SOTA

- Resolving $P \stackrel{?}{=} NP$ has proven extremely difficult

- In the past 50 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 $P \stackrel{?}{=} NP$

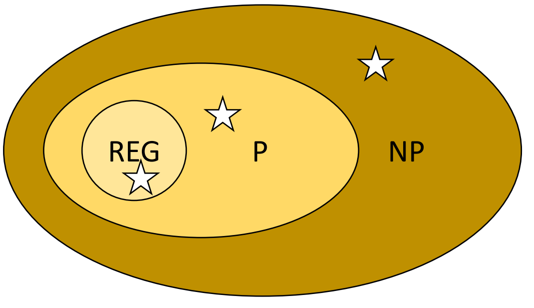
- Most(?) computer scientists believe $P \neq NP$

Transforming

Sub-Classes in NP

- Problems in NP vary widely in their difficulty, even if P = NP
- How can we rank the relative difficulties of problems?

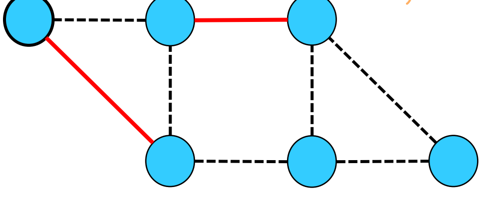
2 problems



Maximum Matching Problem

- 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.

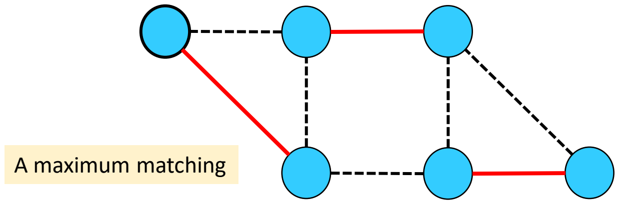
2 edges
→ X share endpoint

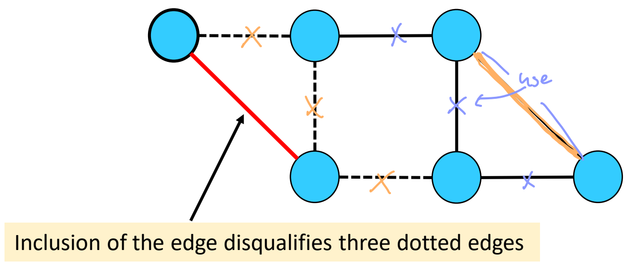


A matching, but not a maximum matching

Is max. matching P or NP?

Maximum Matching Problem



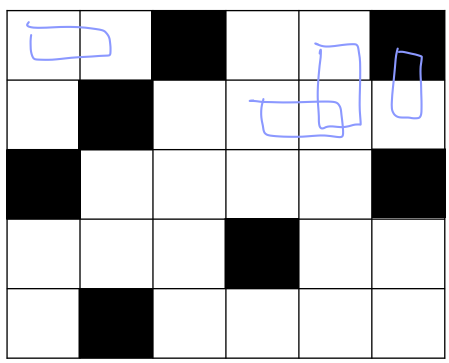


maximum matching

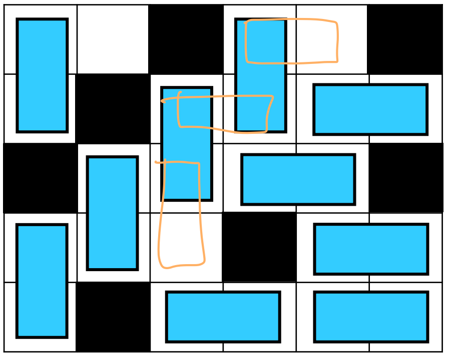
~~maximal~~

Domino Tiling Problem

- Given an $n \times m$ board with blocks, tile as many 2×1 dominos as possible



Domino Tiling

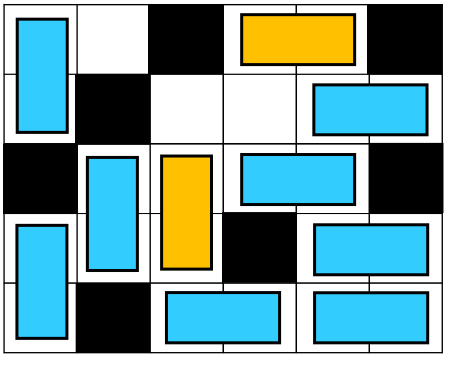


one more domino possible

Is this optimal?

