COL 351 : Analysis & Design of Algorithms

LECTURE 40

NP-COMPLETENESS IV : REDUCTIONS (CONTD.)

NOV 13, 2026 | ROHIT VAISH

THE CLASS NP

A search problem belongs to the complexity class NP if

1. Candidate solutions have description length (in bits) polynomial in the input size.

2. Feasibility of candidate solutions can be checked in time polynomial in the input size.

EXAMPLES OF PROBLEMS IN NP

Search version of TSP

Search version of MST

Search version of sequence alignment

Satisfiability

Directed Hamiltonian Path

Job scheduling

Vertex cover

Independent set

⋮

AMASSING EVIDENCE OF INTRACTABILITY

What we wanted : Class of problems that are solvable by exhaustive search

What we defined : Class of problems NP with efficiently recognizable solutions

What's the connection?

AMASSING EVIDENCE OF INTRACTABILITY

What we wanted : Class of problems that are solvable by exhaustive search

What we defined : Class of problems NP with efficiently recognizable solutions

What's the connection?

Every problem in NP can be solved by using exhaustive search.
(in exponential time)

NP-HARD PROBLEMS

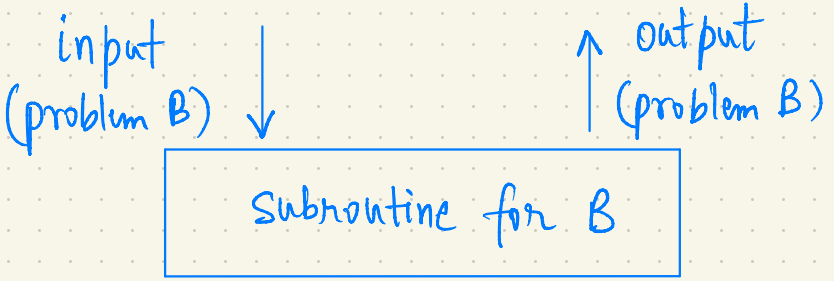
A problem is NP-hard if every problem in NP reduces to it.

REDUCTIONS

A problem A reduces to problem B if any algorithm that solves B is "easily translated" to one that solves A.

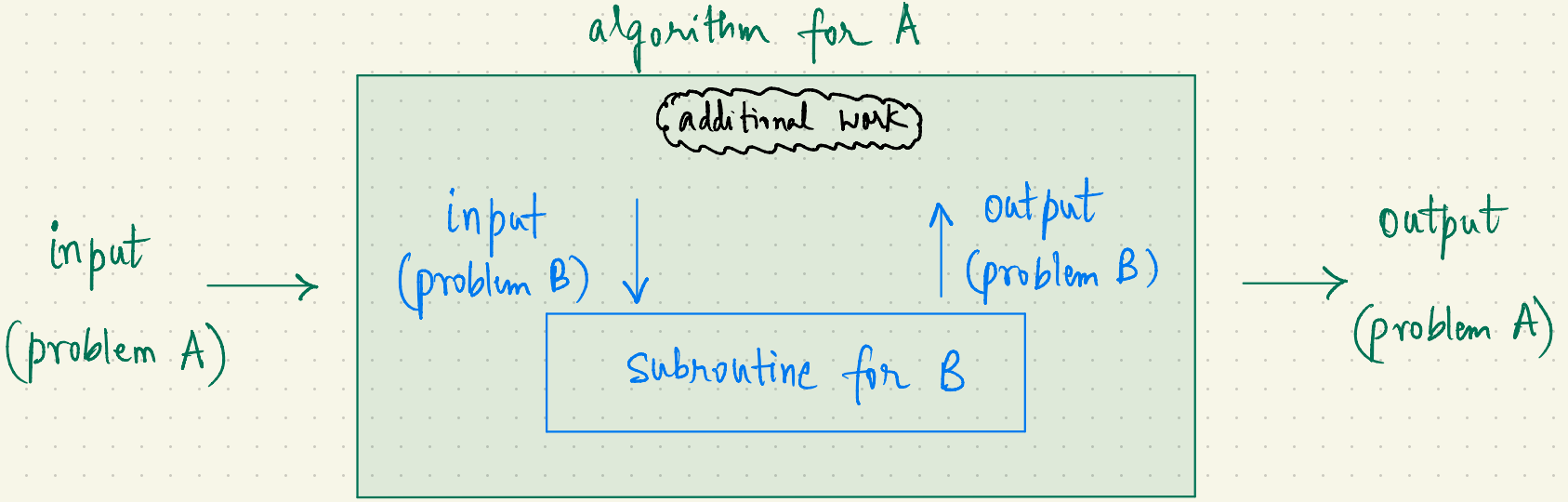
REDUCTIONS

A problem A reduces to problem B if any algorithm that solves B is "easily translated" to one that solves A.



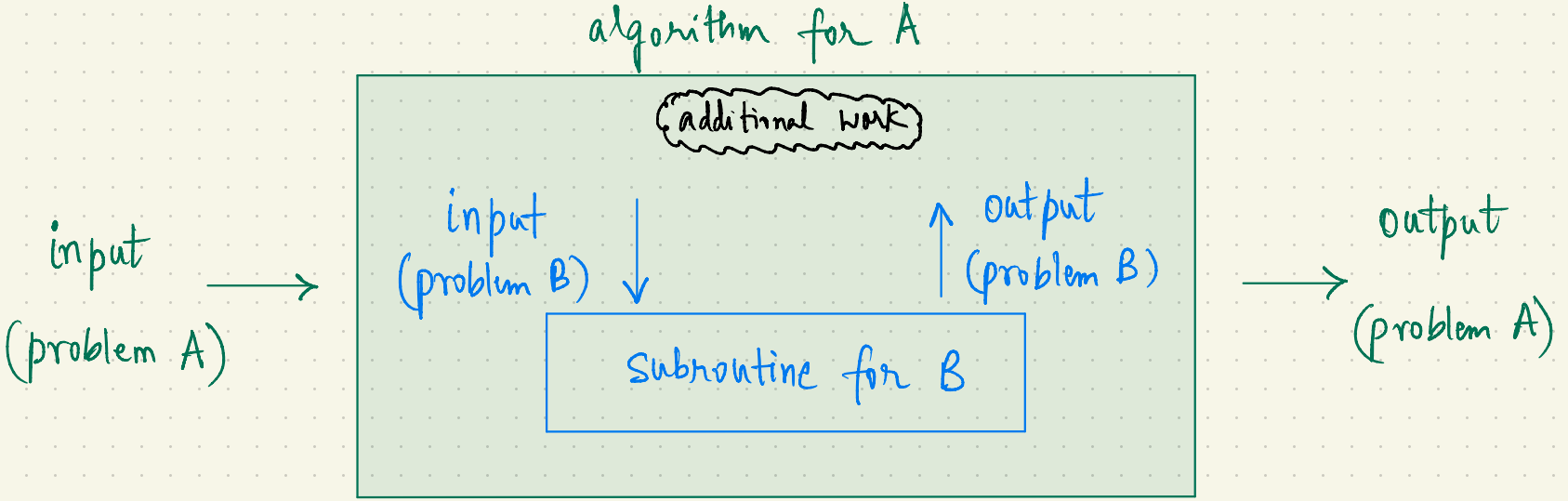
Reductions

A problem A reduces to problem B if any algorithm that solves B is "easily translated" to one that solves A.



Reductions

A problem A reduces to problem B if any algorithm that solves B is "easily translated" to one that solves A.



① # invocations of subroutine for B is at most polynomial in size of input of A.

② Any additional work done by algo. for A is bounded by poly. in size of input of A.

Reductions

1. Spread tractability

2. Spread intractability

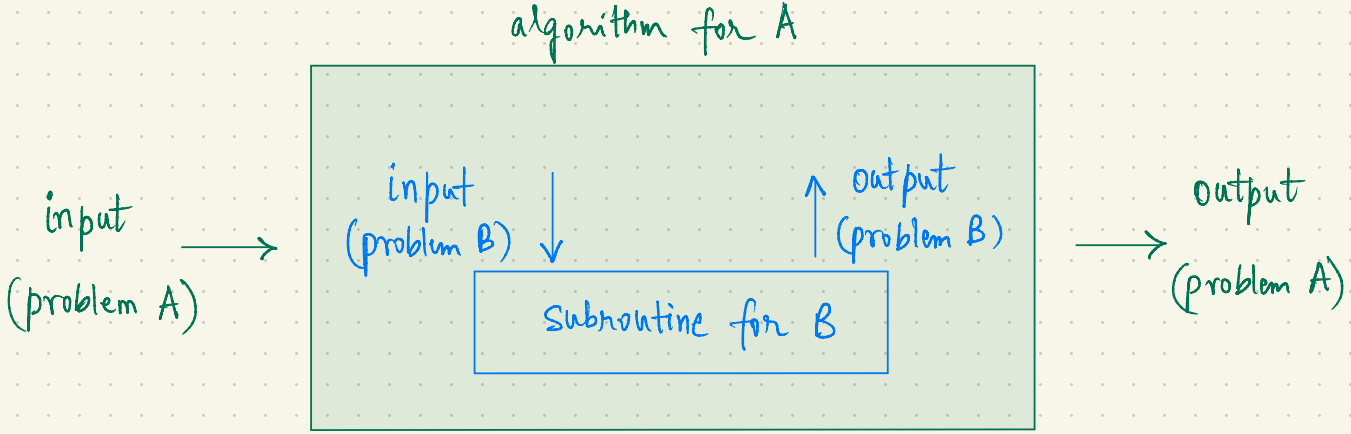
Reductions

1. Spread tractability

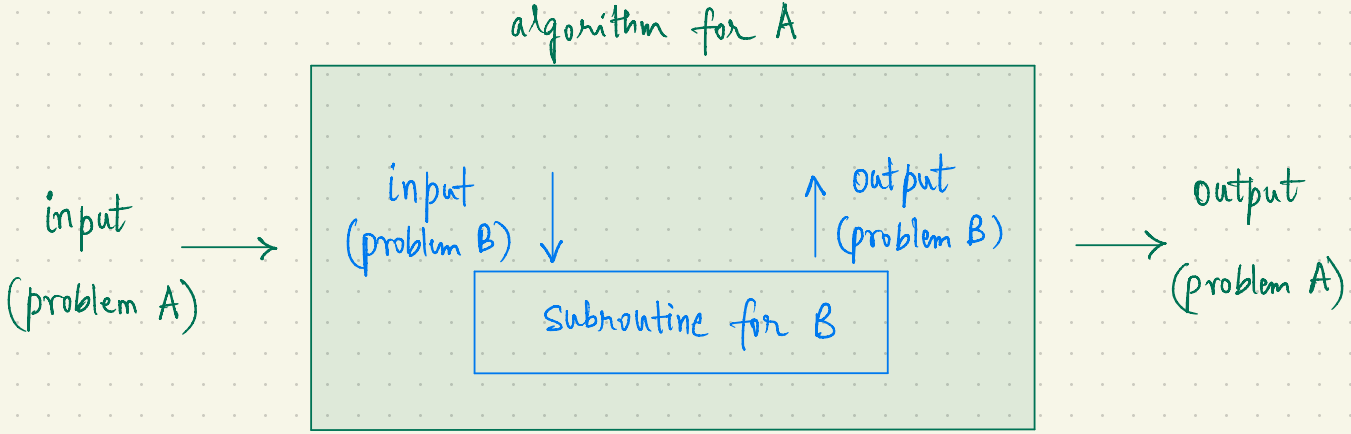
2. Spread intractability

USING REDUCTIONS TO DESIGN FAST ALGORITHMS

Using Reductions to Design Fast Algorithms



Using Reductions to Design Fast Algorithms



If A reduces to B and B can be solved in polynomial time

$\Rightarrow$ A can be solved in polynomial time.

