Design and Analysis of Algorithms

Algorithms-II : CS345A

Website: hello.iitk.ac.in

Lecture 30

NP Completeness – II

This lecture is going to be

Reasons:

The theory of NP class and NP complete class of problems took decades to get developed. So it is not justified that one can quickly understand the way this class is defined and the reason behind it.

Advice:

Go over the lecture slides with open mind.

On some slides, you will find formulation/definition to capture a class of problems.

If you don't find a formulation/definition <u>convincing</u>,

discard it <u>temporarily</u> and

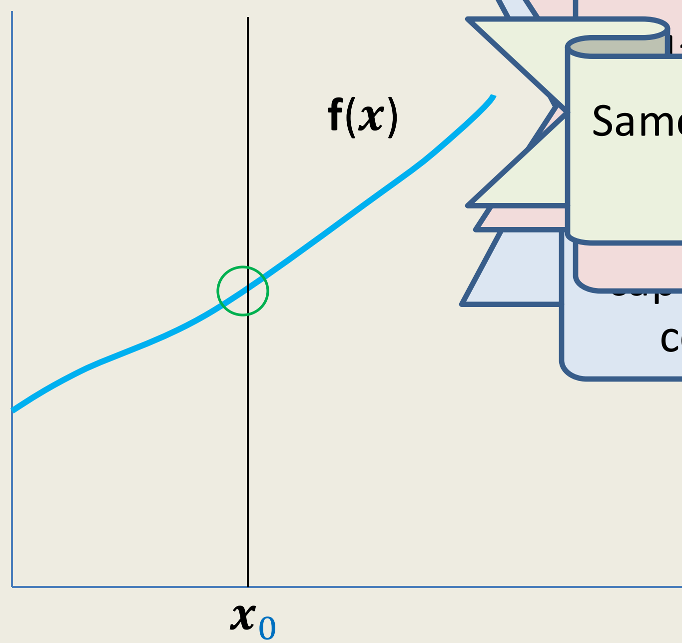
search on your own for an alternate formulation

Now revisit the formulation in the slide.

This should help you understand the formulation in a better way.

You are of course welcome to have a discussion with me.

Definition of Continuous function



Same is going to be the case of NP problems ☺.

continuous function.

Definition:

A function is said to be continuous at point x_0,

if for each $\delta > 0$,

for every x

RECAP FROM LAST LECTURE

Optimization version ➔ Decision version

$$\boldsymbol{A} \leq_{\boldsymbol{P}} \boldsymbol{B}$$

$\boldsymbol{A \leq_P B}$

Complexity theoretic consequence of $\boldsymbol{A \leq_P B}$:

There does not exist any polynomial time algorithm for $\boldsymbol{A}$,

➔There <u>can not </u>exist any polynomial time algorithm for $\boldsymbol{B}$.

"$\boldsymbol{B}$ is computationally **at least** as hard as $\boldsymbol{A}$"

NP
A CLASS OF PROBLEMS

Go back to 1960's

Efficient algorithm was found.	No Efficient algorithm could be designed till date
Shortest Path	Longest Path
Minimum spanning Tree	Travelling salesman Problem
Euler tour	Hamiltonian cycle
Min Cut	Balanced Cut
Independent Set on trees	Independent Set
Bipartite matching	3D matching
Linear Programming	Integer Linear Programming
⋮	⋮

This motivated researchers to search for any common traits among all these problems.

It was quite surprising and even frustrating to be unable to find efficient algorithm for so many problems when their similar looking versions had very efficient algorithms.

- **Travelling Salesman Problem**

Decision version: Given a graph $\boldsymbol{G}$, does there exist a tour of cost at most $\boldsymbol{b}$?

Searching for a tour of cost at most $\boldsymbol{b}$ appears to be difficult ☹

But what about **verifying** whether a given sequence of vertices is a tour of cost at most $\boldsymbol{b}$?

It is quite easy ☺.

- **Vertex cover**

Decision version: Given a graph $\boldsymbol{G}$, does there exist a vertex cover of size $\le \boldsymbol{k}$.

Searching for a subset of $\boldsymbol{k}$ vertices that is a vertex cover of $\boldsymbol{G}$ appears difficult ☹.

But what about **verifying** whether a given subset of $\boldsymbol{k}$ vertices is a vertex cover ?

It is quite easy ☺.

What about verifying a proposed solution ?

Easy

No Efficient algorithm till date ☹	
Longest Path	Is there a path of length $\geq \boldsymbol{k}$ in $\boldsymbol{G}$?
Vertex cover	Does there exist a vertex cover of size $\leq \boldsymbol{k}$ in $\boldsymbol{G}$?
Travelling salesman Problem	Does there exist a tour of cost $\leq \boldsymbol{c}$ in $\boldsymbol{G}$?
Hamiltonian cycle	Does there exist a cycle of length $\boldsymbol{n}$ in $\boldsymbol{G}$?
Independent Set	Does there exist an independent set of size $\geq \boldsymbol{k}$ in $\boldsymbol{G}$?
3D matching	⋮
Integer Linear Programming	⋮
⋮	

What if the answer of an instance is **Yes** ?

There is a short *certificate*.

No Efficient algorithm till date ☹

Longest Path	
Vertex cover	
Travelling salesman Problem	
Hamiltonian cycle	
Independent Set	
3D matching	
Integer Linear Programming	
⋮	

short certificate

Search: difficult

verification: **easy**

Efficient certifier

We shall redefine the behavior of A. Ponder over the new definition.

X : any decision problem

Yes instance

I : any (input) instance of X

No instance

certifier for X :

algorithm A with output {yes,no}

How to capture the fact that A is efficient?

- **Input** :

Proposed solution

- **Behavior**: A can verify if proposed solution s is right or wrong.

