Lecture 20

Computability Wrapup

1. TMs : general model of computation (Church-Turing Thesis)

   Stronger versions: Multi-tape, Nondeterministic

2. Decidable Languages

   Examples: all Regular Languages,

   Recognizable Languages

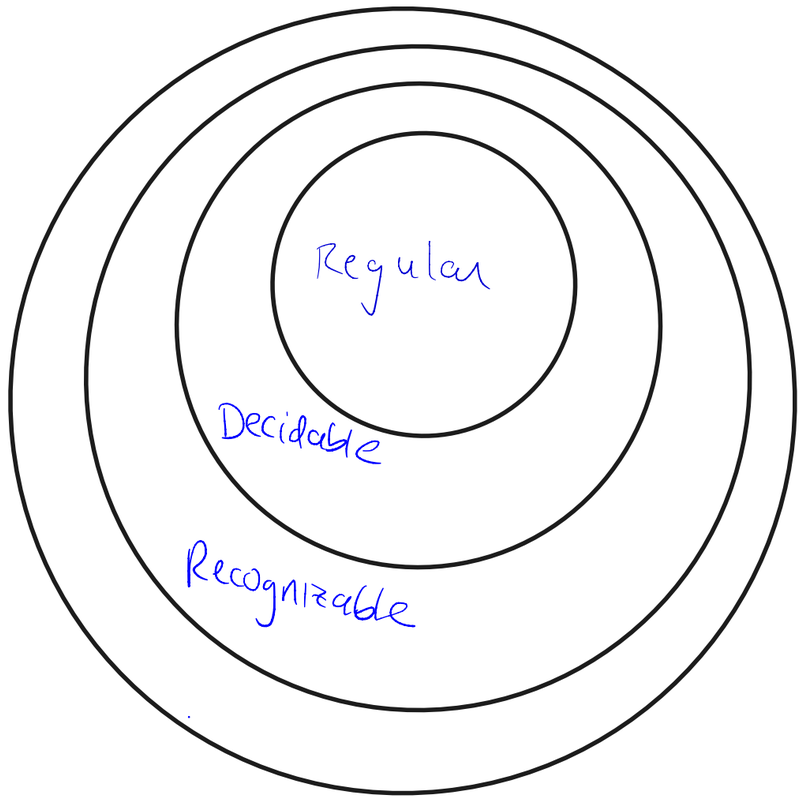
   Examples: All decidable languages HALT, A_{TM}

3. Closure properties of decidable/recognizable L's

4. Undecidable/unrecognizable Languages

   Method of Diagonalization

   Reductions



The Languages we showed are undecidable were all about properties of TMs. What about other more natural functions?

Here is a sample of some other (famous) undecidable problems:

1. Same questions (HALT, A_{TM}) are also undecidable in any other model of computation, e.g. Python programs, quantum computers, etc.

2. Hilbert's Tenth problem is undecidable (1900)

   Input: a diophantine equation (polynomial equation with integer coeff's)

   Example: $3x^2 - 2xy - z^3 + 5x^2y^2 = 0$

   Output: a soln over integers, or "unsolvable"

3. Undecidability of First order Logic (Hilbert's Entscheidungsproblem)

④ Data compression

given a string $s \in \{0,1\}^*$, find shortest program that outputs s

⑤ group theory

given a (finitely presented) group g

Is g finite?
Is g simple?
Is g commutative

} all undecidable

⑥ Physics - spectral gap (2015)

(difference between ground state + first excited state)

Sci. American Oct 2018)

Many subsequent undecidable problems in quantum physics

Complexity Theory

We saw that certain languages are undecidable --
even with unbounded resources (time, memory)
we cant solve these problems in the worst case

But even if a problem is decidable it may take
an enormous amount of time/memory, so it
still may not be solvable in practice

Complexity Theory : the study of important/central
problems and the amount of resources required
to solve them.

time, space, randomness, parallel computation,
quantum computer

Some Examples

(1) Matrix Multiplication: given 2 nxn matrices M_1, M_2
How much time (elementary plus/times operations) to output $M_1 \cdot M_2$?

