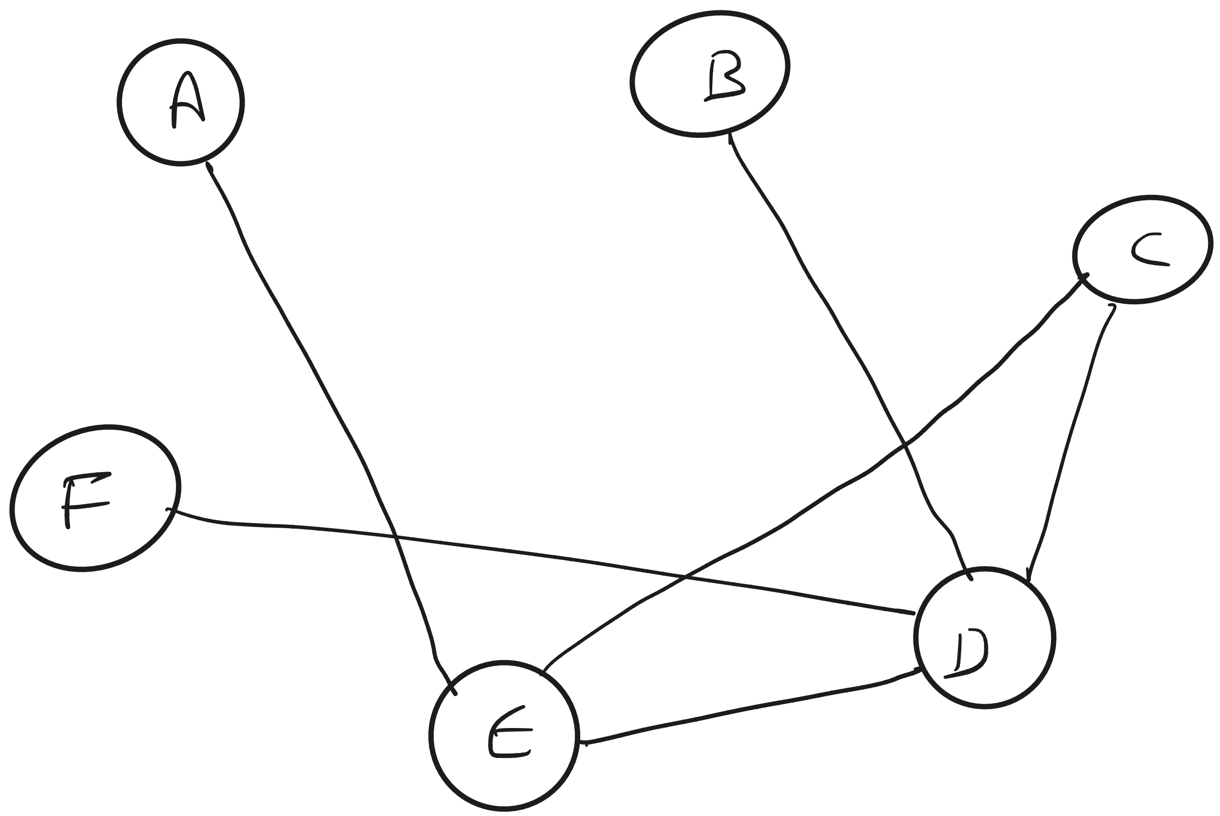
Lecture 21

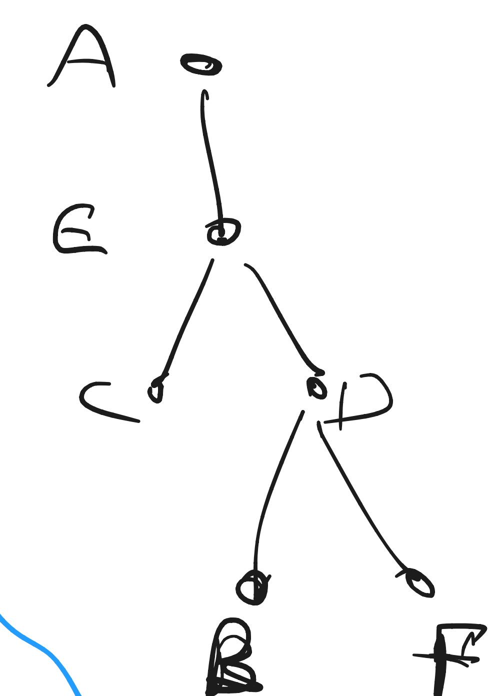
Today : NP, NP-completeness

Some Problems in P

① s-t connectivity : given graph G, and 2 vertices s, t does there exist a path from s to t?



If $(G, s=A, t=F)$:



Naive soln: try all possible paths $\sim n!$ paths

runtime $\sim n! > 2^n$

Better soln: $O(n+m)$ $n = \#$ vertices
$m = \#$ edges

Some Problems in P

decimal or

(2) Primes : given x in binary, is x prime?