Efficient certifier

X : any decision problem

Yes instance

I : any (input) instance of X

No instance

Ponder over the redefined behavior of A. Take your time …

Efficient certifier for X :

A polynomial time algorithm A with output {yes,no}

- **Input** : (I, s)

  Proposed solution

- **Behavior**: There is a polynomial function p such that

  I is yes-instance of X **if and only if**

  there exists a string s

Efficient certifier

Examples

	Efficient certifiers:
Longest Path	Determines if the given string s is a indeed path of length $\geq k$ in G
Vertex cover	Determines if the given string s is indeed a vertex cover of size $\leq k$ for G
Travelling salesman Problem	Determines if the given string s is indeed a tour of cost $\leq c$ in G
Hamiltonian cycle	Determines if the given string s is indeed a cycle in G
Independent Set	⋮
3D matching	
Integer Linear Programming	
⋮	

Convince yourself that these certifiers satisfy

the *redefined* behavior of efficient certifiers described in the previous slide.

NP class

Definition (**NP**):

The set of all <u>decision</u> problems

NP : “Non-deterministic polynomial time”

Definition (**P**):

The set of all decision problems

Any Relation between **P** and **NP** : ?

P is contained in **NP**

X : any decision problem in **P**

Yes instance

I : any (input) instance of X

No instance

Let Q be the polynomial time algorithm for <u>solving</u> X.

Efficient certifier for X :

A polynomial time algorithm A with output {yes,no}

- **Input** : (I, s)

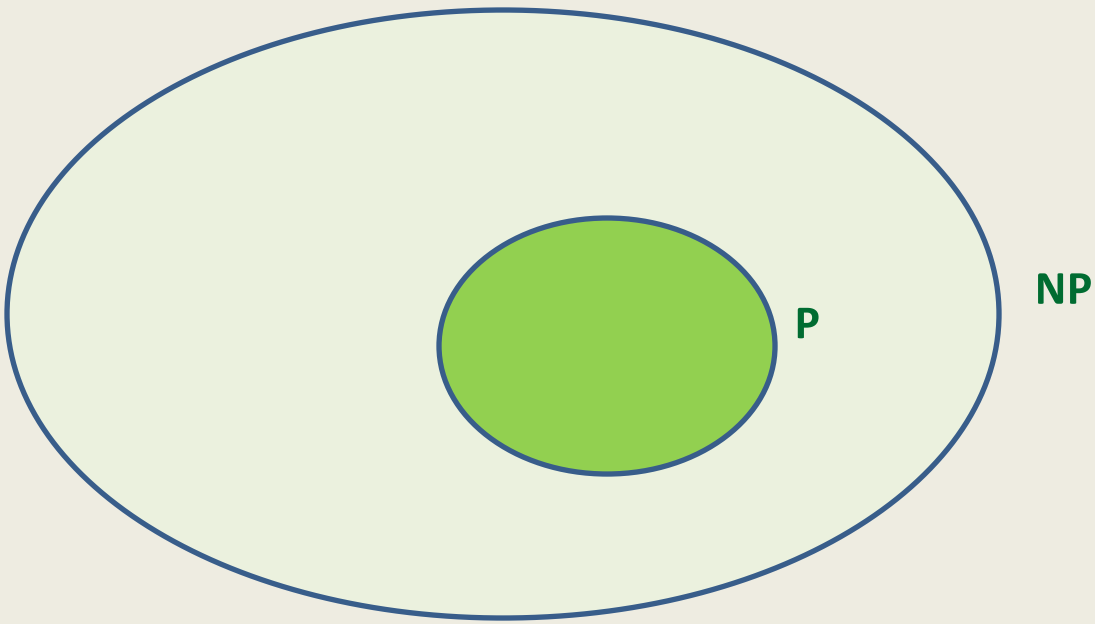
Proposed solution

- **Behavior**: On getting input (I, s),
  just ignore s,
  execute the algorithm Q on input I.
  If the answer is yes, output yes; if the answer is no, output no.

Convince yourself that this certifiers satisfy the *redefined* behavior of efficient certifiers.

NP versus P

Is P = NP ?



Verifying a <u>proposed solution</u> versus **finding a <u>solution</u>**

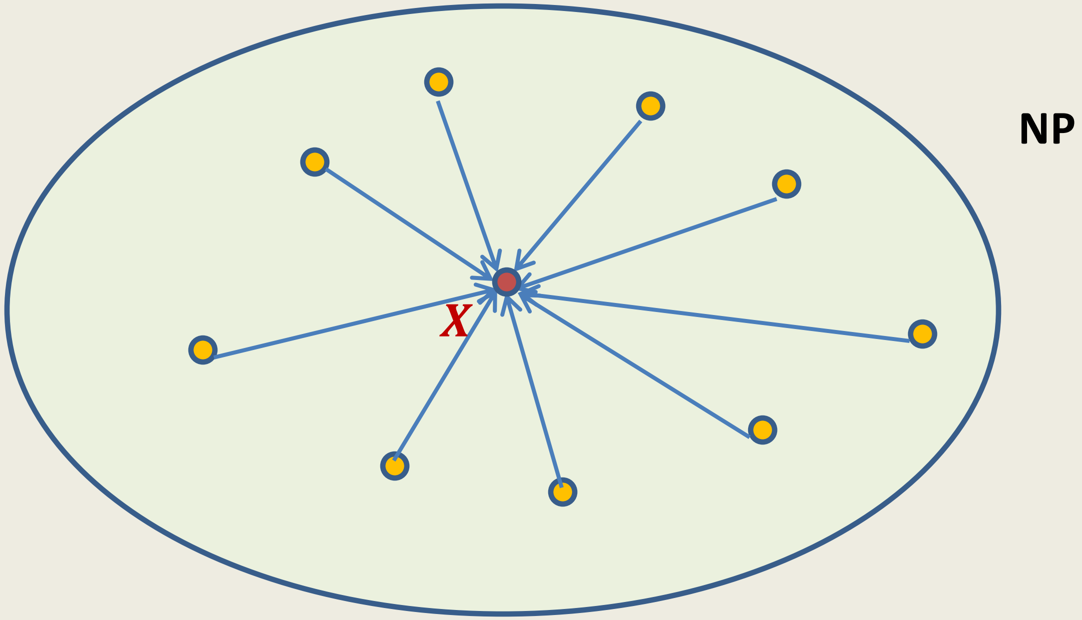
NP COMPLETE
A CLASS OF PROBLEMS

and how it came into existence

NP-complete

- A problem X in **NP** class is **NP-complete**

$$A \leq_P X$$



Does any NP-complete problem exist ?