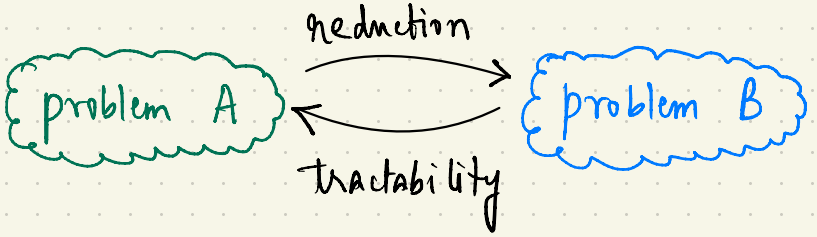
USING REDUCTIONS TO DESIGN FAST ALGORITHMS

1. Finding the median (A) of an array reduces to sorting (B) the array.

2. All-pairs shortest path (A) problem reduces to single-source shortest path (B).

3. Max flow problem with multiple sources/sinks (A) reduces to max flow problem with single source/sink (B).

NOTE:



Reductions

1. Spread tractability

2. Spread intractability

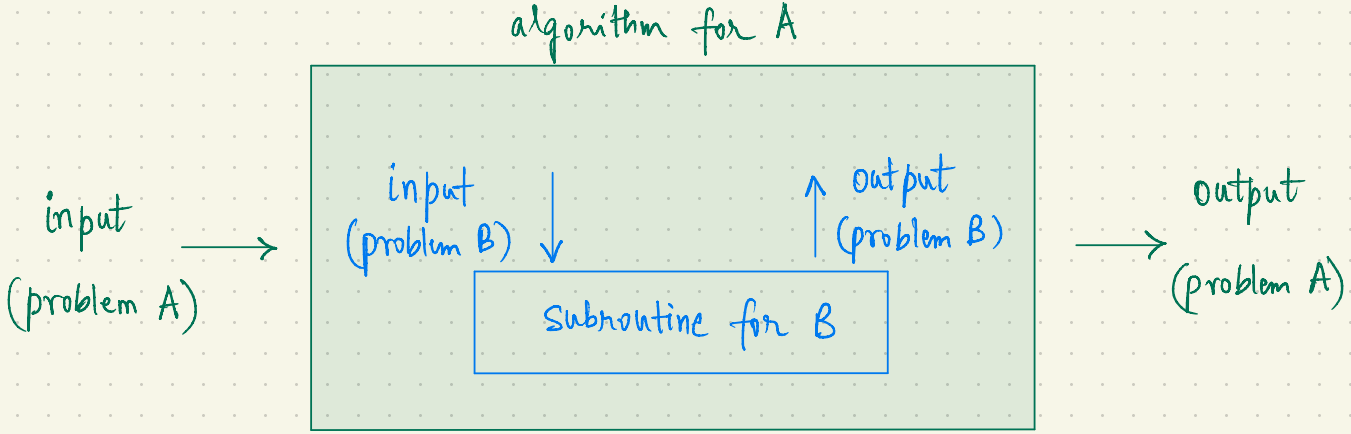
Reductions

1. Spread tractability

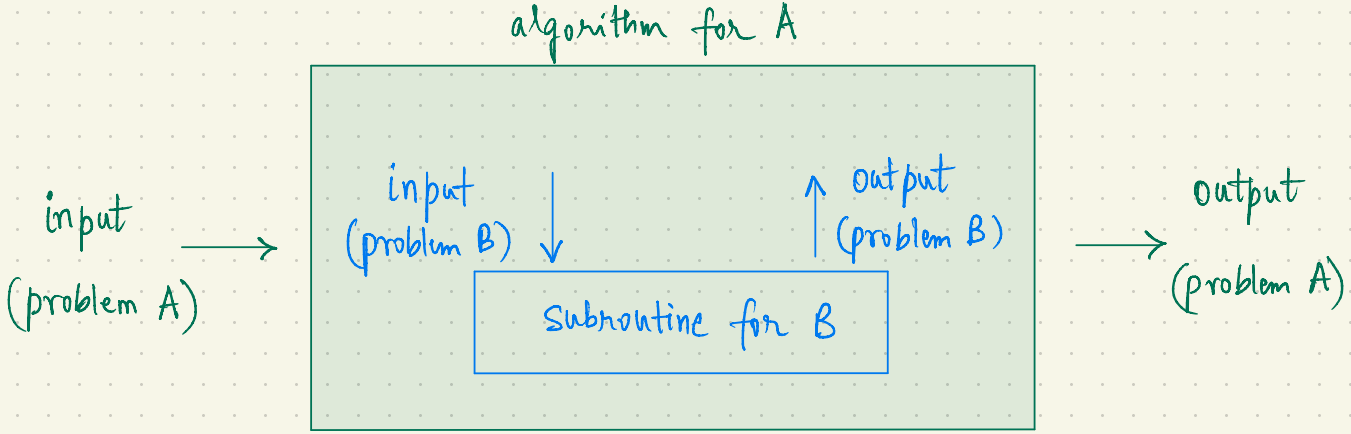
2. Spread intractability

USING REDUCTIONS TO SPREAD NP-HARDNESS

Using Reductions to Spread NP-Hardness

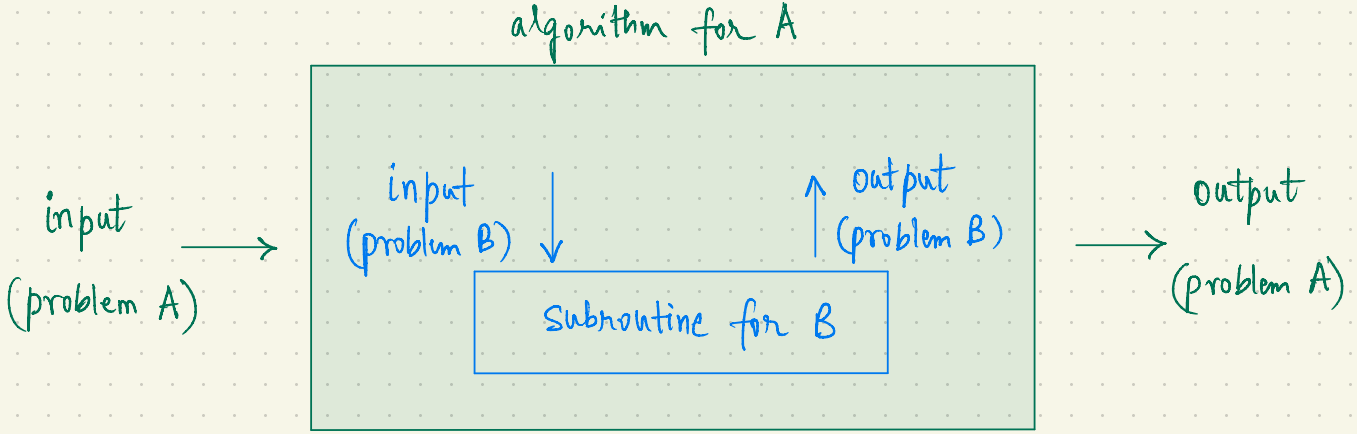


USING REDUCTIONS TO SPREAD NP-HARDNESS



If A reduces to B and A is NP-hard $\Rightarrow$ B is NP-hard

USING REDUCTIONS TO SPREAD NP-HARDNESS

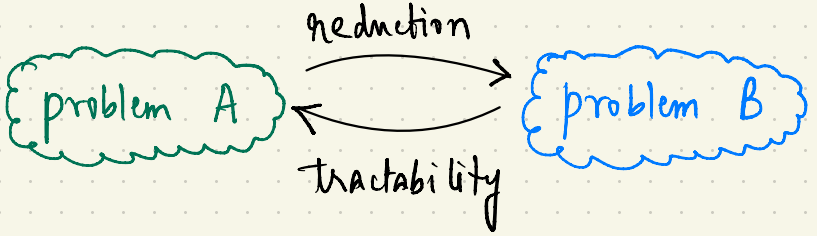


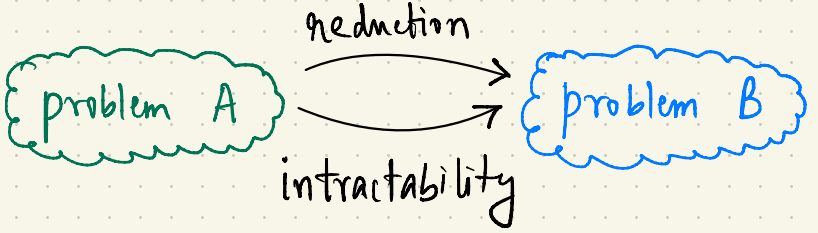
If A reduces to B and A is NP-hard $\Rightarrow$ B is NP-hard

Why?