Naive alg: try to divide x by $2, 3, 4, \dots, x-1$

Runtime : $\sim x$ steps

$= $ exponential in $|x|$

runtime of brute force algorithm is $O(x)$

Highly Nontrivial alg: Primes $\in P$

$O(x)$ is exponential in $|x|$

Some Problems in P

③ All Regular, CFL's are in P

④ graph connectivity:

given G, is there a path between every pair of vertices in G?

⑤ Linear Programming:

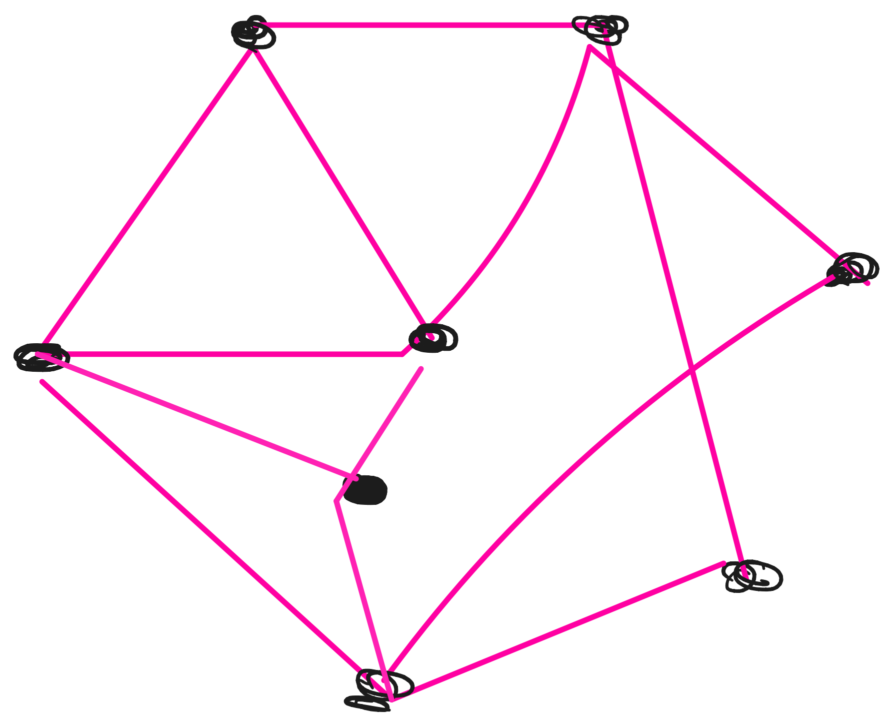
given linear set of constraints and linear objective function, Find optimal solution (over $\mathbb{R}$)

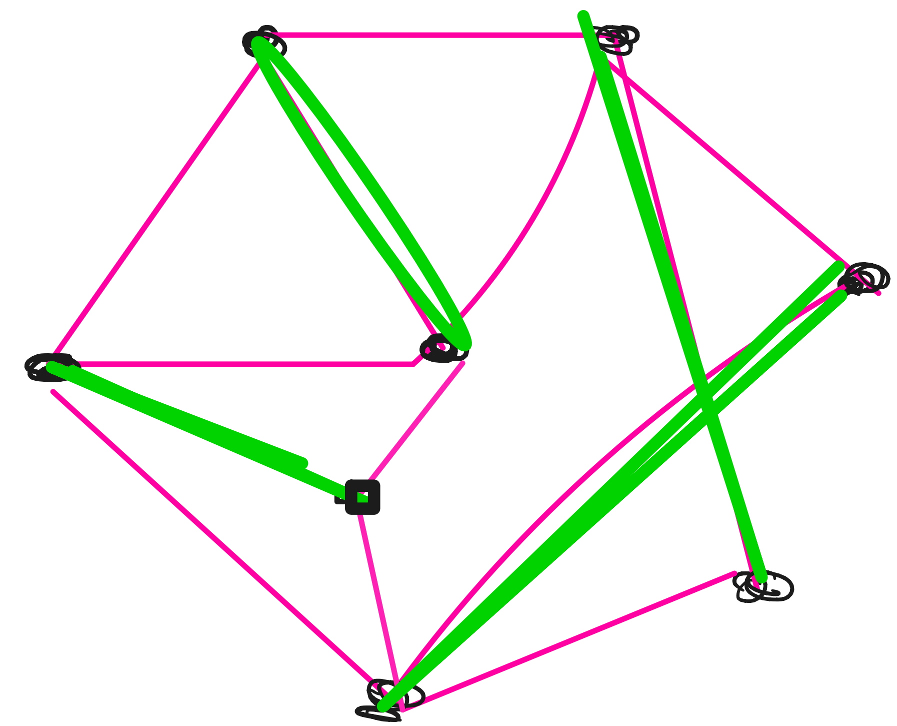
⑥ perfect matching: given G,

does $\exists$ a perfect matching in G?