It really needs

- courage to ask such a question and
- great insight to pursue its answer ?

Because:

- Every problem, known as well as unknown, from class **NP** has be reducible to this problem.
- Such a problem would indeed be the hardest of all problems in **NP**.

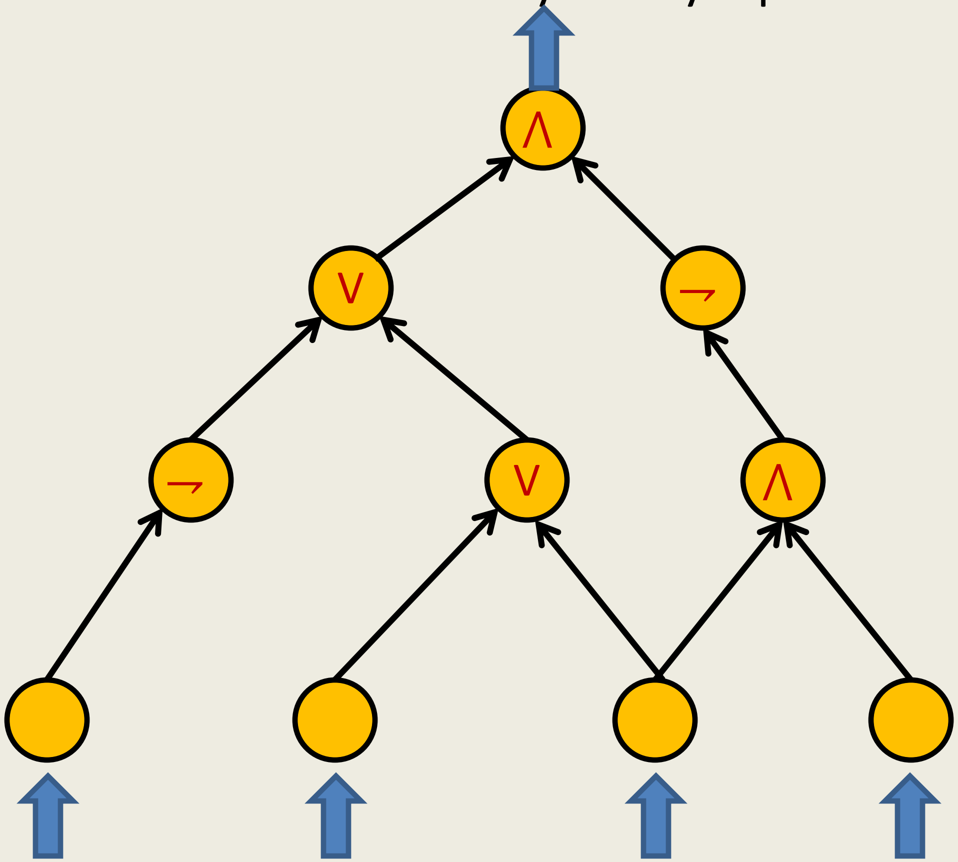
But only such great questions in science lead to great inventions.

Does any NP-complete problem exist ?

Circuit satisfiability problem: [Cook and Levin , 1971]

A DAG with nodes corresponding to **AND**,**NOT**,**OR** gates and n binary inputs,

does there exist any binary input which gives output 1 ?



This slide is optional.
(meant for the student whose aim is beyond just a good grade)

Question:

How can every problem from NP be reduced to circuit satisfiability ?

Answer:

Consider any problem $\boldsymbol{X}$ ∈ **NP**.

What we know is that it has an efficient certifier, say $\boldsymbol{Q}$.