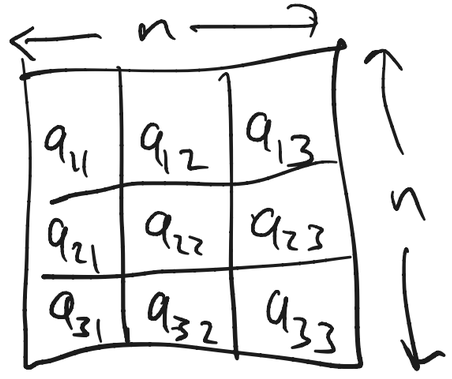
(2) Prime: given a number x in binary, is it prime?

(3) Factoring: given x in binary, output prime factorization

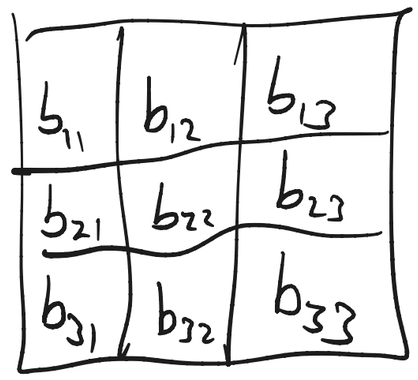
(4) Clique: given a graph G on n vertices, and a number k,
does G contain a clique of size k?

(5) Sudoku: input nxn puzzle, output a solution

Example: $n = 3$ $\quad M_1 =$



$M_2 =$



Entry (i,j) of $M_1 \times M_2$:

$$= a_{i1}b_{1i} + a_{i2}b_{2i} + a_{i3}b_{3i}$$ } $O(n)$ operations

Runtime of the obvous alg $\approx \underbrace{n^2}_{n^2 \text{ entries}} \cdot \underbrace{n}_{O(n) \text{ operations each}}$

Q. Is there an alg running in n^2 time?

Best known $\approx n^{2.3}$ time

Primality

Say $x = \underbrace{23}$ decimal
binary

There is a randomized alg. that runs very fast

Open for a long time: is there a fast deterministic alg (runtime n^2, n = length of x)

↖ Yes!

Time Complexity

A step of a TM is a single transition of the TM on an input

The time complexity of a TM is a function (denoted $t(n)$ or $f(n)$) that measures the worst-case number of steps M takes (before halting) on any input w of length n

Time complexity, Big-O notation

Big-O : ignore everything except the dominant growth term, including constant factors

Defn For any 2 functions $f(n)$, $g(n)$
$f(n) = O(g(n))$ if $\exists c, n_0$ s.t. $\forall n \geq n_0$ $f(n) \leq c \cdot g(n)$.

Examples

1. $4n + 10 = O(n)$ ⟵ $f(n) = 4n+10$ $g(n) = n$
   $f(n) = O(g(n))$
2. $n^2 + 3n + 4 = O(n^2)$
3. $2^n + n^3 = O(2^n)$
4. $n^2 \log n + \log\log n = O(n^2 \log n)$

Time Complexity, Big-O Notation

Why do we care about asymptotic (Big-O) growth?

- We want to estimate the runtime of an algorithm.
  However differences in hardware / implementation can lead to differences in runtime.

  Example: register size, caching, etc.