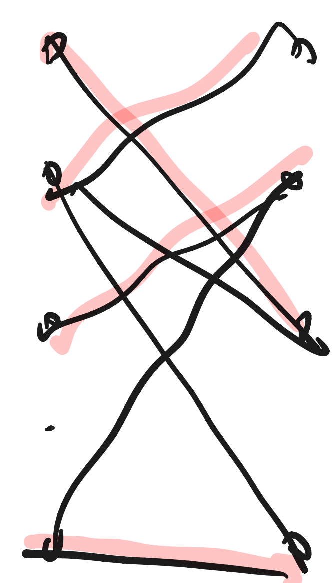
Perfect matching cont'd

G:

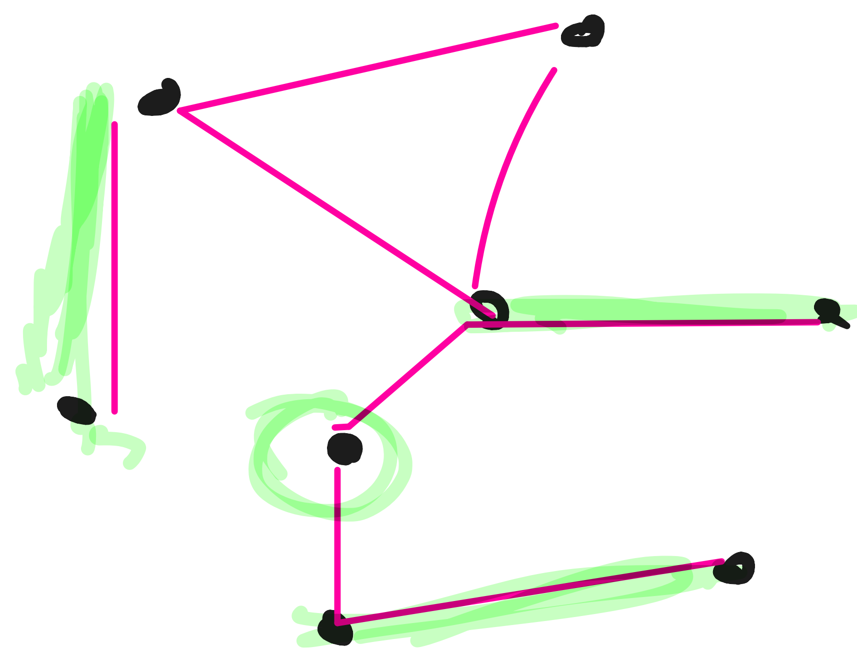




So G has a PM



Perfect matching cont'd



this G has no PM.

NP: When is it easy to Find a Needle in Haystack?

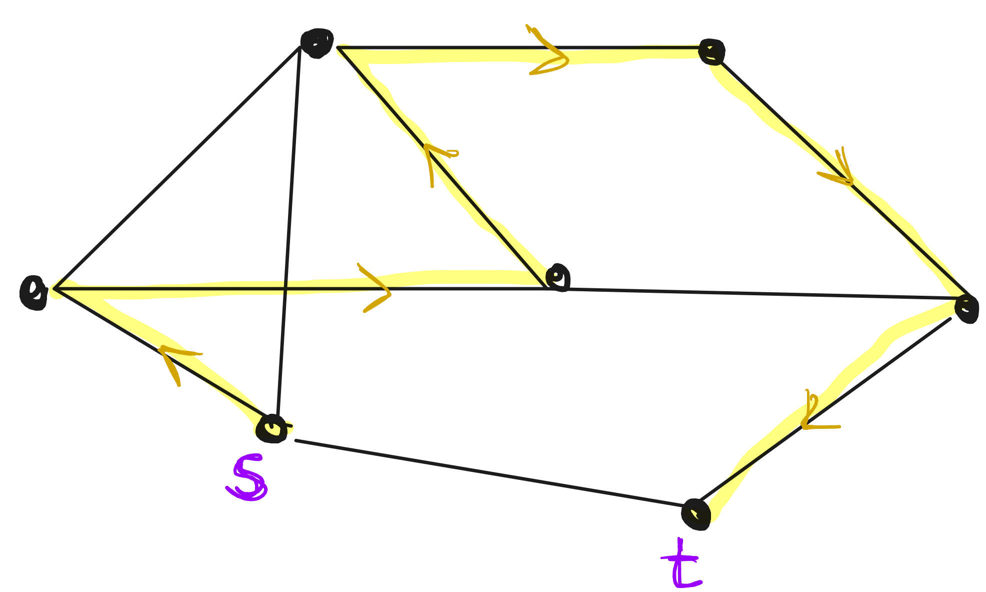
Many of the problems we are interested in are questions about searching for a solution in a huge (exponential sized) set of possible solutions.