Any algorithm which outputs yes/no can be represented as a DAG

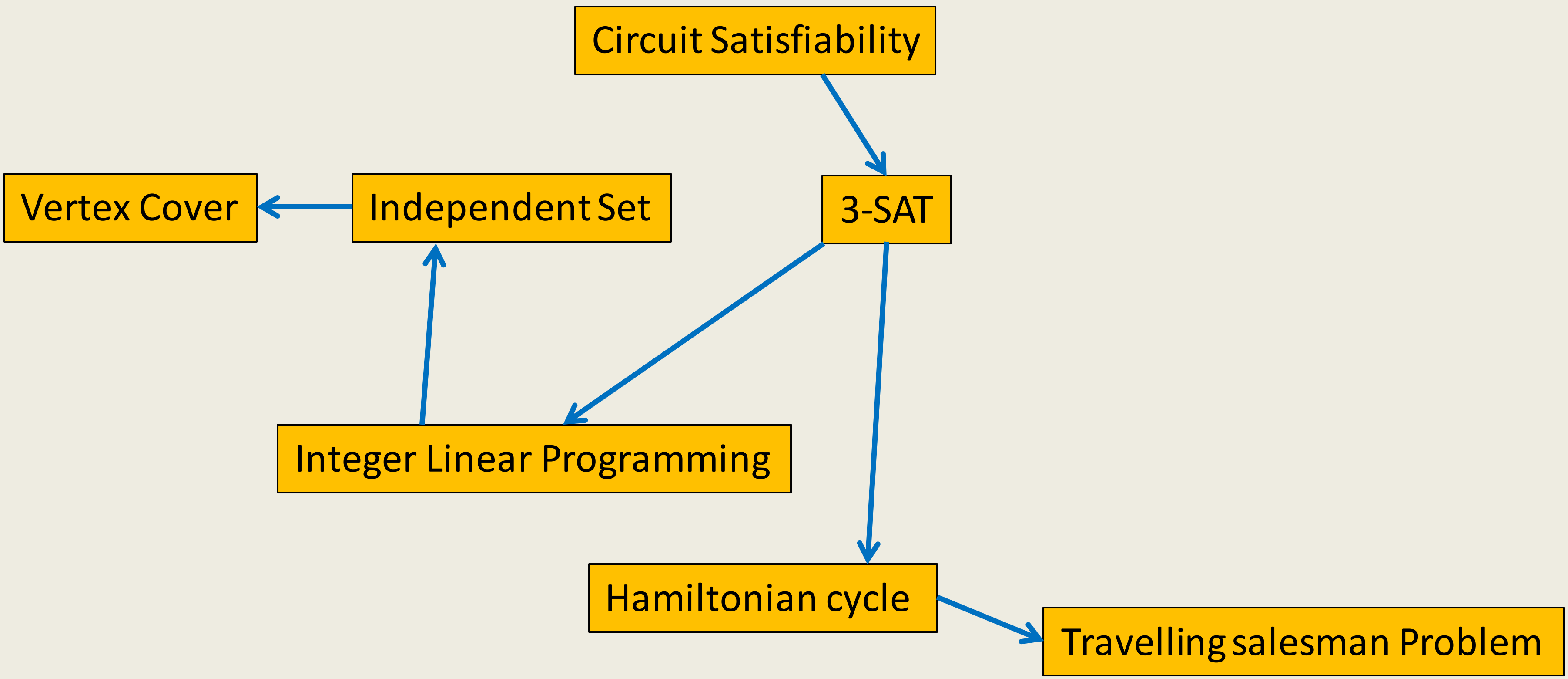
- Where internal nodes are gates.
- Leaves are binary inputs
- Output is 1/0.

So the Cook & Levin essentially transform $\boldsymbol{Q}$ into the corresponding DAG, and then simulates $\boldsymbol{Q}$ on the proposed solution.

[This is just a sketch. Interested students should study it sometime in future.]

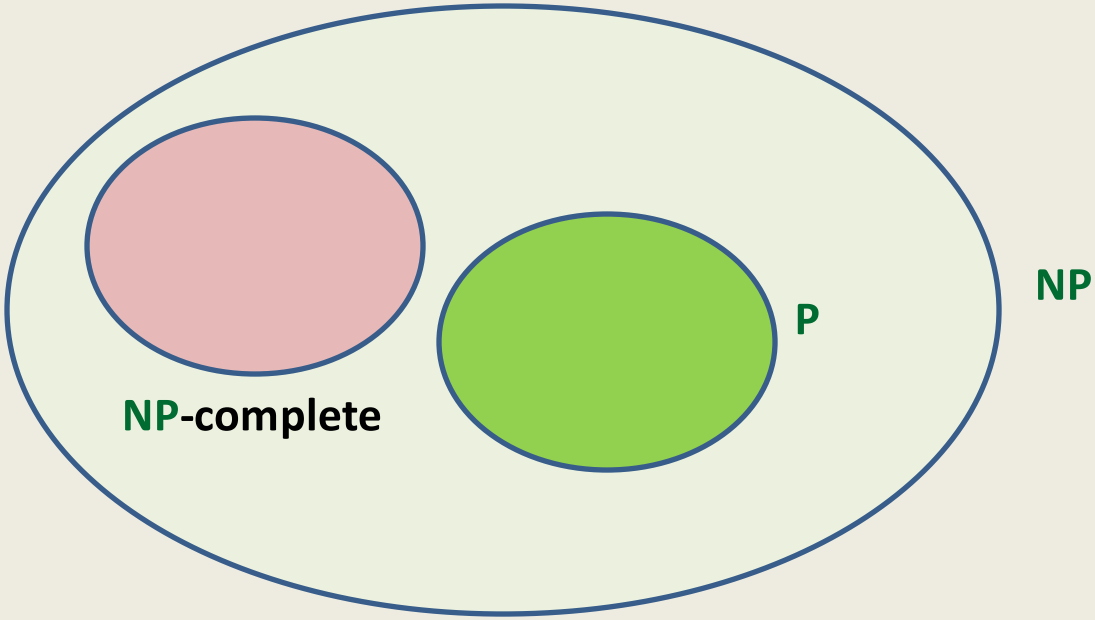
How many NP-complete problems exist ?

Polynomial reduction $A \leq_P X$ [Richard Karp, 1972]



NP versus P

Is P = NP ?