NO

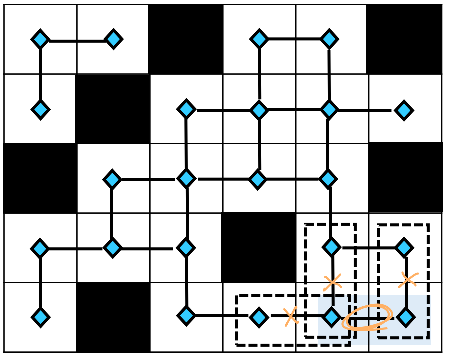
Domino Tiling



Optimal

Relation between maximum matching and domino tiling?

Domino Tiling



Transformation
think each block = node
L connected by link

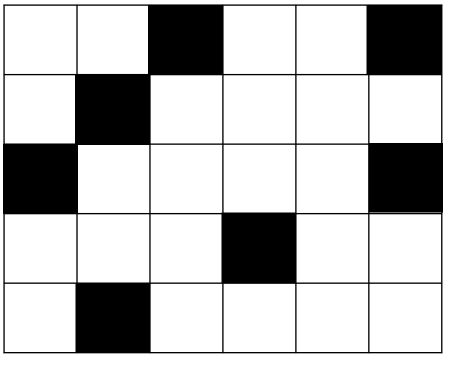
Represent an empty square as a node
Connect adjacent empty squares with an edge

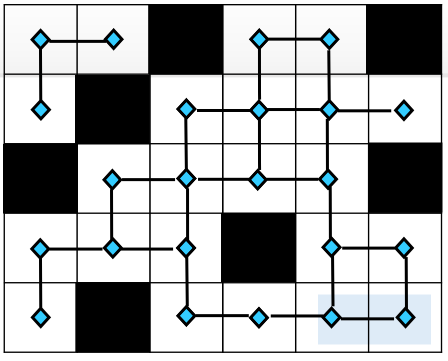
Suppose you place a horizontal domino on right bottom corner
Then you can not place dominos as shown

➔ Maximum matching problem

Domino Tiling

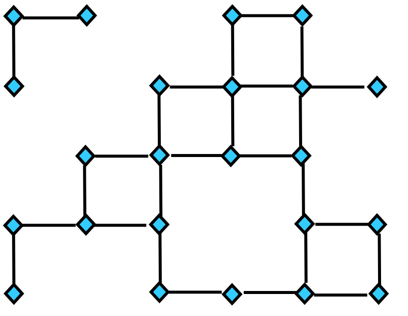
KENTECH
Korea Institute of Energy Technology
AI





(Domino Tiling problem)

(Maximum Matching problem)



Relative Difficulty


```
bool canPlaceDominos (Grid G, int k) {
    return hasMatching (GridToGraph(G), k);
}
```

연계는 가능

Domino / Matching problem

?.

Among Domino tiling & MM problems, which one is easier (or more difficult)? → 뭐가 더 어려울지 = MM.

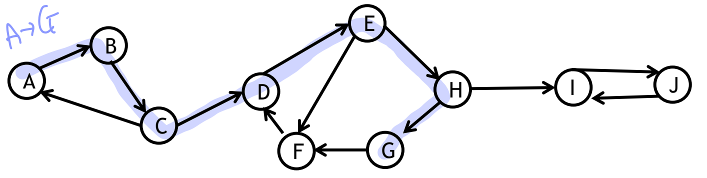
Domino tiling can't be "harder" than maximum matching, because if we can solve maximum matching efficiently, we can solve domino tiling efficiently

Another Example: Reachability

- Reachability problem:

Given a directed graph G and nodes s and t in G,

is there a path from s to t?



- It's known that this problem can be solved in polynomial time (use DFS or BFS)
- Given that we can solve the reachability problem in polynomial time, what other problems can we solve in polynomial time?