Examples:

1. Does g have a clique of size $\frac{n}{2}$?

2. Hamiltonian Path HAMPATH(g, s, t)

g:



← A path from s to t that visits every vertex exactly once

NP: When is it easy to Find a Needle in Haystack?

Many of the problems we are interested in are questions about searching for a solution in a huge (exponential sized) set of possible solutions.

Examples:

1. Does G have a clique of size $\frac{n}{2}$?
2. Hamiltonian Path

In these examples it is always easy to verify a solution
But sometimes it is hard to find a solution

What characterizes NP is that it is always easy to verify a solution (out of $\sim 2^n$ potential solutions)

What characteriz

The (even more famous) class NP

Defn 1 NP = { L | L is a language decided by a Nondeterministic TM running in polynomial time }

polytime : runtime is $O(n^k)$ for some $k \geq 0$

The (even more Famous) class NP

Defn 1 NP = { L | L is a language decided by a Nondeterministic TM running in polynomial time }

Equivalent Defn of NP

A verifier for Language $L \subseteq \{0,1\}^*$ is an algorithm

$L = \{ w \mid V(w,c) \text{ accepts} \}$ where c is an additional string that we call a certificate or proof

A verifier is polynomial-time if it runs in time polynomial in $|w|$.

* Note that if A is a polytime verifier then $|c|$ must also be polynomial in $|w|$.

Defn 2 NP = { L | L has a polytime verifier }

Equivalence Between Defns 1 and 2

$\mathcal{L}_1 = \{ L \mid L$ is accepted by a nondeterministic polytime algorithm$\}$

$\mathcal{L}_2 = \{ L \mid L$ has a polytime verifier$\}$

① $L \in \mathcal{L}_2 \Rightarrow L \in \mathcal{L}_1$:

Let Algorithm A be verifier for L running in time n^k

Nondet TM N: on input w, $|w| = n$

Nondeterministically select c, $|c| \leq n^k$

Run V on (w, c)

If V accepts (w, c), accept, otherwise reject

Equivalence Between Defns 1 and 2

$\mathcal{L}_1 = \{ L \mid L$ is accepted by a nondeterministic polytime algorithm$\}$

$\mathcal{L}_2 = \{ L \mid L$ has a polytime verifier$\}$

(2) $L \in \mathcal{L}_1 \Rightarrow L \in \mathcal{L}_2$:

Let N be a nondeterministic TM accepting L and running in time n^k

Verifier A on $\langle w, c \rangle$:

Simulate N on w, where c is a description of the nondeterministic choices to make at each step

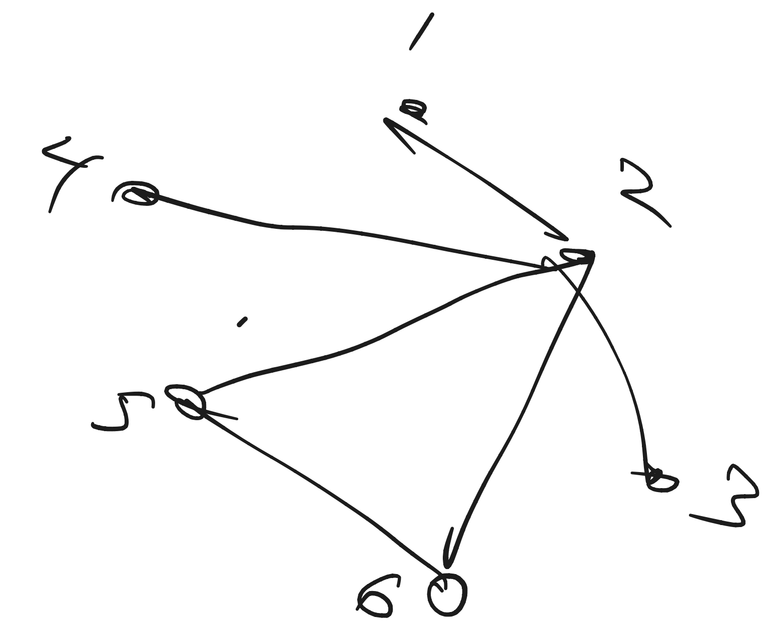
If this computation path (described by c) accepts then accept $\langle w, c \rangle$; otherwise reject

Examples of Languages in NP

① Any $L \in P$ is also in NP

Verifier V on input (w, c) : ignore c and just run polytime alg for L on input w.