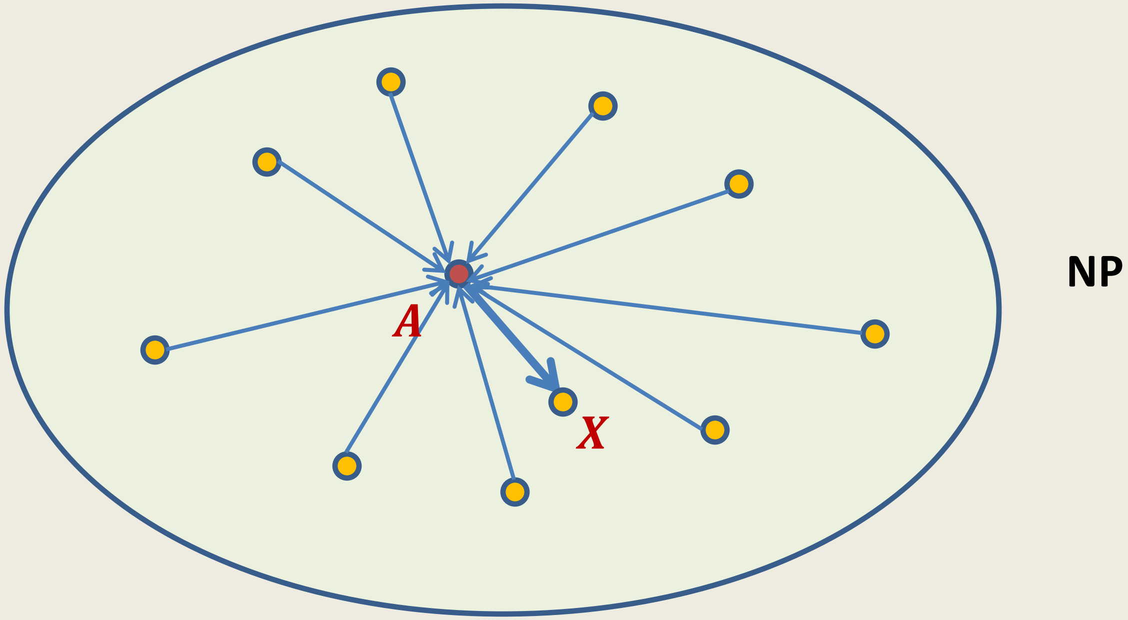
If any NP-complete problem is solved in polynomial time

➔ P = NP

How to show a problem to be NP-complete ?

Let X be a problem

1. Show that $X \in$ **NP**
2. Pick a problem A
3. Show that $A \leq_P X$



EXAMPLE

Showing Dominating Set to be NP-complete

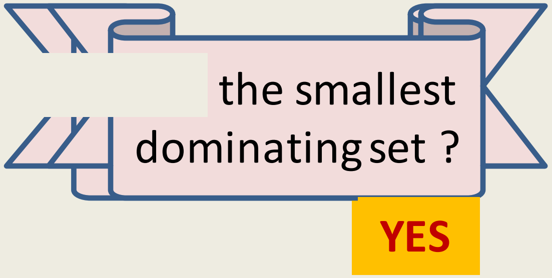
Dominating Set

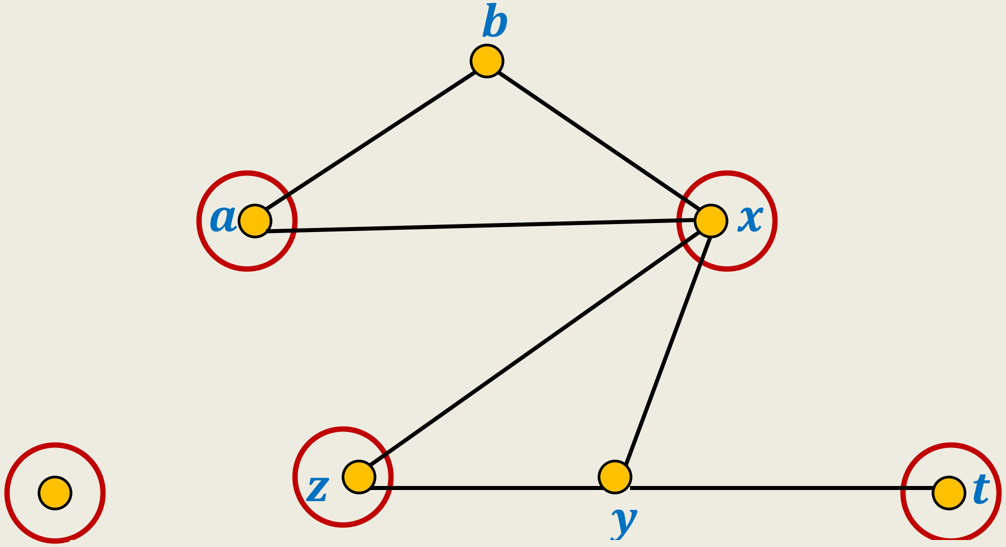
Definition: Given an undirected graph $G = (V, E)$,

$N(u)$=

a subset $X \subseteq V$ is said to be an dominating set if

For each $u \in V$,





Optimization version:

Decision version:

Dominating Set

Definition: Given an undirected graph $G = (V, E)$,

$$N(u) = \{v \mid v = u \text{ or } (u, v) \in E\}$$

a subset $X \subseteq V$ is said to be an dominating set if

For each $u \in V$, $N(u) \cap X \neq \emptyset$

Decision version:

Does there exist a dominating set of size k ?

Efficient Certifier:

Input: (G, X), $X \subseteq V$

Behavior:

It checks for each $u \in V$,

This algorithm takes **O**$(m + n)$ time.

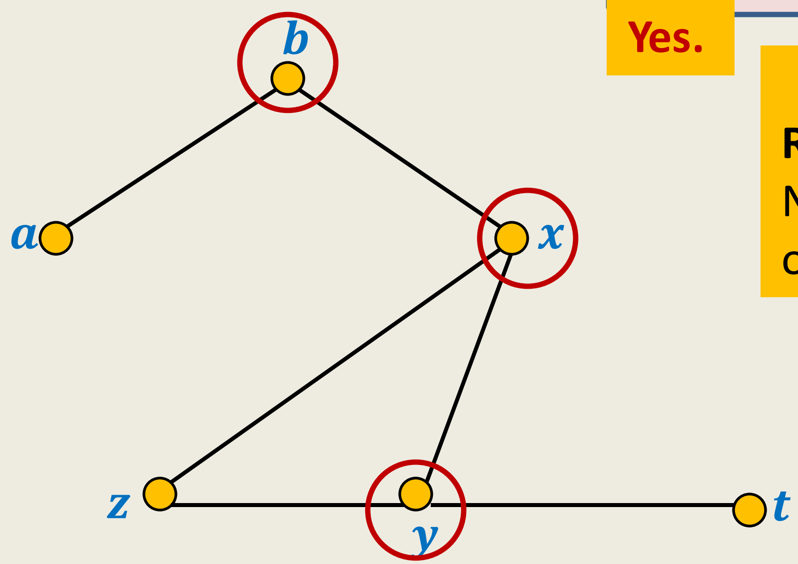
Vertex Cover

Definition: Given an undirected graph $G = (V, E)$,
a subset $X \subseteq V$ is said to be a **vertex cover** if
For each edge $(u, v) \in E$,
either $u \in X$ or $v \in X$

Is it a vertex cover now ?

