate** or a **witness** for w.

An Efficiently Verifiable Puzzle

Question: Can this lock be opened?

Verifiers, Formally

- Formally, a **verifier** is a TM V such that

$$\mathbf{w \in \boldsymbol{L} \quad \text{iff} \quad \exists \boldsymbol{x} \in \Sigma^*. \ \boldsymbol{V} \text{ accepts } \langle \boldsymbol{w}, \boldsymbol{x} \rangle}$$

- In other words

$$\boldsymbol{L} = \{ \ \boldsymbol{w} \in \Sigma^* \mid \exists \boldsymbol{x} \in \Sigma^*. \ \mathbf{V} \text{ accepts } \langle \boldsymbol{w}, \boldsymbol{x} \rangle \ \}$$

- If $w \in L$, the verifier can check this easily if we know the proper x.
- If $w \notin L$, the verifier does not help much.
  - Just because V rejects $\langle w, x \rangle$ does not mean that $w \notin L$.
- **Note that $\mathscr{L}(V) \neq L$.**

Verification is Powerful

- Many undecidable languages can still be verified.
- Here is a verifier for *HALT*:
  - V = "On input $\langle M, w, n \rangle$, where M is a TM, w is a string, and n is a natural number:
    - Run M on w for n steps.
    - If M halts w within that time, accept; otherwise reject."
- V always halts on all inputs (even if M loops on w).
- If M halts on w, there is some choice of n for which V accepts (namely, the number of steps M takes before it halts on w).
- Thus *HALT* can be **verified** but not **decided**.

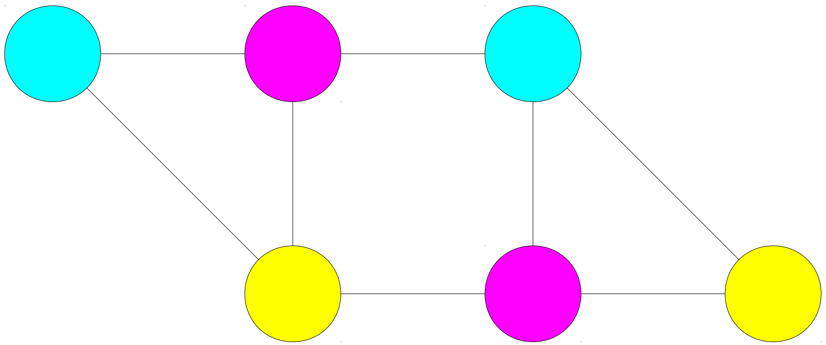
A Problem in **NP**

- 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 graph G be colored with at most k colors?
- M = "On input $\langle G, k, C \rangle$, where C is an alleged coloring:
  - **Deterministically** check whether C is a legal k-coloring of G.
  - If so, accept; if not, reject."



Two Equivalent Formulations of **NP**

- **Theorem**: A language L has a polynomial-time verifier iff $L \in \mathbf{NP}$.
- **Proof sketch**:
  - Any polynomial-time verifier can be turned into a polynomial-time NTM by having the NTM nondeterministically guess the certificate for w, then check it (deterministically) by running the verifier.
  - If an NTM can decide L in polynomial time, a verifier could work by having a certificate saying which nondeterministic choices the original NTM made, then simulating those choices of the NTM to check it.

Next Time

- **NP-Completeness**
  - What are the hardest problems in **NP**?
  - How do you prove a problem is **NP**-complete?