- Analyzing Runtime can be cumbersome — big-O hides a lot of unnecessary details

Example of how Big-O Makes things Easier

M on input w

Scan across tape until we see a 0 or 1 — O(n) steps

If none found → halt and accept — O(1) steps

If one found, continue scanning until a matching 0 or 1 found — O(n) steps

If none found reject — O(1) steps

OW cross off that symbol and repeat — O(n) steps

} O(n) loops

So total worst case runtime on w, $|w| = n$ is

$$\big(O(n) + O(1) + O(n) + O(1) + O(n)\big) \cdot O(n) = O(n) \cdot O(n) = O(n^2)$$

The Famous Class "P"

Defn Let $t : \mathbb{N} \to \mathbb{N}$ ($t \approx$ runtime)

$\text{TIME}(t(n)) = \{ L \mid L$ is a language decided by a $O(t(n))$-time TM $\}$

Defn $t : \mathbb{N} \to \mathbb{N}$ is polynomial if $t(n) = O(n^k)$ for some $k \geq 0$

Ex $t(n) = n^2$, $t(n) = n$, $t(n) = n \log n$ are polynomial

($t(n) = n^{\log n}$ or $t(n) = 2^n$ are not polynomial.)

Defn $P = \{ L \mid L$ is a language decided by a TM running in polynomial time $\}$

* We think of problems in P as those that have relatively efficient algorithms.

The Famous Class "P" : Discussion / Motivation

Q1: Why polynomial time? Why not linear time or quadratic time?

Valid point.
If some program runs in n^{1000} time that certainly isn't feasibly solvable

If some problem $\notin P$ then we can say for sure that it is infeasible to solve in the worst case

Typical polytime algs actually run in time n or n^2 or n^3
so placing a problem in P usually means it is hopeful that it can be solved fairly efficiently

Still, it is important after placing a problem in P, to find a truly fast (ie. $O(n)$ or $O(n \log n)$ time) algorithm

The Famous Class "P" : Discussion / Motivation

Q2: TMs are <u>so</u> slow. why don't we define "P" for a better model of computation

Also good point. Really we want to consider a more realistic model like multitape TMs, or random-access machines.

But the simulation of these by ordinary TMs is <u>polynomial</u> time, so if some problem has a polytime alg in some other model, it will usually also have a polytime TM algorithm

one big exception: quantum computers

The Famous Class "P" : Discussion / Motivation

Q3: Why worst-case runtime?

Another good point.
Just because a problem is hard on some inputs, this isn't the whole story.

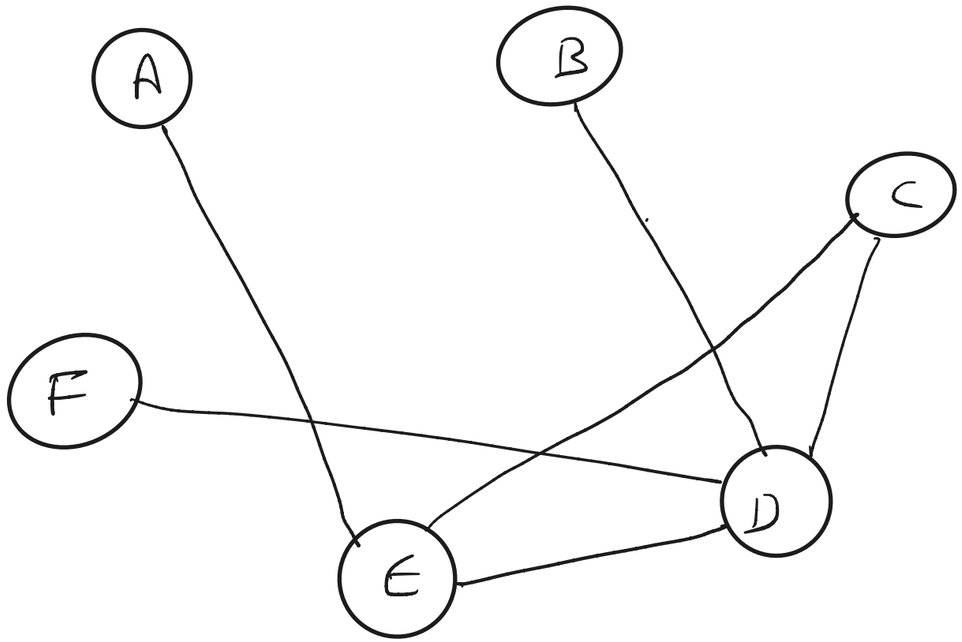
- It may be very easy to solve on 'typical' inputs
  Example: whole field of machine learning, chatgpt