Yes.

NO.
Reason:
None of (y, z) or (y, t) is covered



Optimization version: compute vertex cover of <u>smallest</u> size.

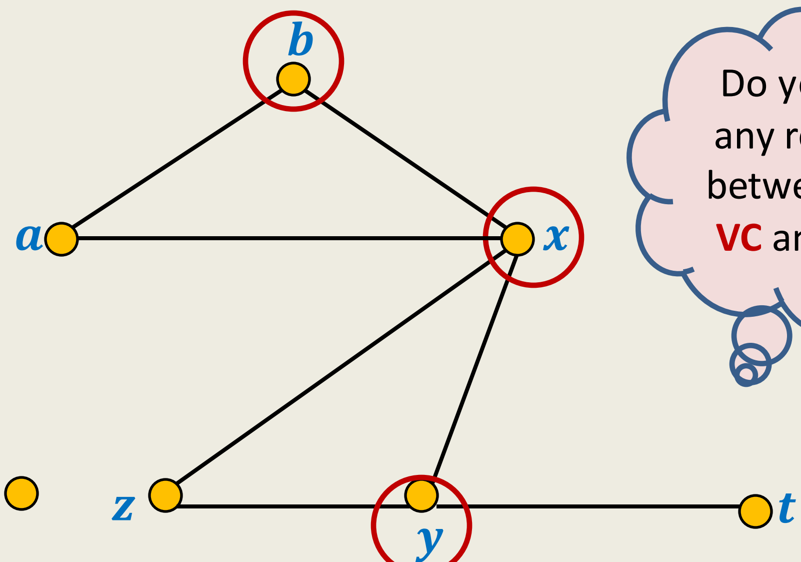
Decision version: Does there exist a vertex cover of size k ?

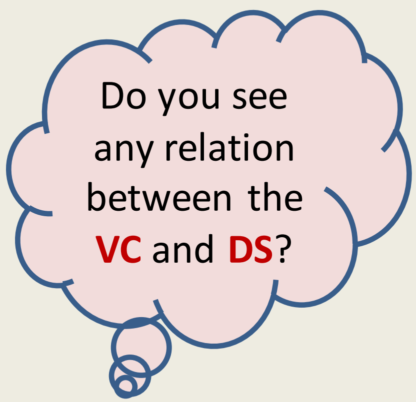
VC $\leq_P$ DS

VC: Vertex Cover

Input: an graph $G = (V, E)$ and $k \in Z^+$

Problem: Does there exist a vertex cover of size k?

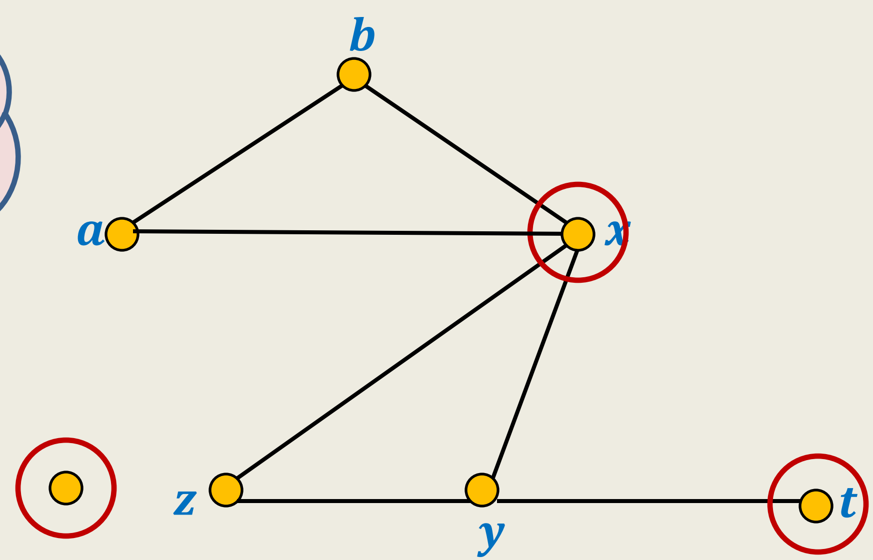




DS: Dominating Set

Input: an graph $G = (V, E)$ and $t \in Z^+$

Problem: Does there exist an dominating set of size t ?