② CLIQUE (g, k). $g = (V, E)$ $|V| = n$



Verifier V on input $(w = (g, k), c)$:

- check that c encodes a subset $V' \subseteq V$ of k vertices
- For all pairs of vertices $i, j \in V'$ check if (i, j) is an edge in E (i.e., $(i,j) \in E$)

$(g, k = 3)$

$n = 6$

c:

1	2	3	4	5	6
0	1	0	0	1	1

Another example.

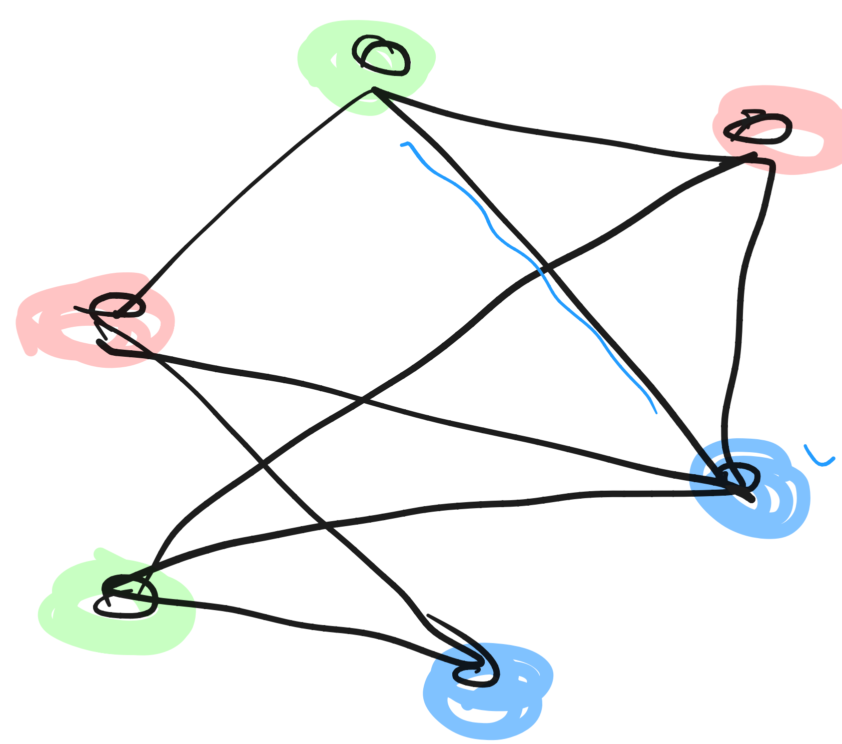
K-coloring Problem

Input (G, K)

accept iff G has a proper K-coloring

4-Color Problem:

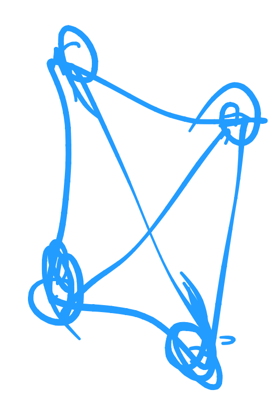
a graph can be 4-colored iff it is planar.



we know

if G has a K-clique

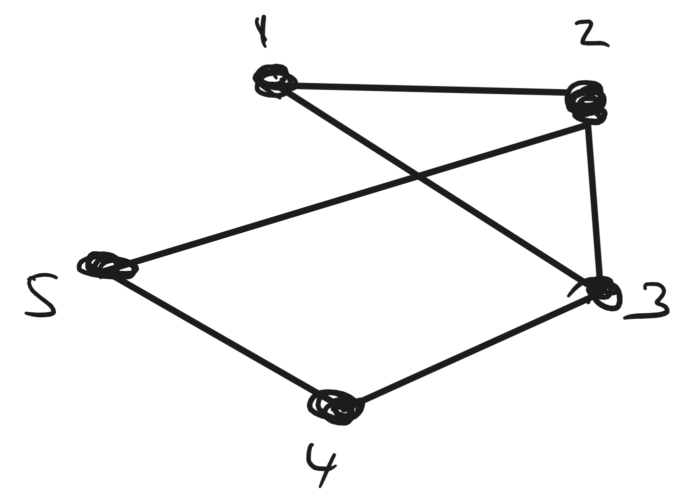
then it requires $\geq K$ colors



Note on encodings of a graph

$g = (V, E)$

$|V| = n$



$V = \{1, 2, 3, 4, 5\}$

2 standard encodings

1. Adjacency List : List all edges

$\{(1,2), (2,3), (1,3), (5,2), (5,4), (3,4)\}$

$m \cdot 2\log n \leq n^2 \cdot 2\log n$

\# edges in g

2. Adjacency Matrix

$|V| \times |V|$ matrix

	1	2	3	4	5
1	1	1	1	0	0
2	1	1	1	0	1
3	1	1	1	1	0
4	0	0	1	1	1
5	0	1	0	1	1

$M(i,j) = 1$ iff $(i,j) \in E$

n^2

Examples of Languages in NP

① Any $L \in P$ is also in NP

Verifier V on input (w, c) : ignore c and just run polytime alg for L on input w.