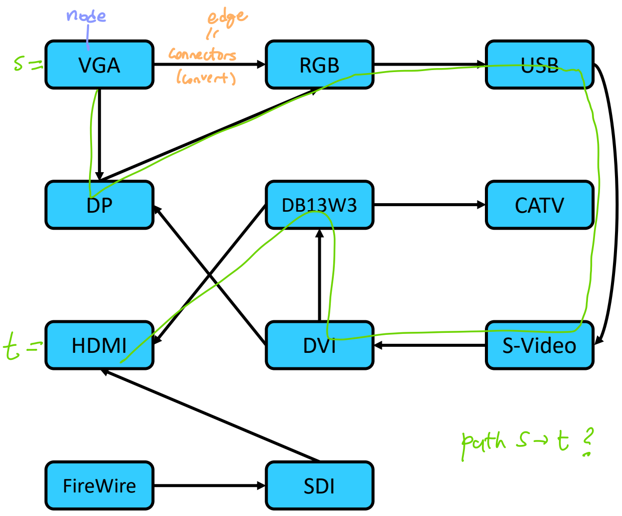
Converter Conundrums

- Suppose that you want to plug your laptop into a projector
- Your laptop only has a VGA output, but the projector needs HDMI input
- You have a box of connectors that convert various types of input into various types of output (for example, VGA to DVI, DVI to DisplayPort, etc.)
- Question: Can you plug your laptop into the projector?
  - Use as many converters as necessary
  - Minimum number of converters

Converter Conundrums

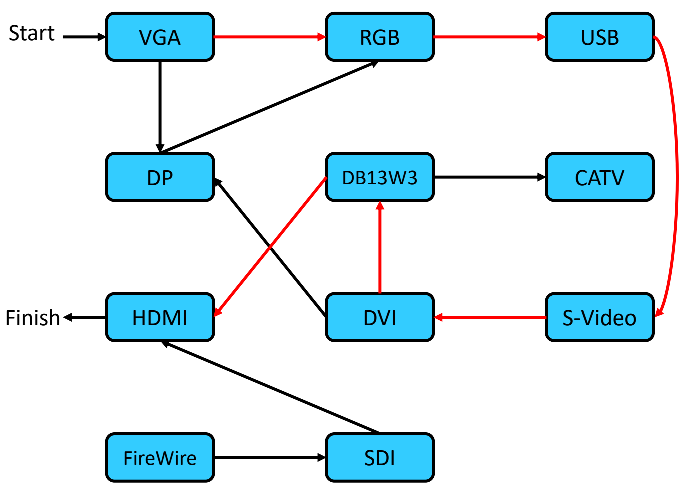
KENTECH Korea Institute of Energy Technology AI

Converters
- RGB to USB node
- VGA to DisplayPort
- DB13W3 to CATV
- DisplayPort to RGB
- DB13W3 to HDMI
- DVI to DB13W3
- S-Video to DVI
- FireWire to SDI
- VGA to RGB
- DVI to DisplayPort
- USB to S-Video
- SDI to HDM



Converter Conundrums

KENTECH Korea Institute of Energy Technology AI



Relative Difficulty

AI

```
bool canPlugIn (List<Plug> plugs) {
                    가능
    return isReachable (plugsToGraph (plugs), VGA, HDMI;
}
```

difficult

Among PlugIn & Reachability problems, which one is easier (or more difficult)?

PlugIn can't be "harder" than the reachability problem, because if we can solve reachability efficiently, we can solve PlugIn efficiently

Reduction

Polynomial-Time Reduction

```
bool problemA (input) {
    return problemB(transform (input));
}
```

A→B

difficult

Problem A can't be "harder" than problem B, because solving problem B lets us solve problem A

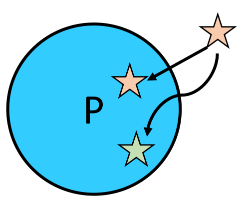
- If A and B are problems where it's possible to solve A by transforming it into B (in polynomial time) and solve B, we write

  $\mathbf{A} \leq_p \mathbf{B}$ → A can't be harder than B.

- ***A is polynomial-time reducible to B***
- The $\leq_p$ relation ranks the relative difficulties of problems