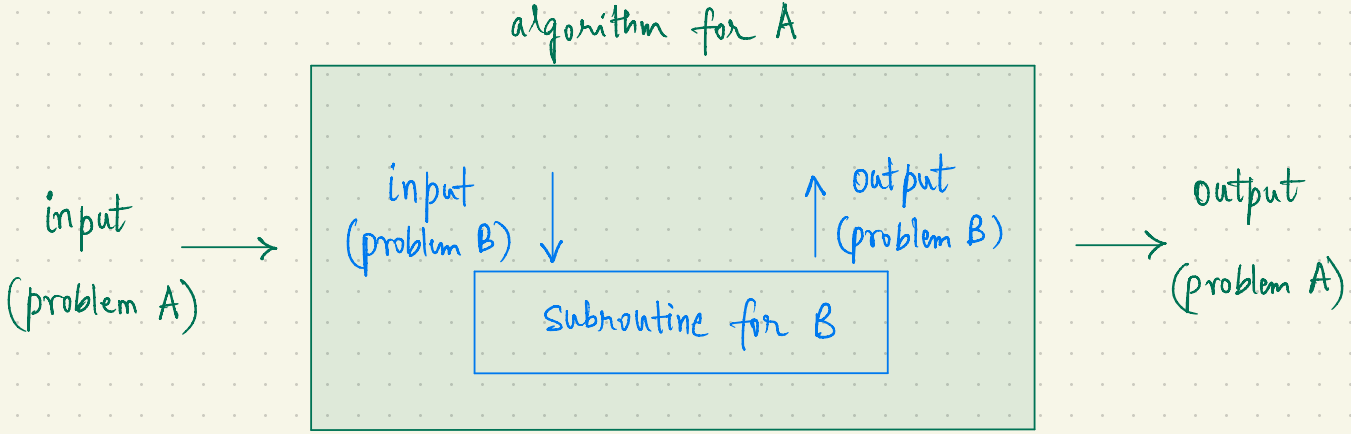
* Since A is NP-hard $\Rightarrow$ EVERY problem X in NP reduces to A.
* Reductions are transitive (X red. to A and A red. to B $\Rightarrow$ X red. to B).

Reductions


reduction
problem A
problem B
tractability

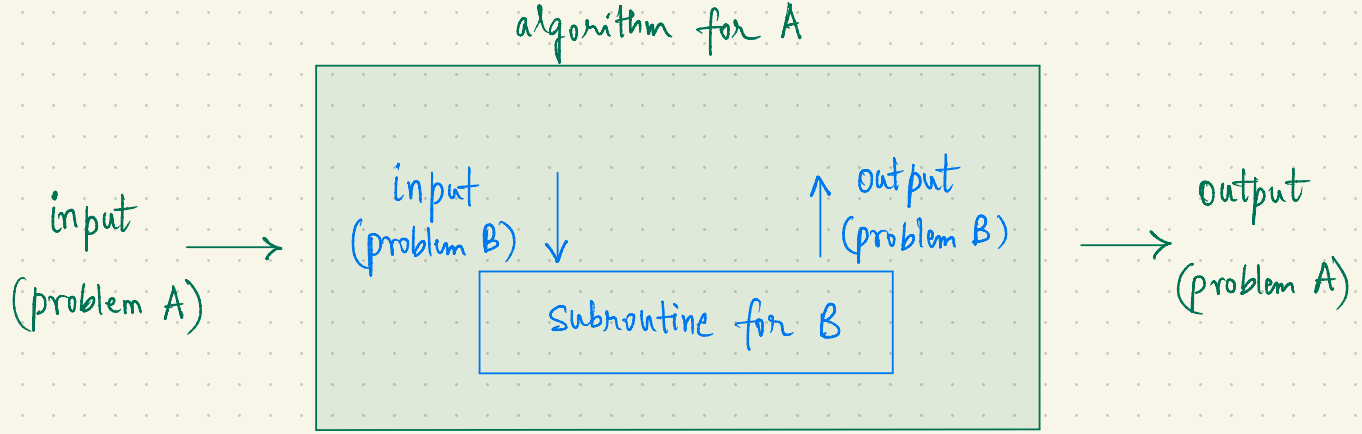

reduction
problem A
problem B
intractability

Using Reductions to Spread NP-Hardness



To prove problem B is NP-hard

Using Reductions to Spread NP-Hardness



To prove problem B is NP-hard

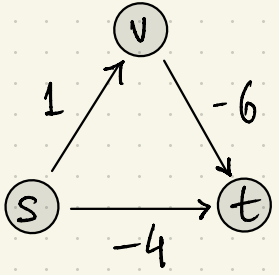
1. Identify a known NP-hard problem A.
2. Show that problem A reduces to problem B.

CYCLE-FREE SHORTEST PATHS

input: A directed graph $G = (V, E)$, starting vertex s, a possibly negative length ℓ_e for each edge $e \in E$

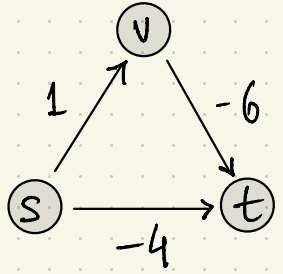
CYCLE-FREE SHORTEST PATHS

input: A directed graph $G = (V, E)$, starting vertex s, a possibly negative length ℓ_e for each edge $e \in E$



CYCLE-FREE SHORTEST PATHS

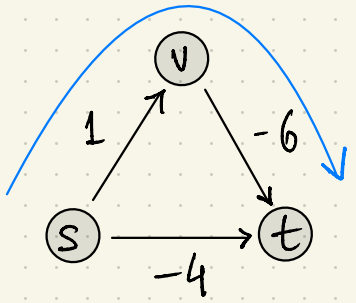
input: A directed graph $G = (V, E)$, starting vertex s, a possibly negative length ℓ_e for each edge $e \in E$



dist $(s, v) = 1$ // dist : length of shortest path

CYCLE-FREE SHORTEST PATHS

input: A directed graph $G = (V, E)$, starting vertex s, a possibly negative length ℓ_e for each edge $e \in E$

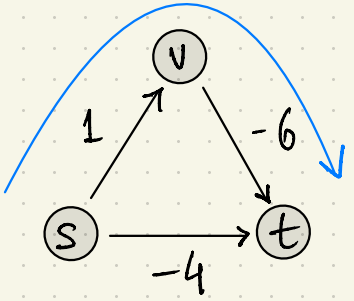


$dist(s, v) = 1$

$dist(s, t) = -5$

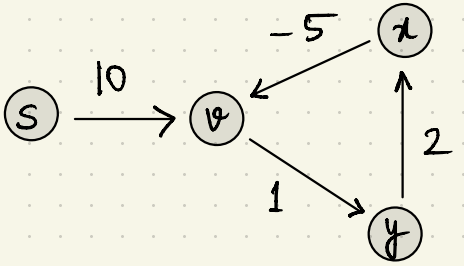
CYCLE-FREE SHORTEST PATHS

input: A directed graph $G = (V, E)$, starting vertex s, a possibly negative length ℓ_e for each edge $e \in E$



dist $(s, v) = 1$

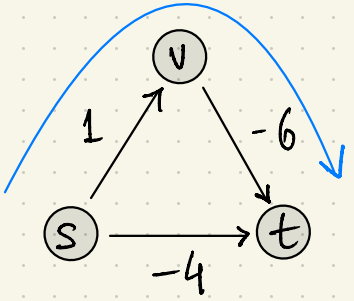
dist $(s, t) = -5$



dist $(s, v) = ?$

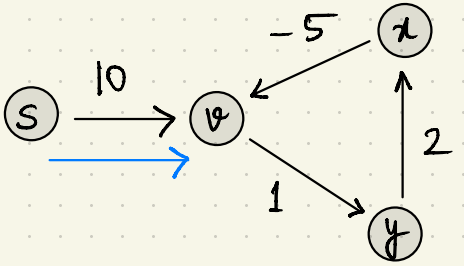
CYCLE-FREE SHORTEST PATHS

input: A directed graph $G = (V, E)$, starting vertex s, a possibly negative length ℓ_e for each edge $e \in E$



dist $(s, v) = 1$

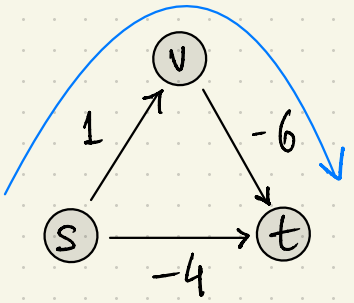
dist $(s, t) = -5$



dist $(s, v) = ?$

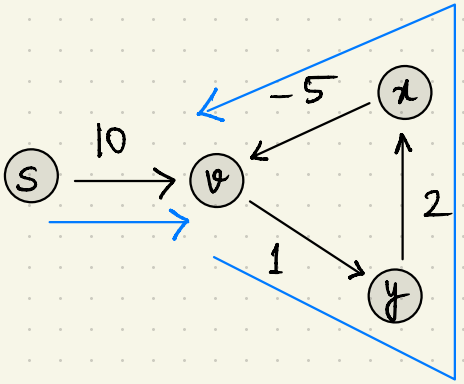
CYCLE-FREE SHORTEST PATHS

input: A directed graph $G = (V, E)$, starting vertex s, a possibly negative length ℓ_e for each edge $e \in E$



$dist(s, v) = 1$

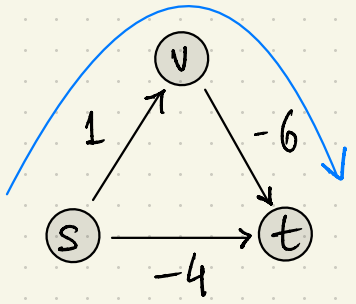
$dist(s, t) = -5$



$dist(s, v) = ?$

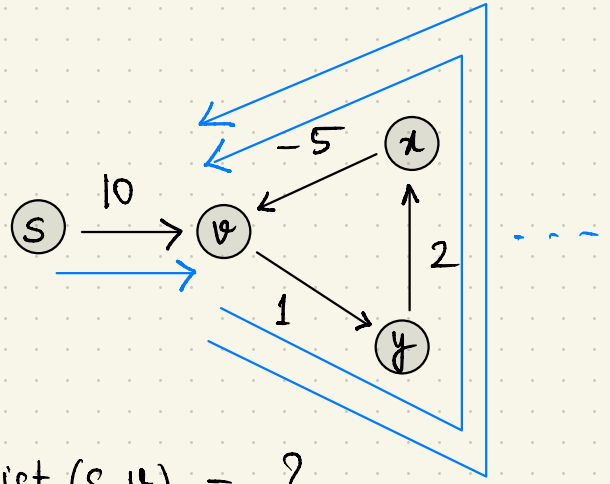
CYCLE-FREE SHORTEST PATHS

input: A directed graph $G = (V, E)$, starting vertex s, a possibly negative length ℓ_e for each edge $e \in E$



$dist(s, v) = 1$

$dist(s, t) = -5$



$dist(s, v) = ?$

CYCLE-FREE SHORTEST PATHS

input: A directed graph $G = (V, E)$, starting vertex s, a possibly negative length ℓ_e for each edge $e \in E$

output: for all destinations $v \in V$, compute the length of the shortest cycle-free $s-v$ path

CYCLE-FREE SHORTEST PATHS

input: A directed graph $G = (V, E)$, starting vertex s, a possibly negative length ℓ_e for each edge $e \in E$

output: for all destinations $v \in V$, compute the length of the shortest cycle-free $s-v$ path

Fact: This problem is NP-hard.