② CLIQUE $= \{ (g, k) \mid g$ is an undirected graph containing a size-k clique$\}$

Verifier V on input $((g, k), s)$:

- check that S encodes a subset $S \subseteq V$ of k vertices
- For all pairs of vertices $i, j \in V'$ check if (i, j) is an edge in E (i.e., $(i, j) \in E$)

③ K-SAT

Examples of Languages in NP

(3) $K\text{-SAT} = \{ \phi \mid \phi \text{ is a satisfiable } K\text{-CNF formula} \}$

Input is a Boolean formula over $x_1 \ldots x_n$ in kCNF form.

kCNF form: $C_1 \wedge C_2 \wedge \cdots \wedge C_m$

where each C_i is an OR of $\leq k$ literals

Example: $\phi = (x_1 \vee \bar{x}_2 \vee x_3) \wedge (x_3 \vee x_4) \wedge (\bar{x}_2 \vee \bar{x}_1) \wedge (\bar{x}_2 \vee x_3)$

ϕ is satisfiable if there is a 0/1 assignment $\alpha \in \{0,1\}^n$ the variables of ϕ such that $\phi(\alpha) = 1$

$x_1 = 0 \quad x_2 = 0 \quad x_3 = 1 \quad x_4 = 0$ satisfies ϕ

$$\underbrace{(x_1 \vee \bar{x}_1 \vee x_3)}_{\text{always satisfiable}}$$

$$(x_1 \vee x_1 \vee x_4) \equiv (x_1 \vee x_4)$$

Examples of Languages in NP

(3) $K\text{-SAT} = \{ \phi \mid \phi \text{ is a satisfiable } K\text{-CNF formula} \}$

Input is a Boolean formula over $x_1 \ldots x_n$ in KCNF form.

KCNF form: $C_1 \wedge C_2 \wedge \ldots \wedge C_m$

Example: $\phi = (x_1 \vee \bar{x}_2 \vee x_3) \wedge (x_3 \vee x_4) \wedge (\bar{x}_2 \vee \bar{x}_1) \wedge (\bar{x}_2 \vee x_3)$

ϕ is satisfiable if there is a 0/1 assignment $\alpha \in \{0,1\}^n$ to the variables of ϕ such that $\phi(\alpha) = 1$

Let $\alpha = x_1 = 1,\ x_2 = 0,\ x_3 = 1,\ x_4 = 1$. ϕ is satisfiable since $\phi(\alpha) = 1$

Example 2 $\quad \phi' = (x_1 \vee x_2)(\bar{x}_1 \vee x_2)(\bar{x}_3 \vee x_4 \vee \bar{x}_2)(x_3)(\bar{x}_4 \vee \bar{x}_2)$

$\qquad\qquad\qquad 0 \quad 0 \quad 0$

this is unsatisfiable. (check all 2^4 assignments)

$x_3 = 1 \quad x_2 = 1 \quad x_4 = 0$

Examples of Languages in NP

(3) $K\text{-SAT} = \{ \phi \mid \phi$ is a satisfiable K-CNF formula $\}$

Input is a Boolean formula over $x_1, \ldots x_n$ in KCNF form.

KCNF form: $C_1 \wedge C_2 \wedge \cdots \wedge C_m$

Example: $\phi = (x_1 \vee \bar{x}_2 \vee x_3) \wedge (x_3 \vee x_4) \wedge (\bar{x}_2 \vee \bar{x}_1) \wedge (\bar{x}_2 \vee x_3)$

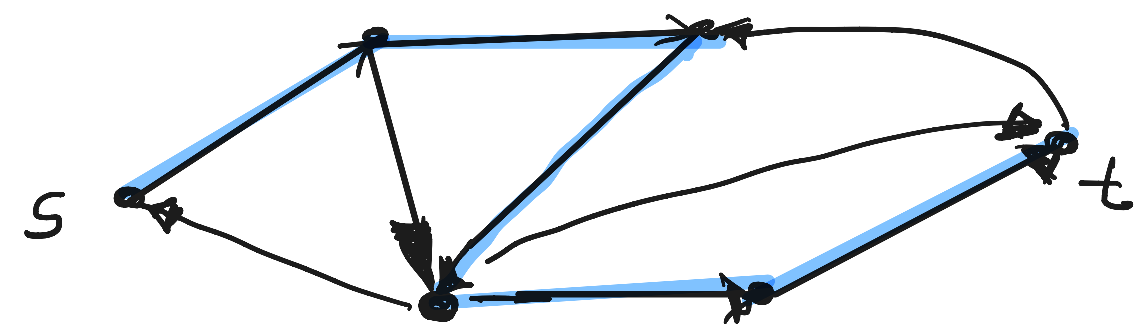
Verifier V on input (ϕ, α):

check that α is a Boolean satisfying assignment for ϕ. If yes $\rightarrow$ accept, otherwise $\rightarrow$ reject