Polynomial-Time Reduction

- If A $\leq_p$ B and B ∈ **P**, then A ∈ **P**



bool probA ( ){
bool probB (+,)
}

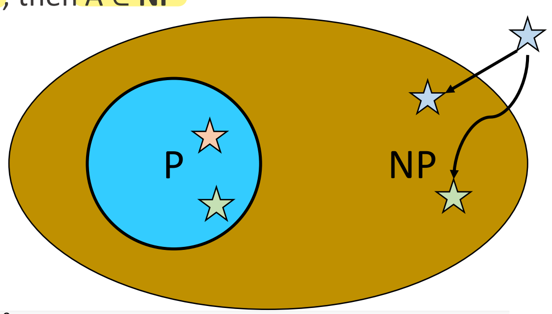
A $\leq_p$ B

new problem

- If A $\leq_p$ B and B ∈ **NP**, then A ∈ **NP**

well known problem

~50 ~100



NP-Hardness, NP-Completeness

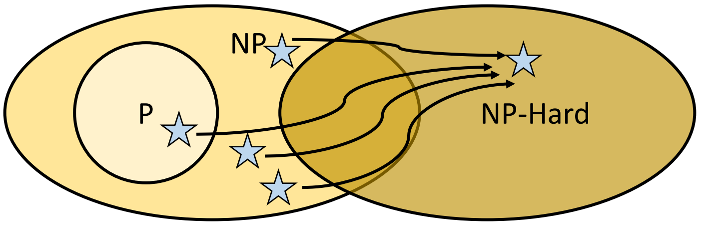
NP-Hardness

KENTECH Korea Institute of Energy Technology AI

- If A $\leq_p$ B, then problem B is at least as hard as problem A
- To show that some problem is hard, show that lots of other problems reduce to it
  - To show that B is hard, find other problems that can be reduced to B

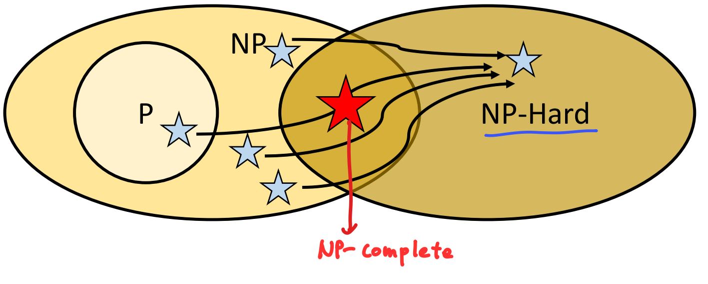
NP-Hardness

- A language L is called ***NP-hard*** if for every A ∈ NP, we have $A \leq_p L$



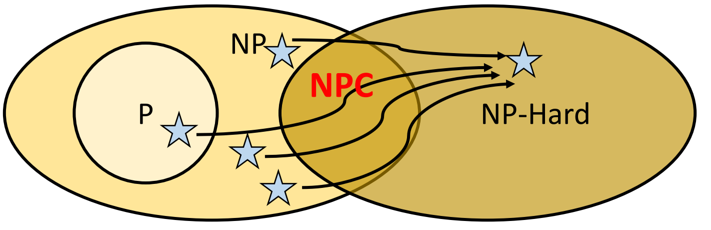
L has to be at least as hard as every problem in NP, since an algorithm for L can be used to decide all problems in NP

NP-Completeness



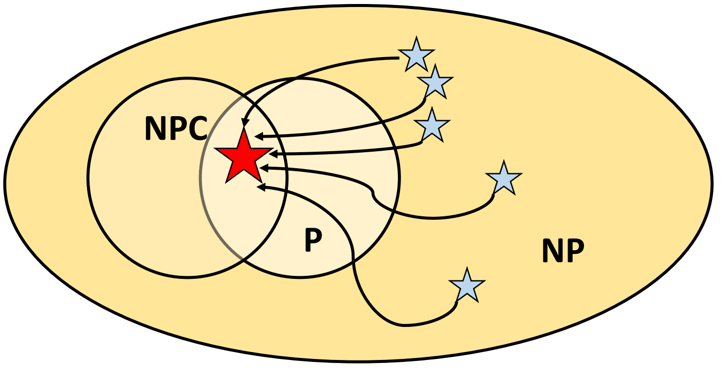
- A language L is called **NP-hard** if for every A ∈ NP, we have $A \leq_p L$
- A language in L is called ***NP-complete*** if L is NP-hard and L ∈ NP
- The class ***NPC*** is the set of NP-complete problems