CYCLE-FREE SHORTEST PATHS

input: A directed graph $G = (V, E)$, starting vertex s, a possibly negative length ℓ_e for each edge $e \in E$

output: for all destinations $v \in V$, compute the length of the shortest cycle-free $s-v$ path

Fact: This problem is NP-hard.

Recall: Dijkstra, Bellman-Ford, and Floyd-Warshall all solve a special case of this problem when G has no negative cycle.

No poly-time algorithm known for the general problem.

CYCLE-FREE SHORTEST PATHS

input: A directed graph $G = (V, E)$, starting vertex s, a possibly negative length ℓ_e for each edge $e \in E$

output: for all destinations $v \in V$, compute the length of the shortest cycle-free $s-v$ path

Fact: This problem is NP-hard.

Will prove this fact by reducing FROM another NP-hard problem.

DIRECTED HAMILTONIAN PATH PROBLEM

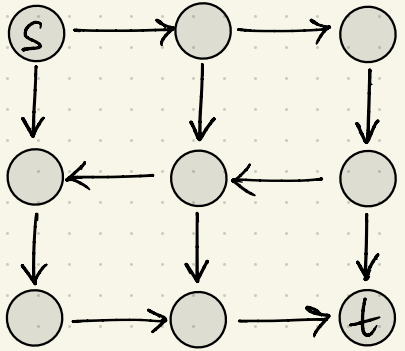
input: A directed graph $G = (V, E)$, a starting vertex $s \in V$, and an ending vertex $t \in V$.

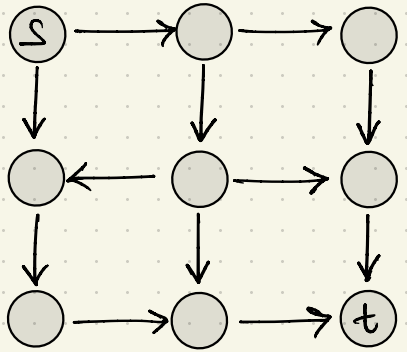
output: An s-t path visiting every vertex in V exactly once, if it exists, or "no" otherwise

DIRECTED HAMILTONIAN PATH PROBLEM

input: A directed graph $G = (V, E)$, a starting vertex $s \in V$, and an ending vertex $t \in V$.

output: An s-t path visiting every vertex in V exactly once, if it exists, or "no" otherwise

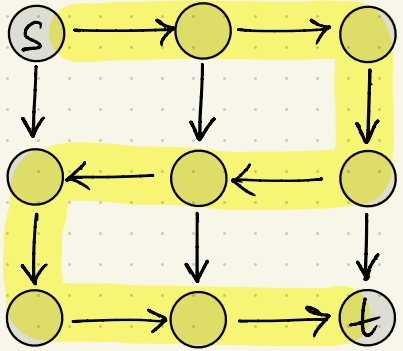


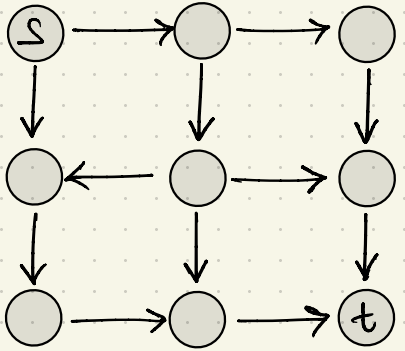


DIRECTED HAMILTONIAN PATH PROBLEM

input: A directed graph $G = (V, E)$, a starting vertex $s \in V$, and an ending vertex $t \in V$.

output: An s-t path visiting every vertex in V exactly once, if it exists, or "no" otherwise





None!

DIRECTED HAMILTONIAN PATH PROBLEM

input: A directed graph $G = (V, E)$, a starting vertex $s \in V$, and an ending vertex $t \in V$.

output: An s-t path visiting every vertex in V exactly once, if it exists, or "no" otherwise

Fact: Directed Hamiltonian Path is NP-hard.

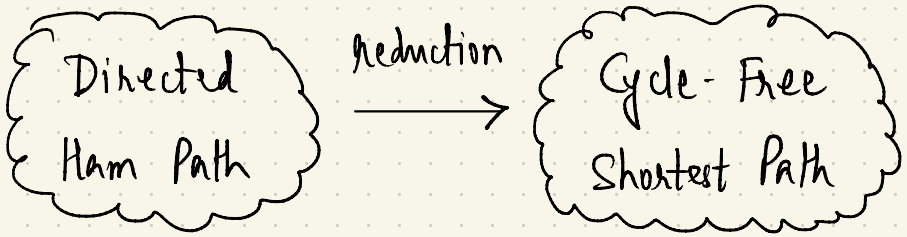
DIRECTED HAMILTONIAN PATH PROBLEM

input: A directed graph $G = (V, E)$, a starting vertex $s \in V$, and an ending vertex $t \in V$.

output: An s-t path visiting every vertex in V exactly once, if it exists, or "no" otherwise

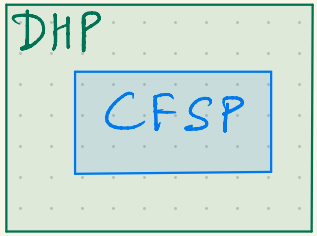
Fact: Directed Hamiltonian Path is NP-hard.

Coming up:



DIR. HAM. PATH REDUCES TO CYCLE-FREE SH. PATH

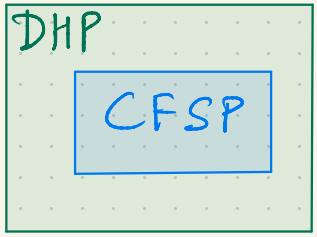
DIR. HAM. PATH REDUCES TO CYCLE-FREE SH. PATH



DIR. HAM. PATH REDUCES TO CYCLE-FREE SH. PATH

Given: an instance of Directed Hamiltonian Path:

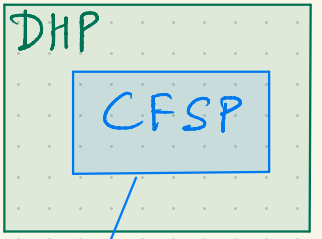
- a directed graph $G = (V, E)$
- a starting vertex $s \in V$
- an ending vertex $t \in V$



DIR. HAM. PATH REDUCES TO CYCLE-FREE SH. PATH

Given: an instance of Directed Hamiltonian Path:

- a directed graph $G = (V, E)$
- a starting vertex $s \in V$
- an ending vertex $t \in V$

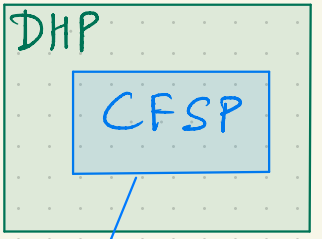


A directed graph $G = (V, E)$, starting vertex s, a possibly negative length ℓ_e for each edge $e \in E$

DIR. HAM. PATH REDUCES TO CYCLE-FREE SH. PATH

Given: an instance of Directed Hamiltonian Path:

- a directed graph $G = (V, E)$
- a starting vertex $s \in V$
- an ending vertex $t \in V$

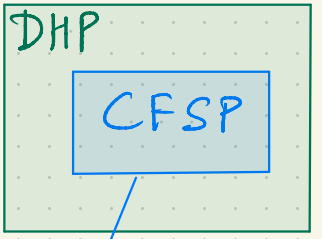


A directed graph $G = (V, E)$, starting vertex s, a possibly negative length ℓ_e for each edge $e \in E$

DIR. HAM. PATH REDUCES TO CYCLE-FREE SH. PATH

Given: an instance of Directed Hamiltonian Path:

- a directed graph $G = (V, E)$
- a starting vertex $s \in V$
- an ending vertex $t \in V$

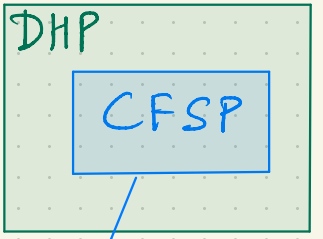


A directed graph $G = (V, E)$, starting vertex s, a possibly negative length ℓ_e for each edge $e \in E$

DIR. HAM. PATH REDUCES TO CYCLE-FREE SH. PATH

Given: an instance of Directed Hamiltonian Path:

- a directed graph $G = (V, E)$
- a starting vertex $s \in V$
- an ending vertex $t \in V$



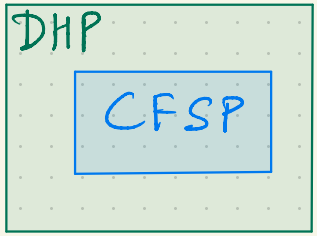
A directed graph $G = (V, E)$, starting vertex s,
a possibly negative length ℓ_e for each edge $e \in E$

?

DIR. HAM. PATH REDUCES TO CYCLE-FREE SH. PATH

Given: an instance of Directed Hamiltonian Path:

- a directed graph $G = (V, E)$
- a starting vertex $s \in V$
- an ending vertex $t \in V$





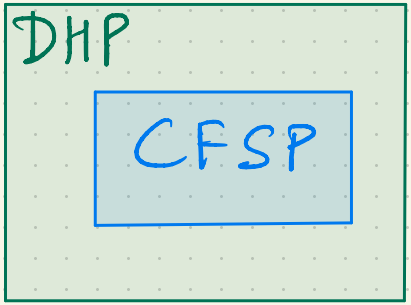
Trick the CFSP subroutine into thinking that "long" paths are actually "short".

Set $l_e = -1$ for all edges $e \in E$.