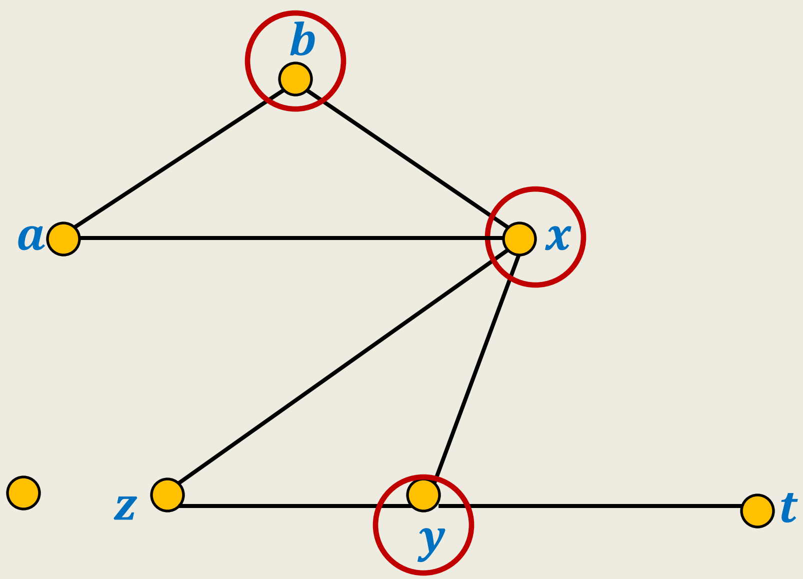


VC $\leq_P$ DS

VC: Vertex Cover

Input: an graph $G = (V, E)$ and $k \in Z^+$

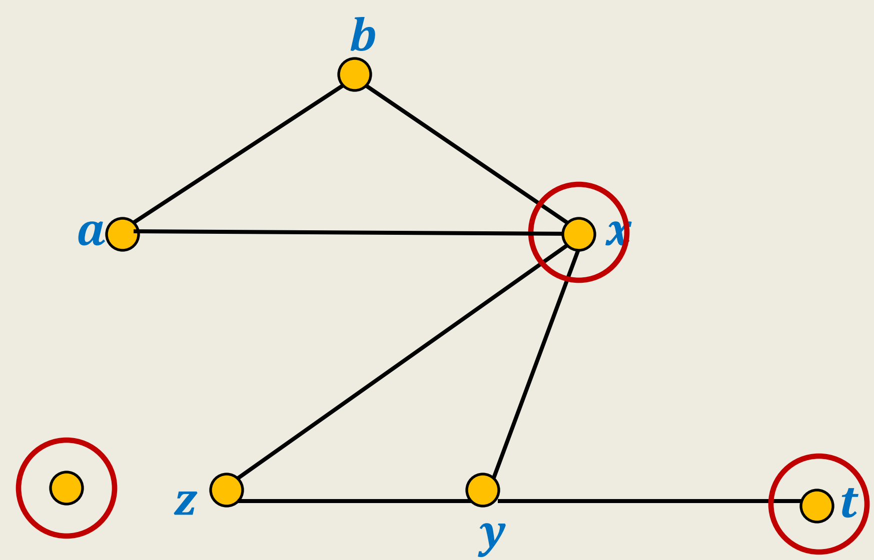
Problem: Does there exist a vertex cover of size k?



DS: Dominating Set

Input: an graph $G = (V, E)$ and $t \in Z^+$

Problem: Does there exist an dominating set of size t ?



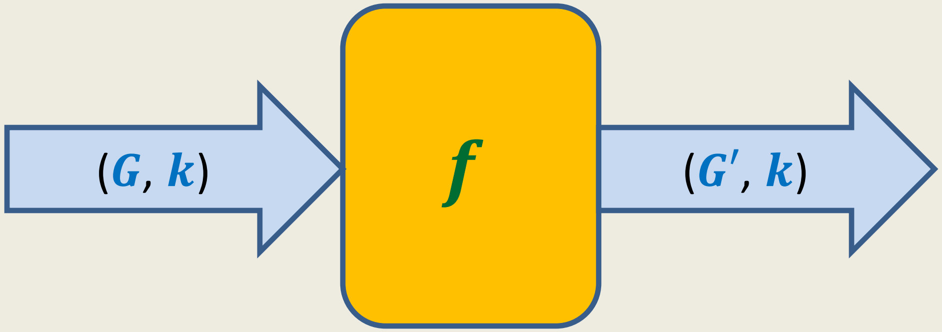
VC $\leq_P$ DS

Observation 1: Let $X \subseteq V$ be vertex cover of G. X is also a dominating set for G provided there are no isolated vertex in G.

➔

So without loss of generality assume G does not have any isolated vertex.