Examples of Languages in NP

④ HAMPATH = $\{(G, s, t) \mid G$ is a directed graph containing a Hamilton path (visits all vertices once) from s to $t\}$

Example



Verifier V on input $((G, s, t), p)$:

Check if p encodes a Hamiltonian path from s to t.
If yes → accept; otherwise → reject

The Ubiquity of NP

→ It turns out there are thousands of problems in NP!

→ Many NP problems are fundamental in their respective areas of study

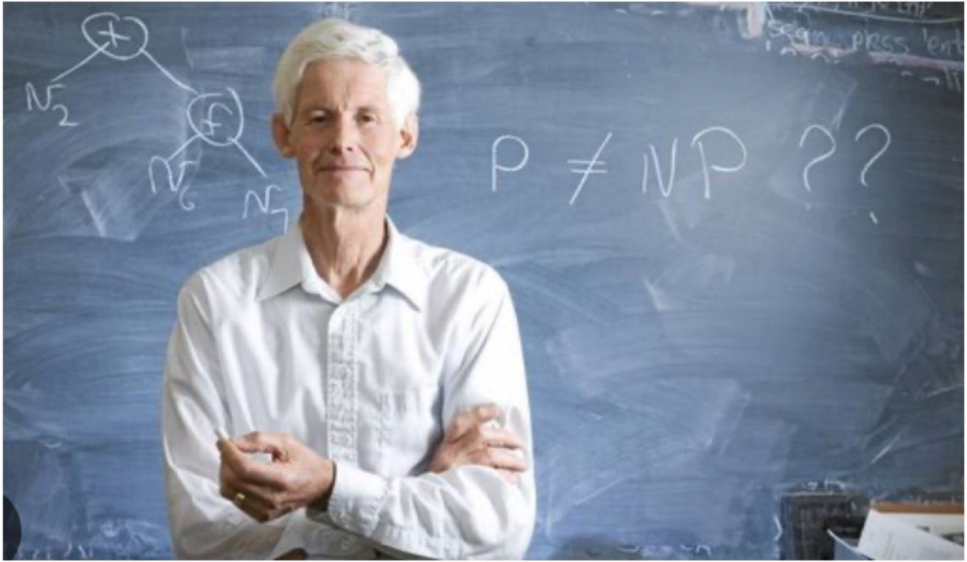
→ BIG QUESTION: $P \stackrel{?}{=} NP$


P=NP
$e^{\pi i} = -1$
1,4

The $1M question

The Clay Mathematics Institute
Millennium Prize Problems

1. Birch and Swinnerton-Dyer Conjecture
2. Hodge Conjecture
3. Navier-Stokes Equations
4. P vs NP
5. Poincaré Conjecture
6. Riemann Hypothesis
7. Yang-Mills Theory


P ≠ NP ??

NP - completeness

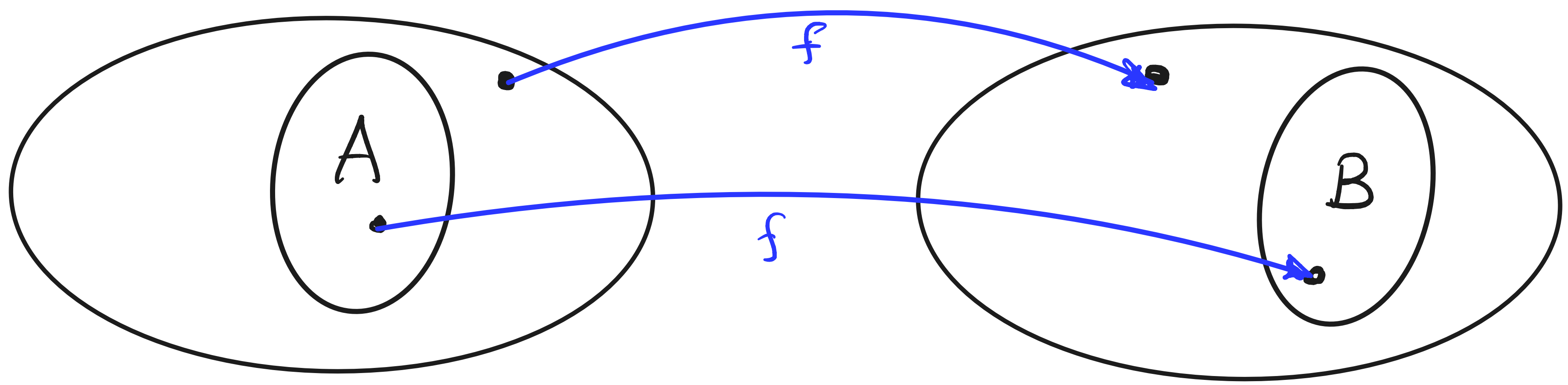
Cook (my advisor) and independently Levin established in 1970's that certain problems in NP (called NP-complete languages) whose individual complexity as the entire class of all NP problems!