DIR. HAM. PATH REDUCES TO CYCLE-FREE SH. PATH

Given: an instance of Directed Hamiltonian Path:

- a directed graph $G = (V, E)$
- a starting vertex $s \in V$
- an ending vertex $t \in V$

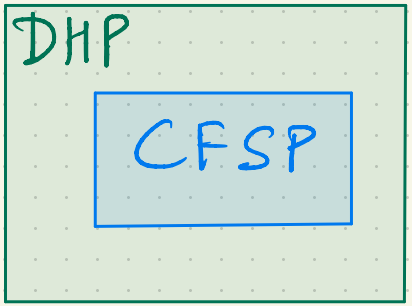


DIR. HAM. PATH REDUCES TO CYCLE-FREE SH. PATH

Given: an instance of Directed Hamiltonian Path:

- a directed graph $G = (V, E)$
- a starting vertex $s \in V$
- an ending vertex $t \in V$

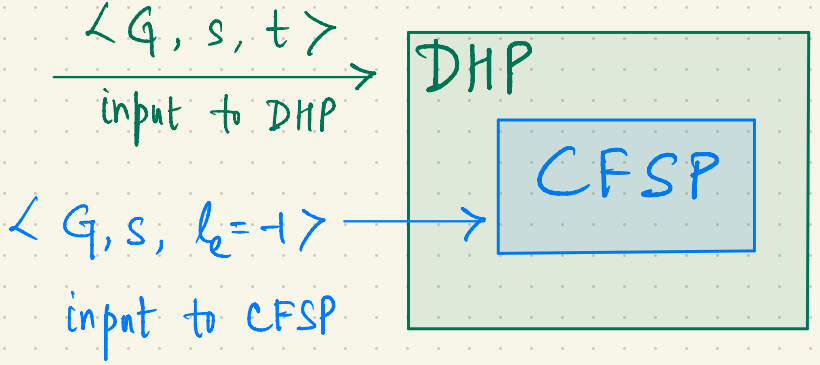
$\langle G, s, t \rangle$ → input to DHP



DIR. HAM. PATH REDUCES TO CYCLE-FREE SH. PATH

Given: an instance of Directed Hamiltonian Path:

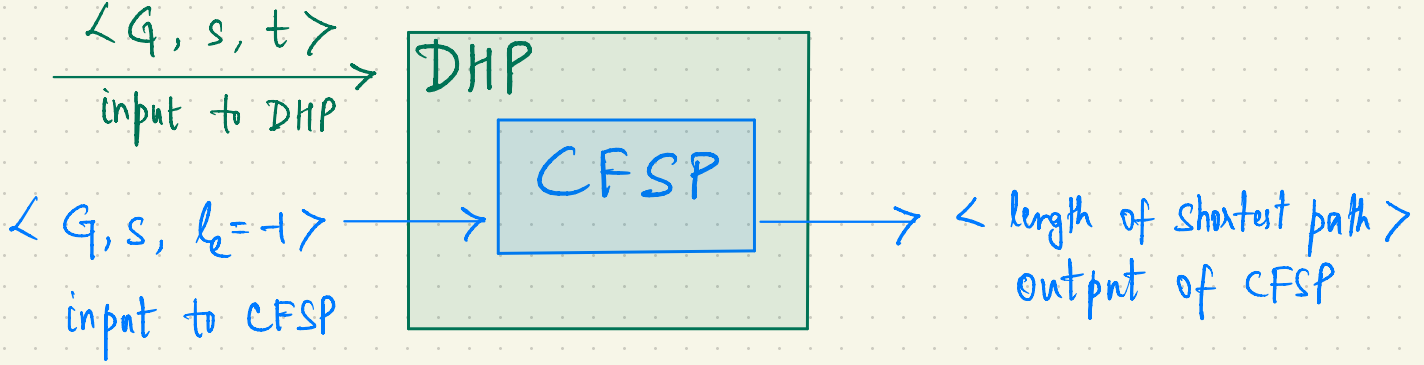
- a directed graph $G = (V, E)$
- a starting vertex $s \in V$
- an ending vertex $t \in V$



DIR. HAM. PATH REDUCES TO CYCLE-FREE SH. PATH

Given: an instance of Directed Hamiltonian Path:

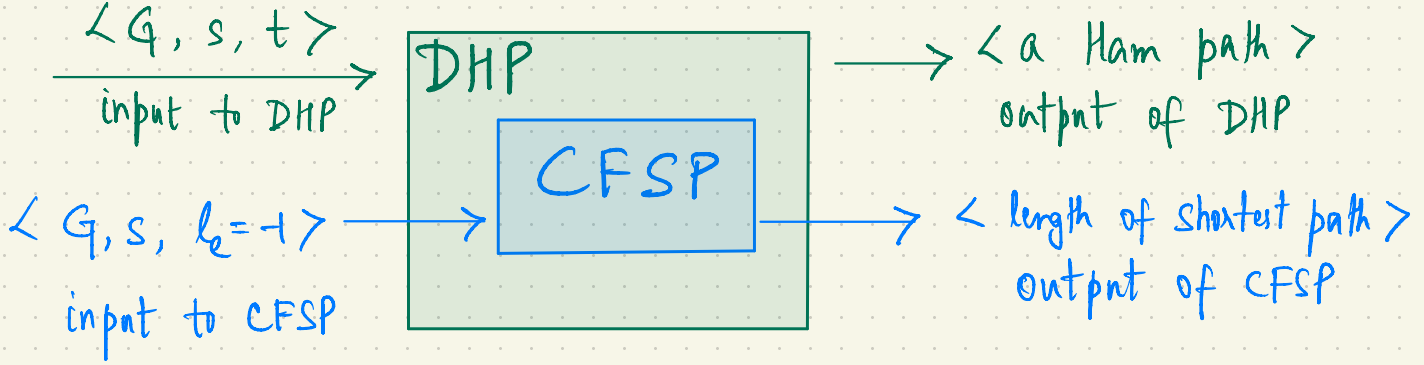
- a directed graph $G = (V, E)$
- a starting vertex $s \in V$
- an ending vertex $t \in V$



DIR. HAM. PATH REDUCES TO CYCLE-FREE SH. PATH

Given: an instance of Directed Hamiltonian Path:

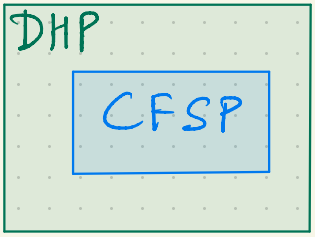
- a directed graph $G = (V, E)$
- a starting vertex $s \in V$
- an ending vertex $t \in V$



DIR. HAM. PATH REDUCES TO CYCLE-FREE SH. PATH

Given: an instance of Directed Hamiltonian Path:

- a directed graph $G = (V, E)$
- a starting vertex $s \in V$
- an ending vertex $t \in V$

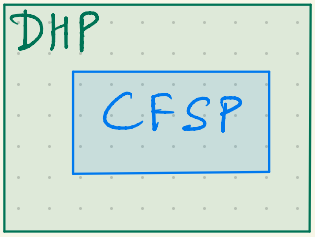


① Set $l_e = -1$ for all edges $e \in E$.

DIR. HAM. PATH REDUCES TO CYCLE-FREE SH. PATH

Given: an instance of Directed Hamiltonian Path:

- a directed graph $G = (V, E)$
- a starting vertex $s \in V$
- an ending vertex $t \in V$



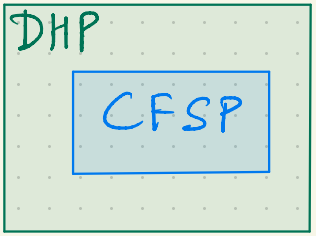
① Set $l_e = -1$ for all edges $e \in E$.

② Invoke CFSP subroutine.

DIR. HAM. PATH REDUCES TO CYCLE-FREE SH. PATH

Given: an instance of Directed Hamiltonian Path:

- a directed graph $G = (V, E)$
- a starting vertex $s \in V$
- an ending vertex $t \in V$



① Set $l_e = -1$ for all edges $e \in E$.

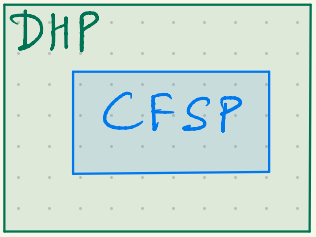
② Invoke CFSP subroutine.

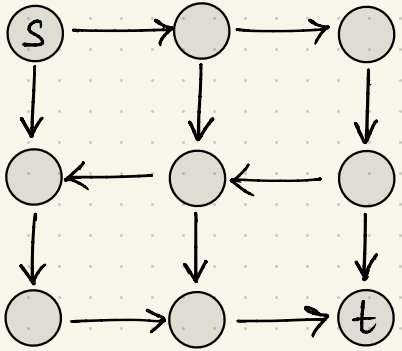
③ If dist $(s,t) = -(n-1)$ return YES (where $n = |V|$)
else return NO.