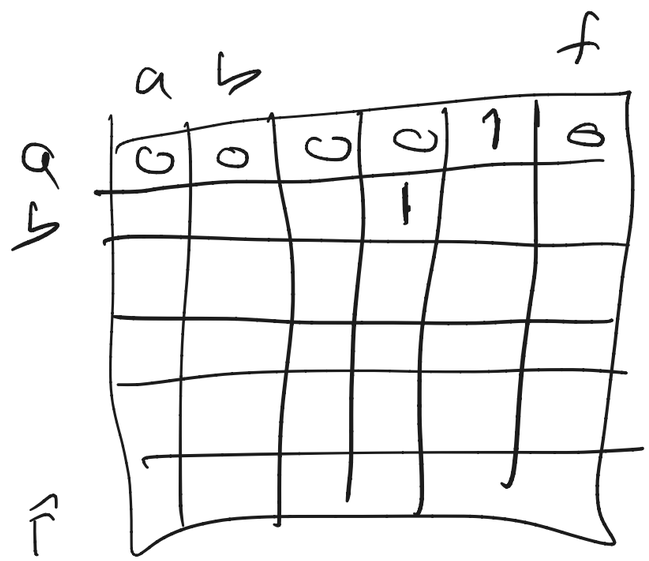
- It may be easy to get a very good approximation in polytime even if solving optimally is not in P

Again, understanding worst case complexity is a starting point
ideal: solve exactly in polytime. If not possible, see if efficient on avg, or easy to approximate.

Some Problems in P

① S-t connectivity : given graph G, find length of shortest path from A to F





Naive solv: try all possible paths ~ n! paths

runtime ~ $n! > 2^n$

Better solv: $O(n+m)$ n = # vertices, m = # edges

If m = # edges

$\# \underbrace{m \log n}_{} \leq n^2 \log n$

Some Problems in P

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

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

Runtime : $\sim x$ steps

= exponential in $|x|$

Highly Nontrivial alg : Primes $\in P$

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?

```
x=0 y=0
For i=1 .. n
    For j=1 .. m
        let x=j+5
        let y=x-7
```

$n \cdot m \cdot O(1)$

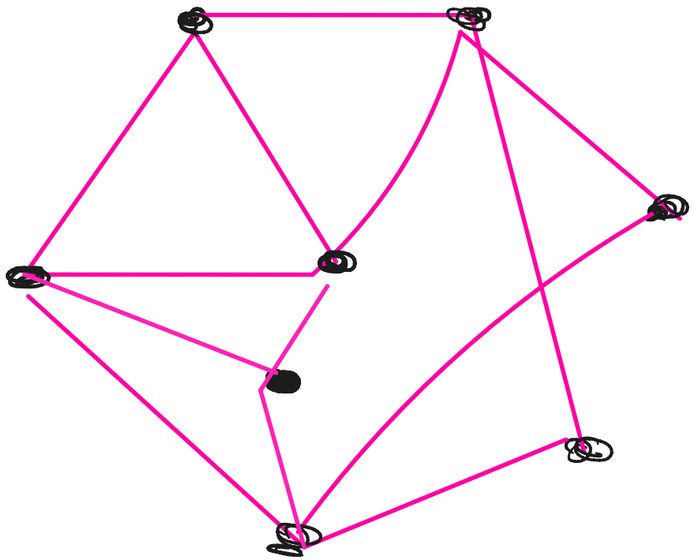
⑤ Linear Programming:
given linear set of constraints and
linear objective function, Find
optimal solution (over $\mathbb{R}$)