NPC

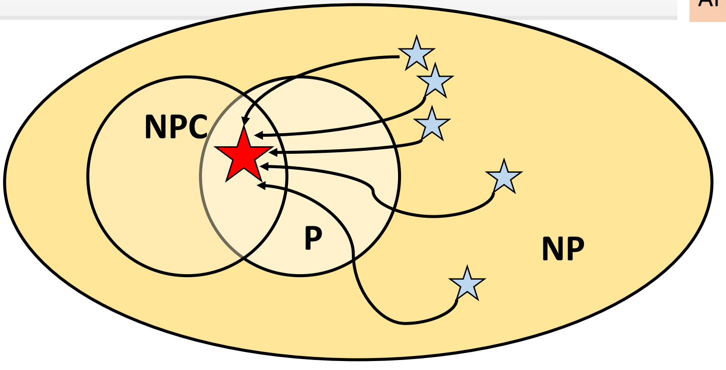


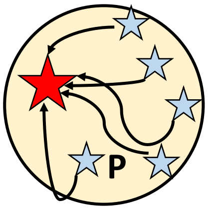
P=NP

- Theorem: If any NP-Complete language is in **P**, then **P=NP**



P=NP



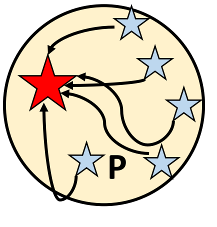


63

P=NP

- **Theorem**: If any NP-complete language is in P, then P = NP
- **Proof**:
  - Suppose that L is NP-complete and $L \in$ P
  - Consider any arbitrary NP problem A
  - Since L is NP-complete, A $\leq_p L$
  - Since $L \in$ P and A $\leq_p L$, A $\in$ P
  - Since A was arbitrary chosen, NP $\subseteq$ P

➔ P = NP



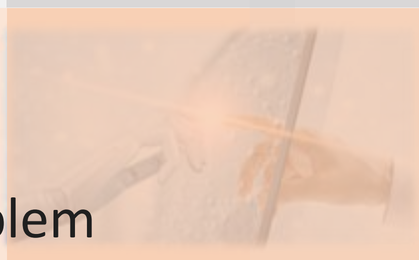
64

P≠NP

AI

- **Theorem**: If any NP-complete language is not in P, then P ≠ NP
- Proof:
  - Suppose that L is an NP-complete language not in P
  - Since L is NP-complete, $L \in$ NP
  - Therefore, $L \in$ NP and $L \notin$ P

  ➔ P ≠ NP



An NP-Complete Problem

Question?

KENTECH Korea Institute of Energy Technology AI

- How do we even know NP-Complete problems exist?

Satisfiability

KENTECH Korea Institute of Energy Technology AI

- A propositional logic formula φ is called satisfiable if there is some assignment to its variables that makes it evaluate to true
  - p ∧ q is satisfiable
  - p ∧ ¬p is unsatisfiable
  - p → (q ∧ ¬q) is satisfiable

- An assignment of true and false to the variables of φ that makes it evaluate to true is called a satisfying assignment

SAT

KENTECH Korea Institute of Energy Technology AI

- The boolean satisfiability problem (SAT) is the following:

  Given a propositional logic formula φ, is φ satisfiable?

- Formally: SAT = { ⟨ φ ⟩ | φ is a satisfiable PL formula }

SAT ∈ NP

KENTECH Korea Institute of Energy Technology AI

- To show that SAT ∈ NP, need to show that verification takes polynomial time
- **Theorem (Cook-Levin):** SAT is NP-complete
- Proof Idea:
  - A certificate (T/F assignment of each variable) is given
  - To show that SAT is NP-hard, given a polynomial-time verifier V for an arbitrary NP language L, for any string w you can construct a polynomially-sized formula φ(w) that says "there is a certificate c where V accepts ⟨w, c⟩."
  - This formula is satisfiable if and only if w ∈ L, so deciding whether the formula is satisfiable decides whether w is in L

SAT

- Resolving P $\stackrel{?}{=}$ NP is equivalent to just figuring out how hard SAT is
- If SAT $\in$ P, then P = NP
- If SAT $\notin$ P, then P ≠ NP

NP-Hard Problems

- ***Computational biology***: Given a set of genomes, what is the most probable evolutionary tree that would give rise to those genomes? (Maximum parsimony problem)
- ***Game theory***: Given an arbitrary perfect-information, finite, twoplayer game, who wins? (Generalized geography problem)
- ***Operations research***: Given a set of jobs and workers who can perform those tasks in parallel, can you complete all the jobs within some time bound? (Job scheduling problem)
- ***Machine learning***: Given a set of data, find the simplest way of modeling the statistical patterns in that data (Bayesian network inference problem)
- ***Medicine***: Given a group of people who need kidneys and a group of kidney donors, find the maximum number of people who can end up with kidneys (Cycle cover problem)
- ***Systems***: Given a set of processes and a number of processors, find the optimal way to assign those tasks so that they complete as soon as possible (Processor scheduling problem)

NP-Intermediate

KENTECH
Korea Institute of Energy Technology
AI

- With few exceptions, every problem we've discovered in NP has either
  - definitely been proven to be in P, or
  - definitely been proven to be NP-complete
- A problem that's NP, not in P, but not NP-complete is called NP-intermediate
- Theorem (Ladner): There are NP-intermediate problems if and only if P ≠ NP.