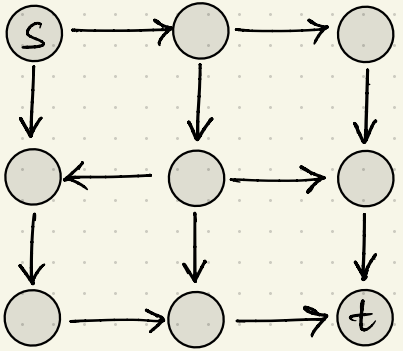
DIR. HAM. PATH REDUCES TO CYCLE-FREE SH. PATH

Given: an instance of Directed Hamiltonian Path:

- a directed graph $G = (V, E)$
- a starting vertex $s \in V$
- an ending vertex $t \in V$



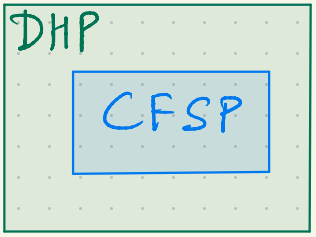


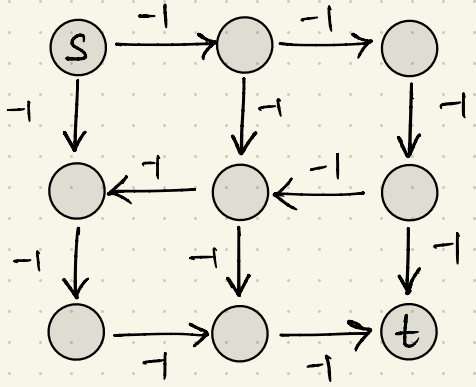


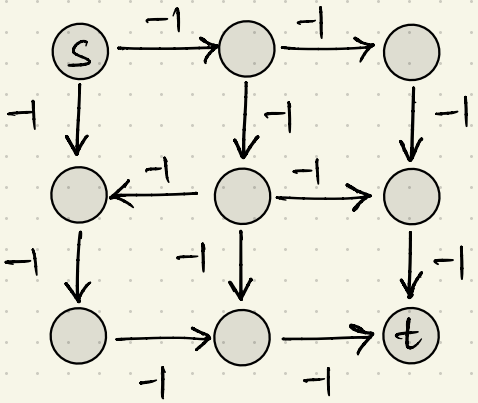
DIR. HAM. PATH REDUCES TO CYCLE-FREE SH. PATH

Given: an instance of Directed Hamiltonian Path:

- a directed graph $G = (V, E)$
- a starting vertex $s \in V$
- an ending vertex $t \in V$



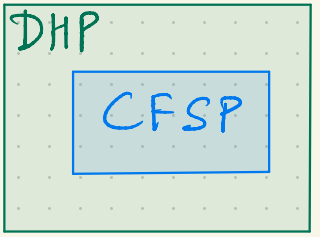


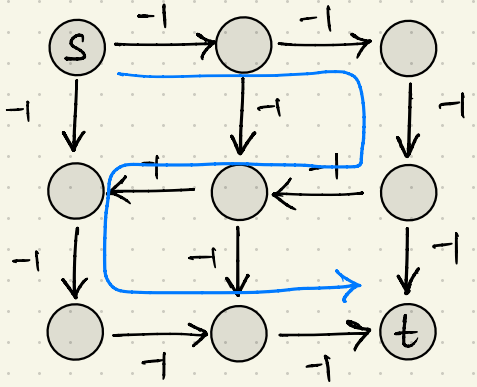


DIR. HAM. PATH REDUCES TO CYCLE-FREE SH. PATH

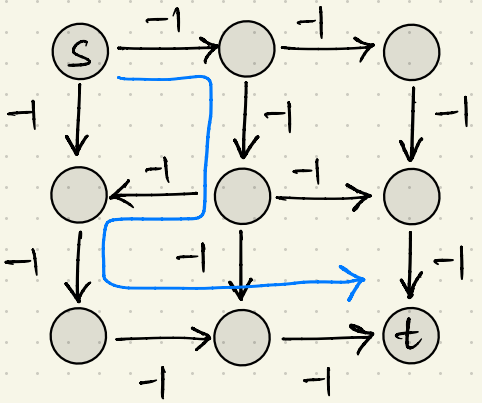
Given: an instance of Directed Hamiltonian Path:

- a directed graph $G = (V, E)$
- a starting vertex $s \in V$
- an ending vertex $t \in V$





length of shortest s-t path = -8

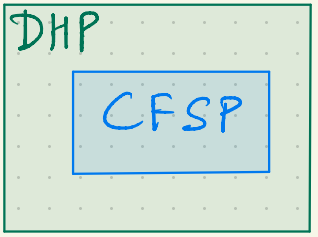


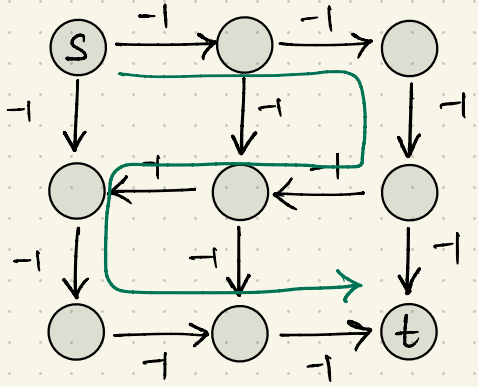
No s-t path with length -8

DIR. HAM. PATH REDUCES TO CYCLE-FREE SH. PATH

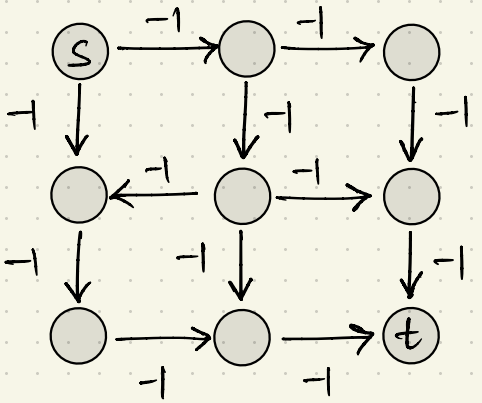
Given: an instance of Directed Hamiltonian Path:

- a directed graph $G = (V, E)$
- a starting vertex $s \in V$
- an ending vertex $t \in V$





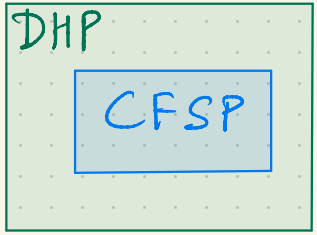
YES



NO

DIR. HAM. PATH REDUCES TO CYCLE-FREE SH. PATH

Proof:

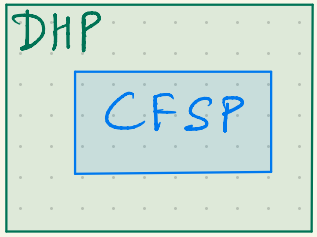


DIR. HAM. PATH REDUCES TO CYCLE-FREE SH. PATH

Proof: Given an instance $\langle G, s, t \rangle$ of DHP,

create an instance $\langle G, s, \{\ell_e\}_{e \in E} \rangle$ of CFSP

where $\ell_e = -1$ for every $e \in E$.



DIR. HAM. PATH REDUCES TO CYCLE-FREE SH. PATH

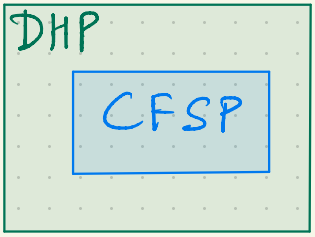
Proof: Given an instance $\langle G, s, t \rangle$ of DHP,

create an instance $\langle G, s, \{l_e\}_{e \in E} \rangle$ of CFSP

where $l_e = -1$ for every $e \in E$.

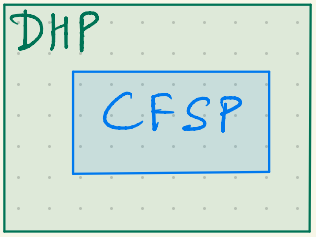
Return "YES" for DHP if and only if

the output of CFSP for (s,t) is $-(n-1)$.



DIR. HAM. PATH REDUCES TO CYCLE-FREE SH. PATH

Proof: (a) If a Ham Path exists in DHP instance:

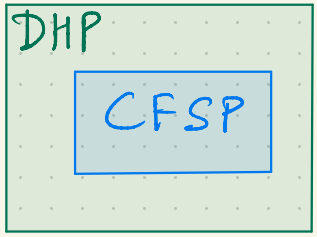


(b) If no Ham Path exists in DHP instance:

DIR. HAM. PATH REDUCES TO CYCLE-FREE SH. PATH

Proof: (a) If a Ham Path exists in DHP instance:

Then there is an s-t path of length $-(n-1)$ in the CFSP instance.



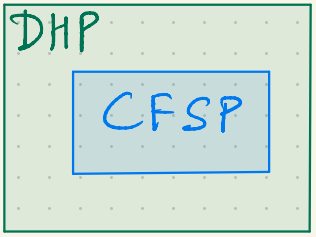
(b) If no Ham Path exists in DHP instance:

DIR. HAM. PATH REDUCES TO CYCLE-FREE SH. PATH

Proof: (a) If a Ham Path exists in DHP instance:

Then there is an s-t path of length $-(n-1)$ in the CFSP instance.

Any path shorter than this must revisit a vertex and is therefore not cycle-free.



(b) If no Ham Path exists in DHP instance:

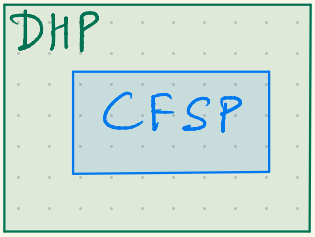
DIR. HAM. PATH REDUCES TO CYCLE-FREE SH. PATH

Proof: (a) If a Ham Path exists in DHP instance:

Then there is an s-t path of length $-(n-1)$ in the CFSP instance.

Any path shorter than this must revisit a vertex and is therefore not cycle-free.

Thus, the DHP algorithm correctly returns YES in this case



(b) If no Ham Path exists in DHP instance:

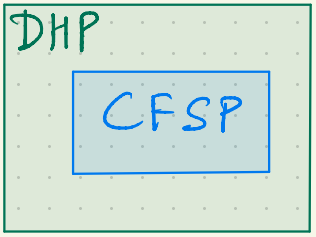
DIR. HAM. PATH REDUCES TO CYCLE-FREE SH. PATH

Proof: (a) If a Ham Path exists in DHP instance:

Then there is an s-t path of length $(n-1)$ in the CFSP instance.

Any path shorter than this must revisit a vertex and is therefore not cycle-free.

Thus, the DHP algorithm correctly returns YES in this case



(b) If no Ham Path exists in DHP instance:

Then, any cycle-free s-t path in the DHP instance has fewer than $(n-1)$ hops. Thus, the DHP algorithm correctly returns NO in this case.

∎

WHERE IT ALL BEGAN

WHERE IT ALL BEGAN

SAT

input : A list of Boolean decision variables $x_1, x_2, \ldots, x_n$
a list of constraints, each an OR of one or more literals

output: A truth assignment to $x_1, x_2, \ldots, x_n$ that satisfies every constraint if one exists, or "no solution" otherwise

WHERE IT ALL BEGAN

3-SAT

input: A list of Boolean decision variables $x_1, x_2, \ldots, x_n$
a list of constraints, each an OR of at most three literals

output: A truth assignment to $x_1, x_2, \ldots, x_n$ that satisfies every constraint if one exists, or "no solution" otherwise

WHERE IT ALL BEGAN

3-SAT

input: A list of Boolean decision variables $x_1, x_2, \ldots, x_n$
a list of constraints, each an OR of at most three literals

output: A truth assignment to $x_1, x_2, \ldots, x_n$ that satisfies every constraint if one exists, or "no solution" otherwise

e.g., $(x_1 \vee x_2 \vee x_3) \wedge (x_6 \vee \neg x_1 \vee x_5) \wedge (\neg x_4 \vee \neg x_1 \vee \neg x_2)$

WHERE IT ALL BEGAN

3-SAT

input: A list of Boolean decision variables $x_1, x_2, \ldots, x_n$
a list of constraints, each an OR of at most three literals

output: A truth assignment to $x_1, x_2, \ldots, x_n$ that satisfies every constraint if one exists, or "no solution" otherwise

3-SAT is a special case of SAT $\Rightarrow$

3-SAT reduces to SAT ?

or

SAT reduces to 3-SAT ?