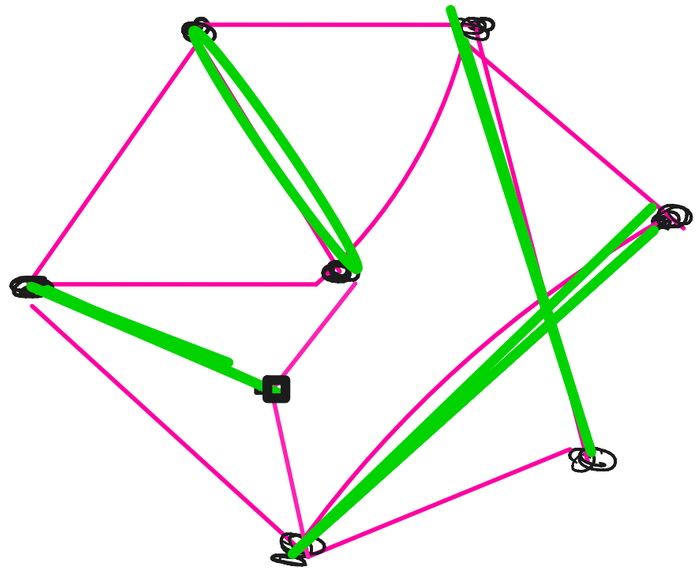
$O(n \cdot m)$

⑥ perfect matching: given g,
does $\exists$ a perfect matching in g?

Perfect matching cont'd

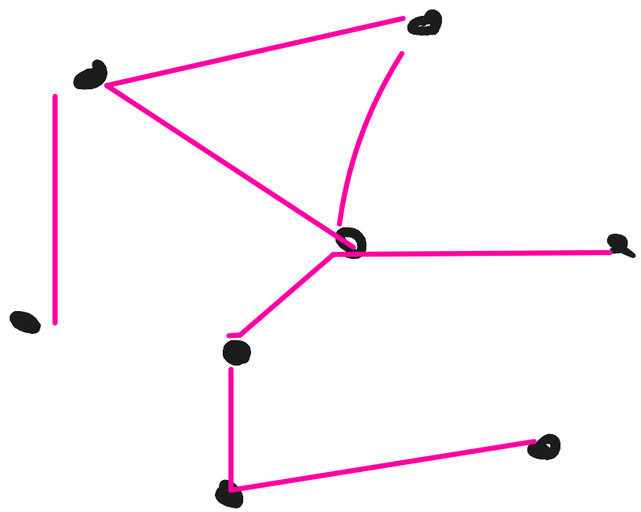
g:





So g has a PM

Perfect matching cont'd



this G has no PM.

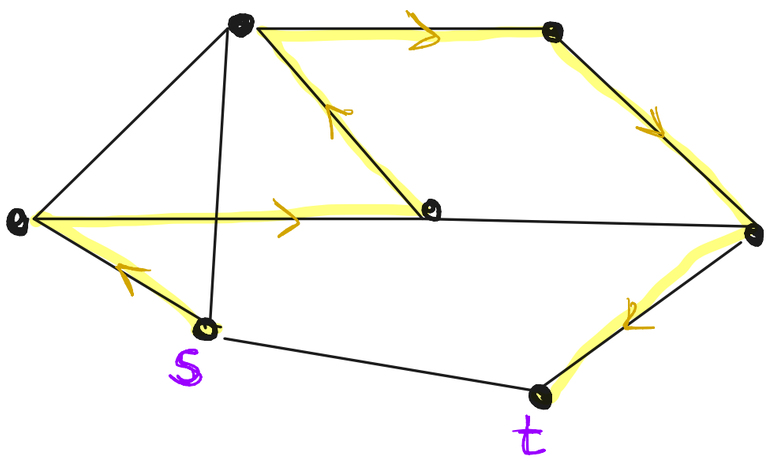
NP: When is H easy to Find a Needle in Haystack?