For Example 3-SAT is NP-complete which implies that if there is a polytime algorithm for 3-SAT then <u>all</u> languages in NP are in P.

To formalize NP-completeness we need the notion of a polynomial-time reduction. This is just like the reductions we defined in previous section on computability, but now we require that the reduction is polynomial-time computable.

NP-completeness

Definition Language A is polynomial-time (mapping) reducible to B (written $A \le_p B$) if there is a polynomial-time computable function $f : \Sigma^* \to \Sigma^*$ such that $w \in A \iff f(w) \in B$

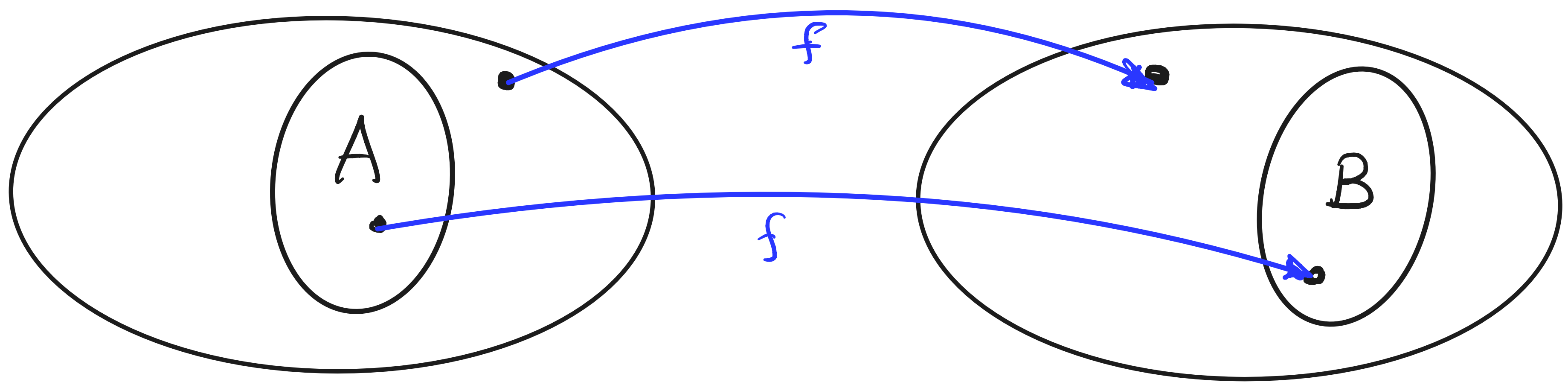


Definition

- A language $B \subseteq \{0,1\}^*$ is NP-hard if for <u>every</u> $A \in NP$ there is a polynomial time reduction from A to B $(A \le_p B)$

NP - completeness

Definition Language A is polynomial-time (mapping) reducible to B (written $A \le_p B$) if there is a polynomial-time computable function $f : \Sigma^* \to \Sigma^*$ such that $w \in A \iff f(w) \in B$



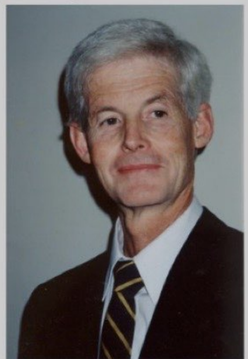
Definition

- A language $B \subseteq \{0,1\}^*$ is NP-hard if for every $A \in NP$ there is a polynomial time reduction from A to B ($A \le_p B$)

- $B \subseteq \{0,1\}^*$ is NP-complete if: (i) B is in NP and
  (ii) B is NP-hard

Theorem (Cook-Levin) 3-SAT IS NP-COMPLETE

The Cook/Levin theorem was independently proved by Stephen Cook and Leonid Levin



- Denied tenure at Berkeley (1970)
- Invented NP completeness (1971)
- Won Turing Award (1982)

- Student of Andrei Kolmogorov
- Seminal paper obscured by Russian, style, and Cold War

- We will prove Cook-Levin Theorem Next week.
- We currently cannot show $P \neq NP$, and therefore we don't know if 3-SAT is in P or not.
- Best evidence that a problem in NP is computationally infeasible (not in P) is by showing it is NP-complete.
- Next: Prove other Languages are NP-complete via <u>reductions</u>.

(Analogous to: Proving other Languages not decidable, once we have one undecidable language)