WHERE IT ALL BEGAN

3-SAT

input: A list of Boolean decision variables $x_1, x_2, \ldots, x_n$
a list of constraints, each an OR of at most three literals

output: A truth assignment to $x_1, x_2, \ldots, x_n$ that satisfies every constraint if one exists, or "no solution" otherwise

✓ 3-SAT reduces to SAT ?

3-SAT is a special case of SAT $\Rightarrow$ or

SAT reduces to 3-SAT ?

WHERE IT ALL BEGAN

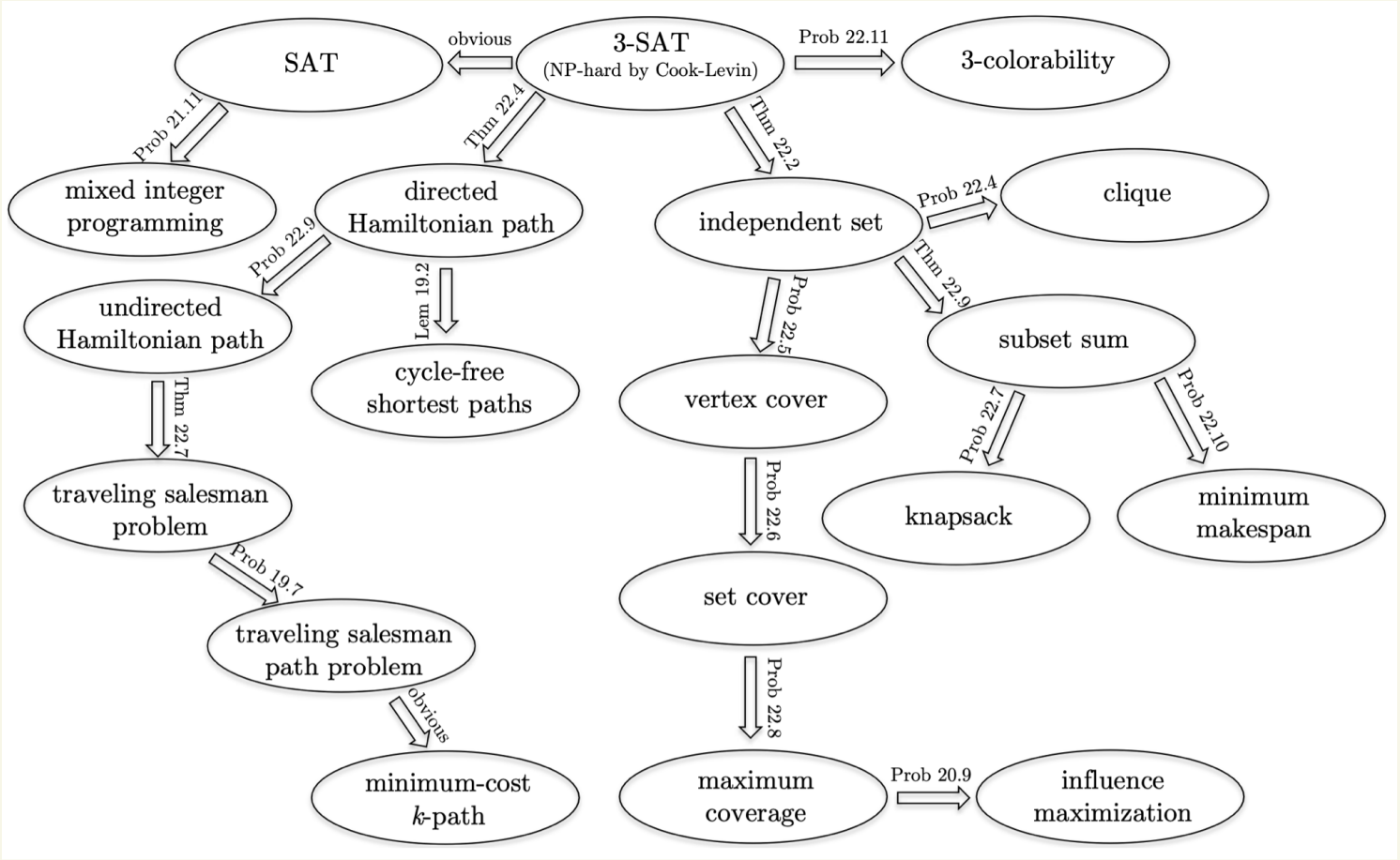
3-SAT

input: A list of Boolean decision variables $x_1, x_2, \ldots, x_n$
a list of constraints, each an OR of at most three literals

output: A truth assignment to $x_1, x_2, \ldots, x_n$ that satisfies every constraint if one exists, or "no solution" otherwise

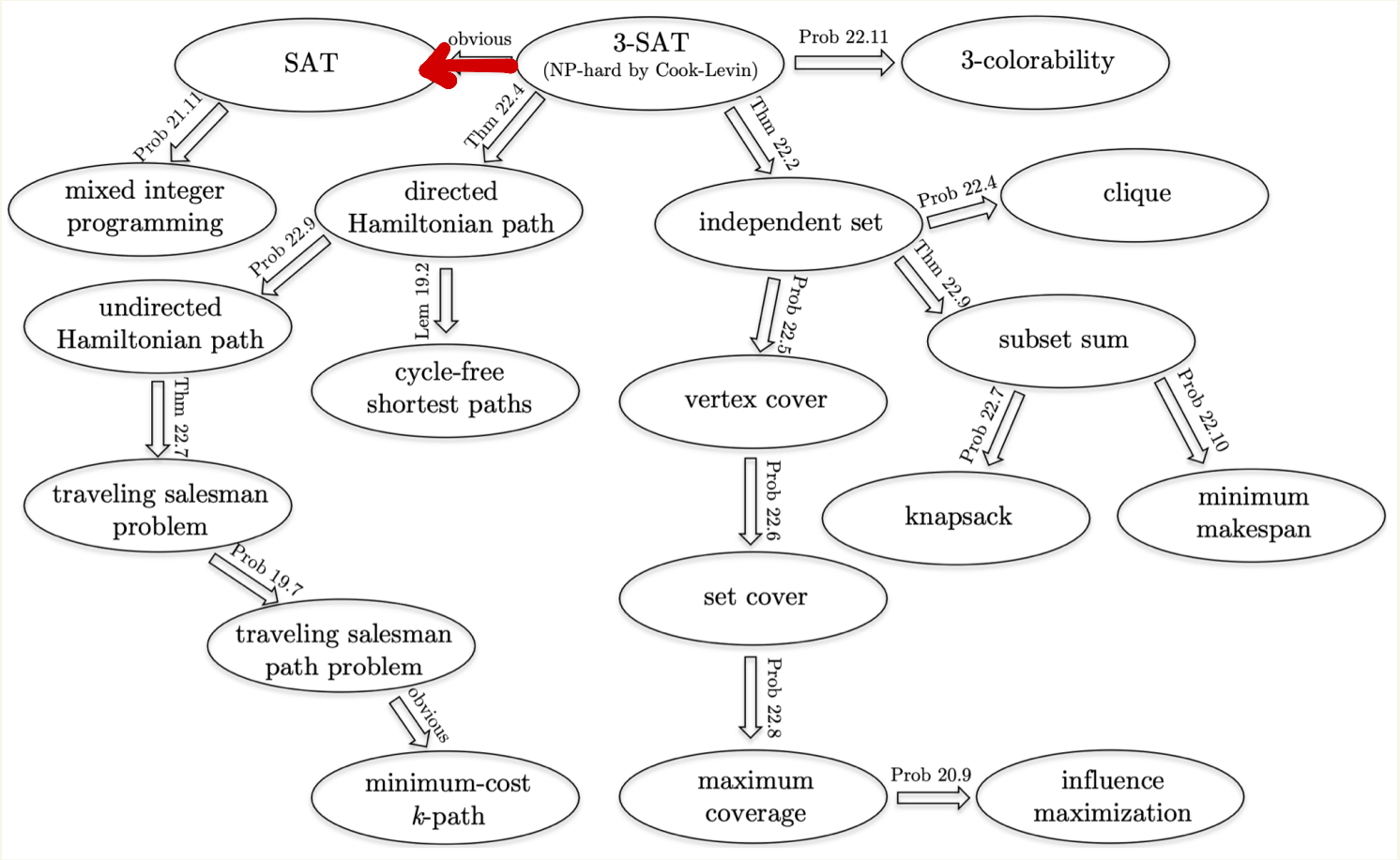
Cook-Levin Theorem: 3-SAT is NP-hard.