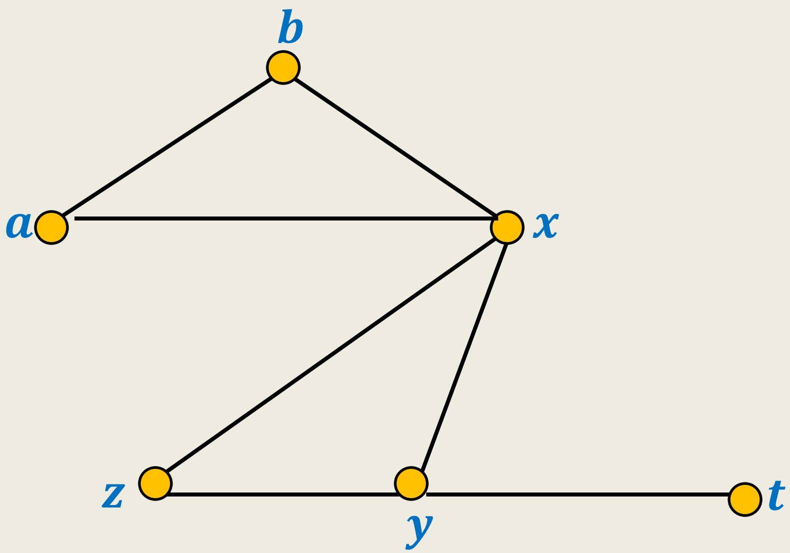
VC $\leq_P$ DS



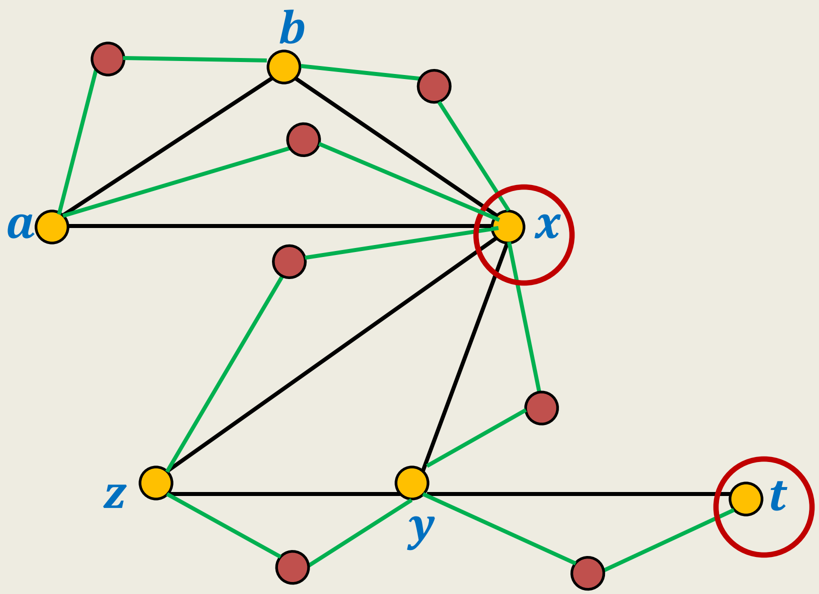
Aim: Given a graph $G = (V, E)$, how should f transform it to graph G' such that G has a vertex cover of size $\leq k$ <u>if and only if</u> G' has a dominating set of size $\leq k$.

VC $\leq_P$ DS

VC: Instance (G, k)

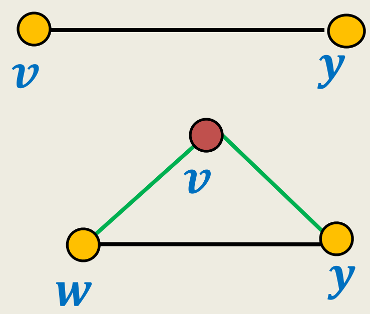


DS: Instance (G', k)



Theorem :

VC $\leq_P$ DS



Theorem (➔):

If G has a vertex cover of size $\leq k$

then G' has a dominating set of size $\leq k$.

Proof:

Let X be a vertex cover of G of size $\leq k$.

Consider any vertex $v \in V'$.

Case 1: $v \in V$

➔ v is dominated (use **Observation 1**)

Case 2: $v \in V' \setminus V$

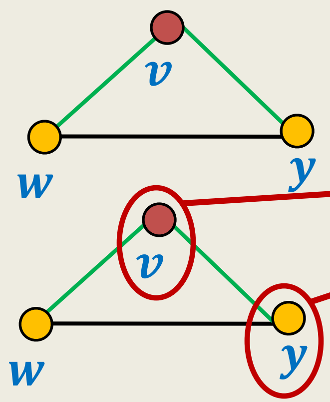
$(w, y) \in E$ and the edge is covered by X,

so either $w \in X$ or $y \in X$.

➔ v is dominated in this case as well ☺

VC $\leq_P$ DS

Replace v by y.
If y is already in X,
then just remove v.



Theorem (🡐):

If G' has a dominating set of size $\leq k$,

then G has a vertex cover of size $\leq k$.

Proof:

Let X be a dominating set of G'.

We can assume that $X \subseteq V$.

Now consider any edge $(w,y) \in E$.

Since v is dominated in X.

➔ $w \in X$ or $y \in X$.

We are done.

Lemma:

If G' has a dominating set of size $\leq k$,

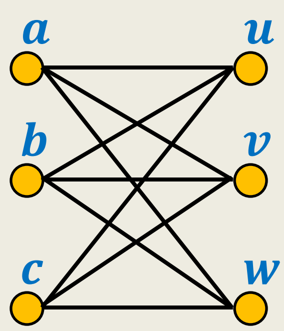
then G' also has a dominating set $X \subseteq V$

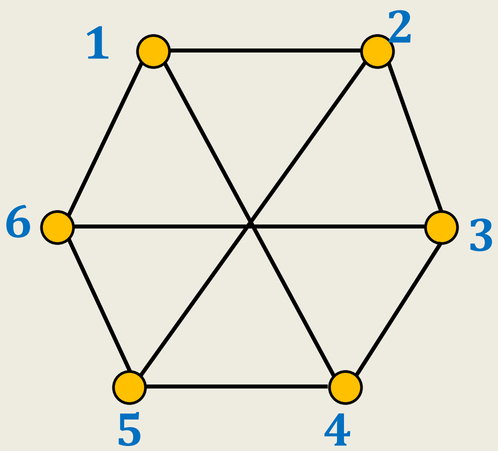
of size $\leq k$.

MORE NP-COMPLETE PROBLEMS

Subgraph Isomorphism

$a \rightarrow 1$
$b \rightarrow 3$
$c \rightarrow 5$
$u \rightarrow 2$
$v \rightarrow 4$
$w \rightarrow 6$





Definition:

if there is a **bijection** $f : V \rightarrow V'$

$$(u, v) \in E \text{ if and only if } (f(u), f(v)) \in E'$$

Problem: Given two graphs G and G',

does there exist a **subgraph** of G

Homework: Show that subgraph isomorphism problem is NP-complete.

Subset sum problem

Problem:

Given a set $\boldsymbol{A}$ of $\boldsymbol{n}$ integers: $\boldsymbol{i_1}, \boldsymbol{i_2}, \ldots, \boldsymbol{i_n}$,

does there exist any <u>subset</u> $\boldsymbol{S} \subseteq \boldsymbol{A}$ such that

$$\sum_{j \in S} j = \frac{1}{2} \sum_{k \in A} k$$

Showing that **subset sum** problem is in **NP**: easy

Showing that **subset sum** problem is **NP-complete :**

neither **Homework**
nor **Exam problem**

HOW TO HANDLE NP-COMPLETE PROBLEMS

Approximation Algorithms

(Next lecture)