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)

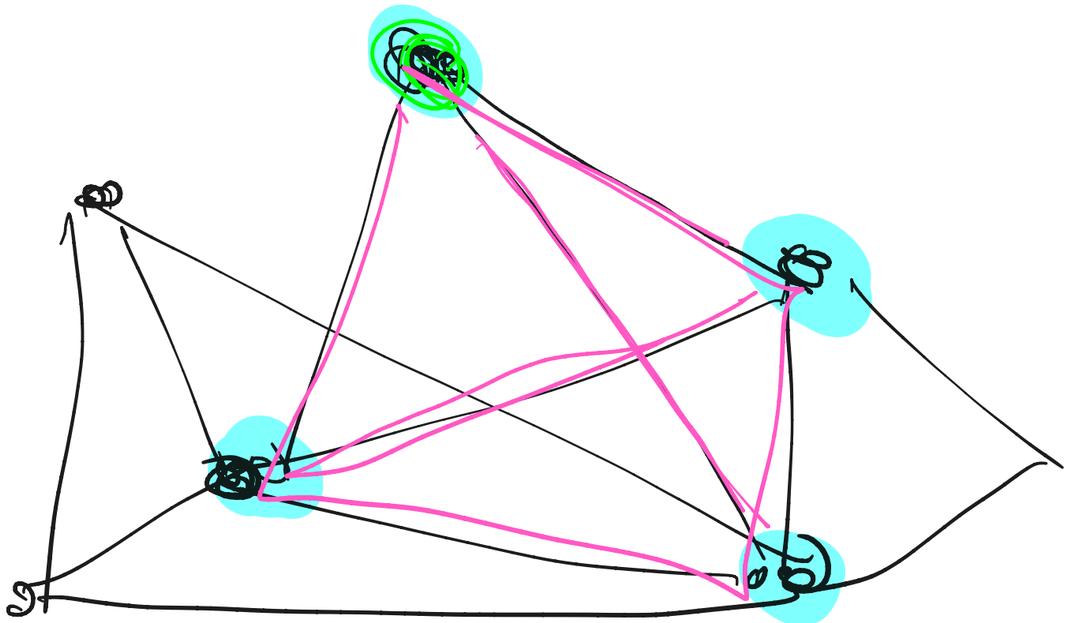
g:



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

Does g have a clique of size k?

g:



← g has a $k=4$ clique but no $k=5$ clique

a size k clique in g

is a subset of k vertices in g such that all pairs of vertices are connected by an edge.

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)

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 $k \geq 0$

Runtime of a Nondet TM ~~B~~ on w

max over all computation paths

Runtime of $M(w, \vec{r})$

Let L be a Language, $L \subseteq \{0,1\}^*$

polytime = $poly(n)$

A <u>verifier</u> for L is a TM V

V takes 2 inputs: $V(w, c)$

input to L ↑ ↖ extra input

A <u>polytime verifier V for L</u> is a TM $V(w,c)$ such that:

① V runs in time $poly(|w|)$ $|w|$ = length of w

② $\forall w \in L \ \exists c$ such that $V(w,c)$ accepts

$\forall w \notin L \ \forall c \ V(w,c)$ rejects

<u>Defn</u> (Alternative defn of NP)

$L \in NP$ if there is a polytime verifier for L

the (even more Famous) class NP

Defn 1 $NP = \{ L \mid L$ 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 V where $L = \{w \mid V(w,c)$ accepts $\}$ for some 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 $NP = \{ L \mid L$ 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

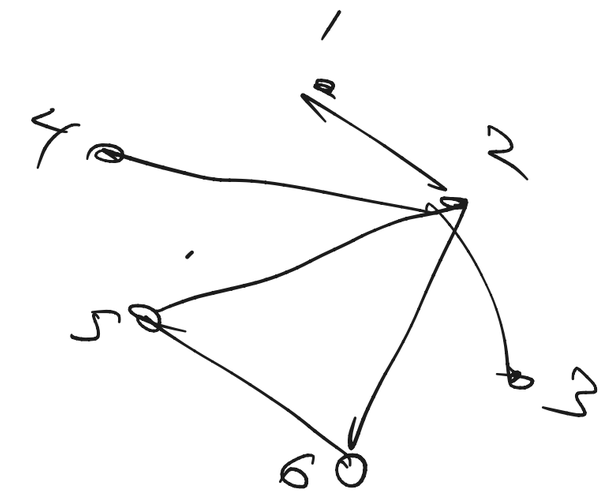
If this computation path (described by c) accepts then accept $\langle w, c \rangle$; otherwise reject