WEB OF REDUCTIONS



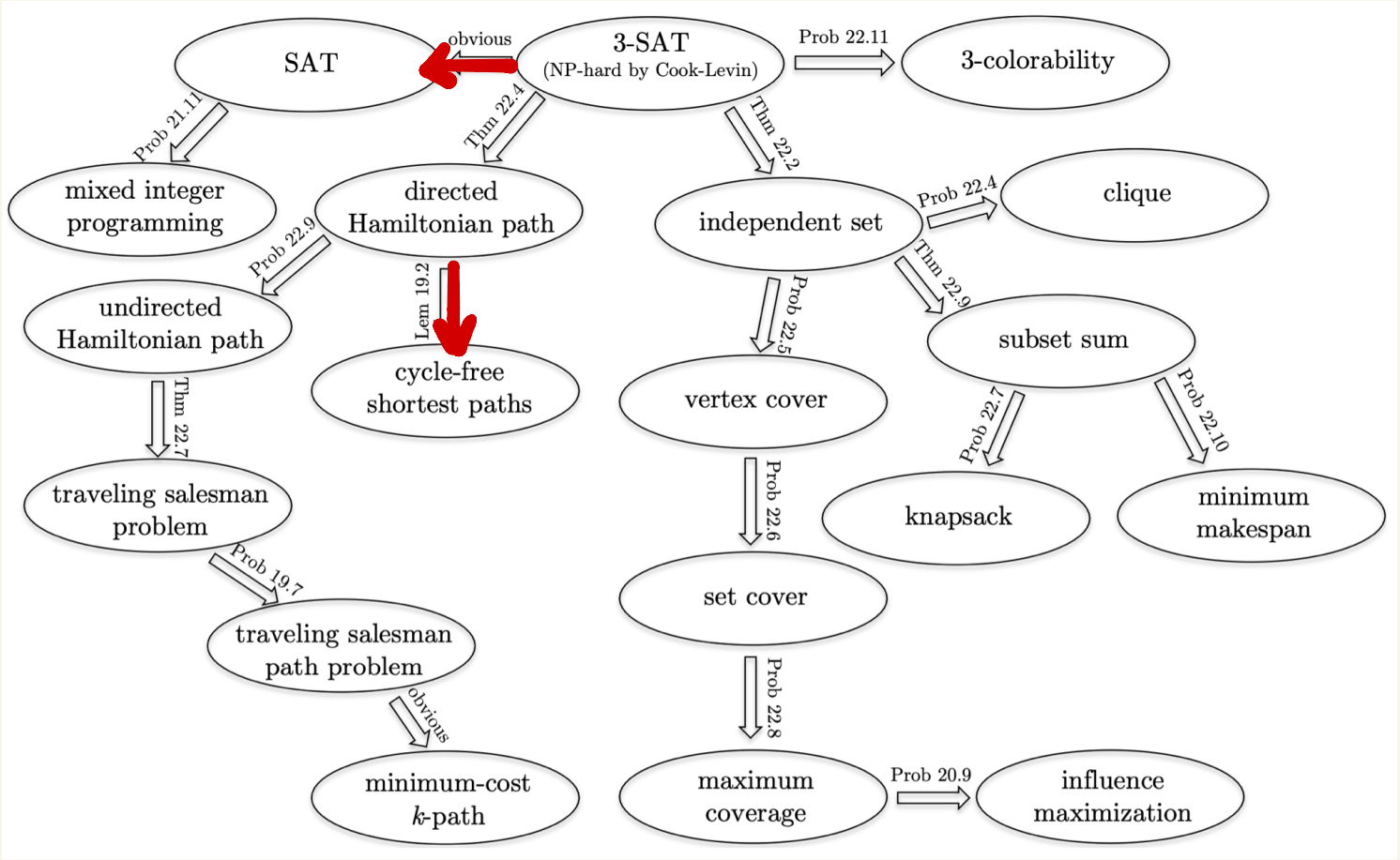
Source: "Algorithms Illuminated" Part 4, Sec 22.3.2

WEB OF REDUCTIONS



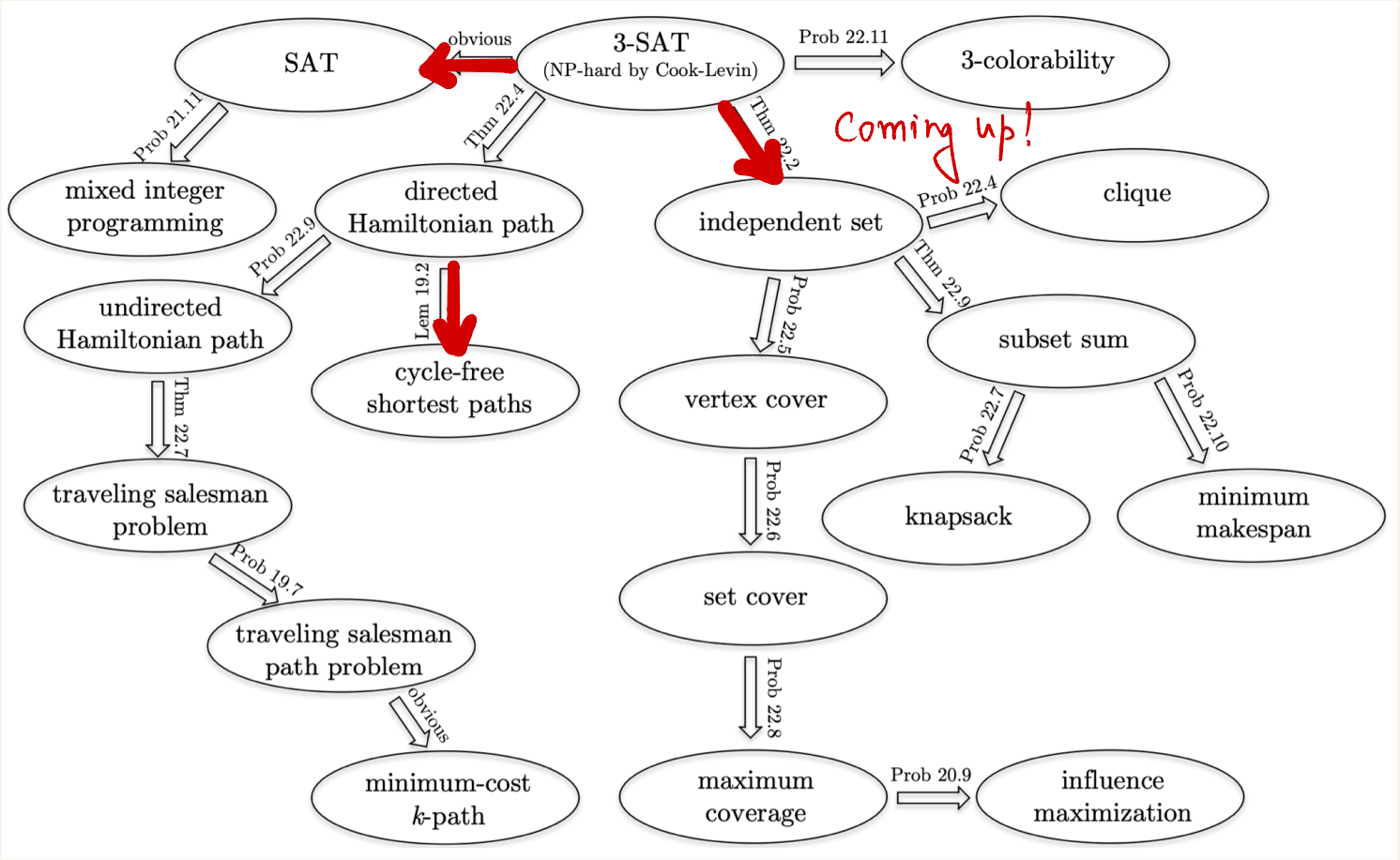
Source: "Algorithms Illuminated" Part 4, Sec 22.3.2

WEB OF REDUCTIONS



Source: "Algorithms Illuminated" Part 4, Sec 22.3.2

WEB OF REDUCTIONS



Source: "Algorithms Illuminated" Part 4, Sec 22.3.2

Independent Set Problem

INDEPENDENT SET PROBLEM

input : undirected graph $G = (V, E)$

output : independent set $S \subseteq V$ of maximum possible size

INDEPENDENT SET PROBLEM

input : undirected graph $G = (V, E)$

output : independent set $S \subseteq V$ of maximum possible size

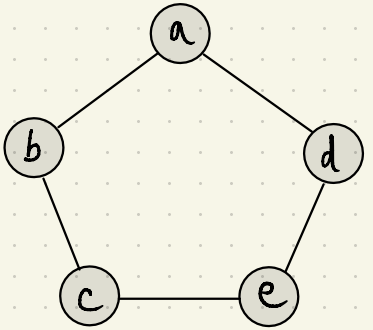
mutually non-adjacent vertices

INDEPENDENT SET PROBLEM

input : undirected graph $G = (V, E)$

output : independent set $S \subseteq V$ of maximum possible size

mutually non-adjacent vertices

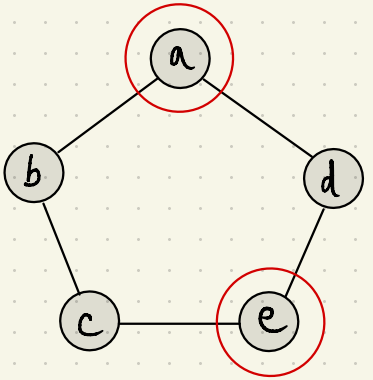


INDEPENDENT SET PROBLEM

input : undirected graph $G = (V, E)$

output : independent set $S \subseteq V$ of maximum possible size

mutually non-adjacent vertices



INDEPENDENT SET PROBLEM

input : undirected graph $G = (V, E)$

output : independent set $S \subseteq V$ of maximum possible size

Theorem : Independent Set is NP-hard.

INDEPENDENT SET PROBLEM

input : undirected graph $G = (V, E)$

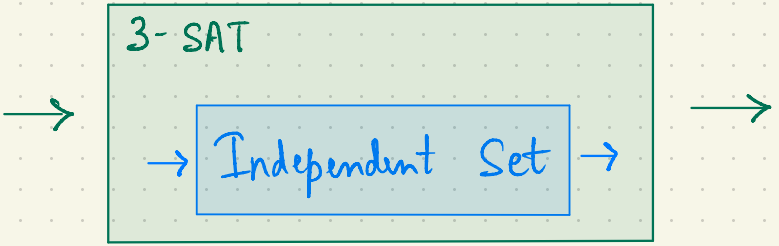
output : independent set $S \subseteq V$ of maximum possible size

Theorem : Independent Set is NP-hard.

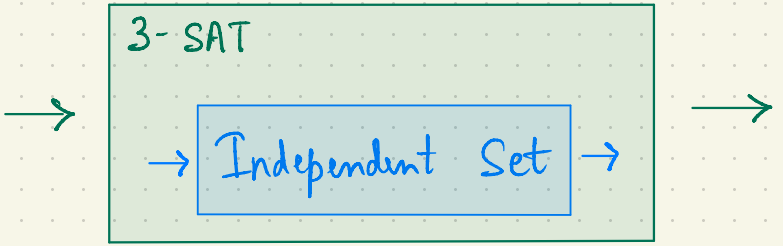
Proof : By reduction from 3-SAT.

THE PLAN

THE PLAN



THE PLAN



Need to translate a logic problem (3-SAT) into a graph problem (IS).

Extract a solution to 3-SAT from max. size independent set.