Examples of Languages in NP

① Any $L \in P$ is also in NP

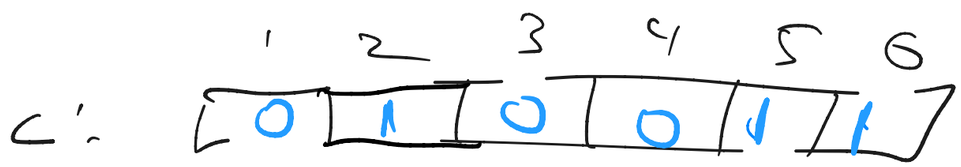
Verifier V on input (w, c) : ignore c and just run polytime alg for L on input w.

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., $(i, j) \in E$)



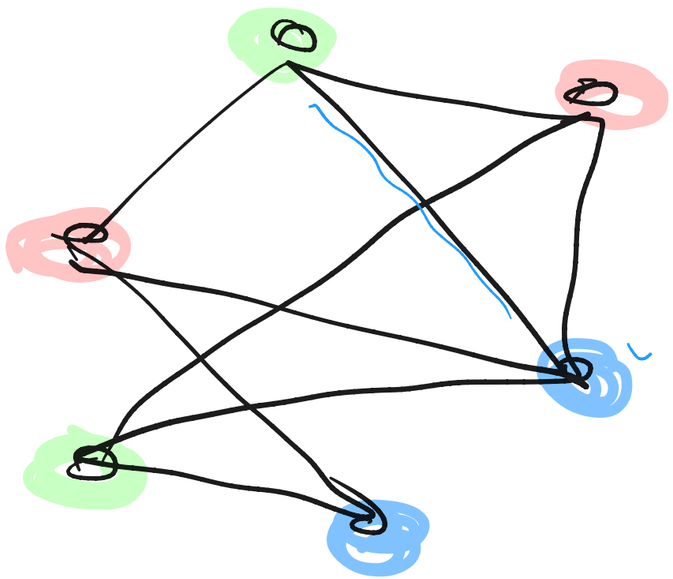
Another Example

Input is an undirected graph g.
Accept g iff g has a proper 4-coloring

4 color theorem
g is 4-colorable iff g is planar

Example

$g \rightarrow$

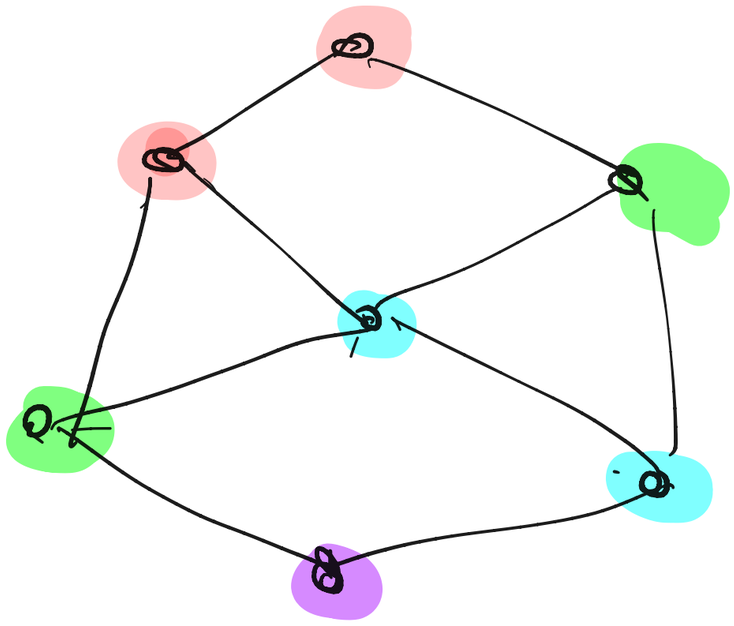


A proper 4-coloring assigns a color (out of 4 colors) to each vertex of g such that $\forall$ edge (i,j) in g, $color(i) \neq color(j)$

$V(g, c)$: if g has a proper 4 coloring then $\exists c$ st $V(g, c)$ accepts
if g has no proper 4 coloring $\rightarrow \forall c$ $V(g, c)$ rejects

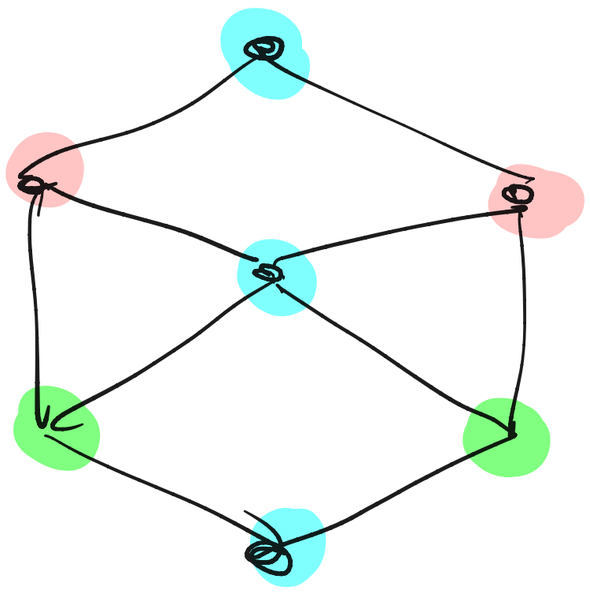
g:

$K=4$



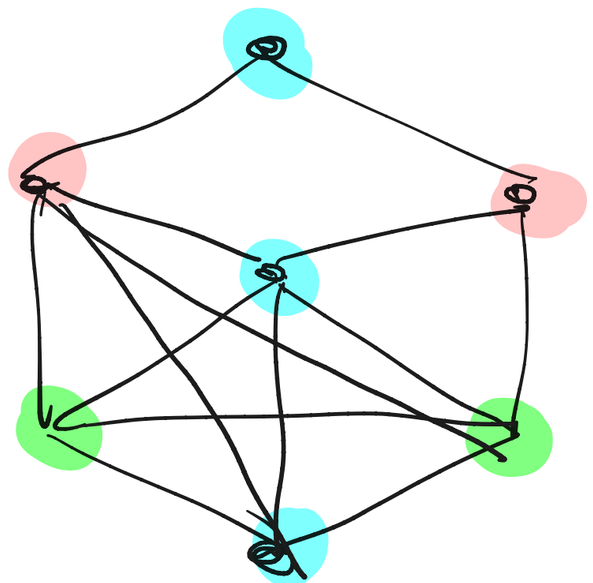
If C is the coloring in picture

$$V(g, C) = 0$$



$\longleftarrow$ $V(g, C) = 1$

K=4 coloring



⟵ For this graph

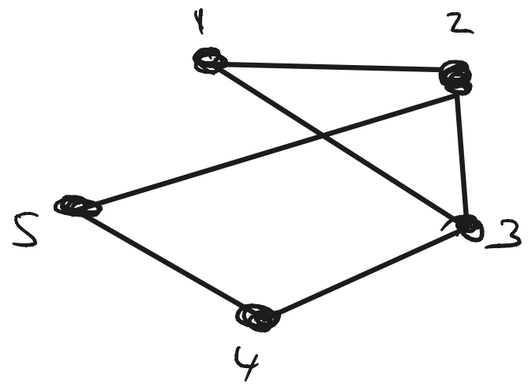
$\forall c$

$$V(g, c) = 0$$

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

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

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

m ← # 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