Last few days we talked about class of P.

There are some problems that we cannot avoid brute force search space.

One important discovery concerning these type of problems shows that the complexities of many problems are linked.

Definition

A verifier for a Language A is an algorithm V, where

$$A = \{w \mid V \text{ accepts } \langle w, c \rangle \text{ for some string } c\}$$

We measure the time of the verifier only in terms of the length of the w, so a polynomial time verifier runs in polynomial time in the length of w. A language A is polynomially verifiable if it has polynomial time verifier.

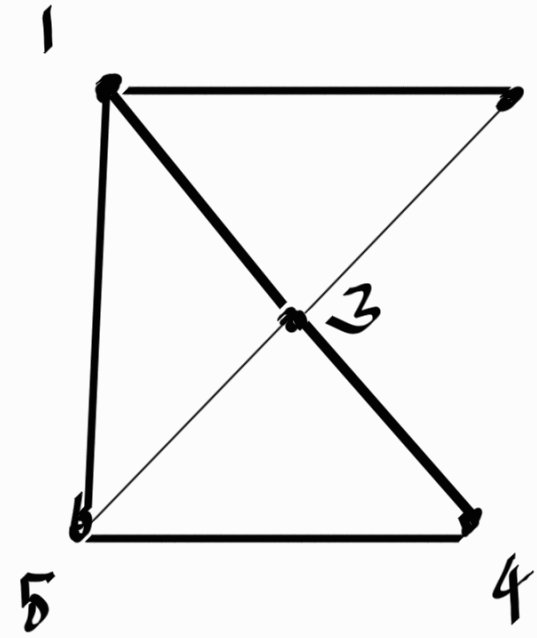
$$w \in A \iff \exists c : V(w, c) = \text{YES}$$

$$w \notin A \iff \forall c : V(w, c) = \text{NO}$$

c - some additional information to help V to verify that w is a member of the language A.

Hamiltonian Cycle

Given a graph $G = (V, E)$, is there a cycle that visits every vertex exactly once and return to the original vertex?



is there a HC in G?

<1, 2, 3, 4, 5, 1>

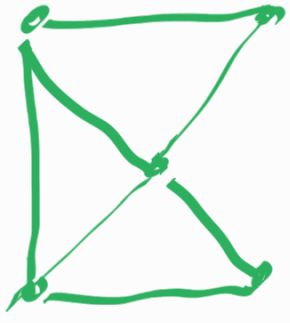
<2, 1, 3, 4, 5, 1, 2)

for this problem we can come up with a polynomial time verifier.

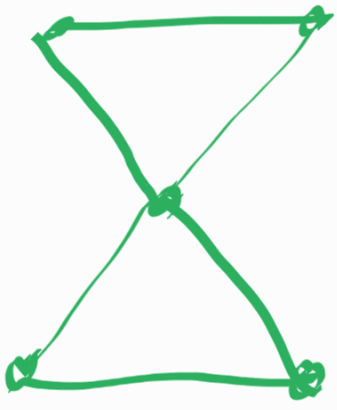
solving the problem $\neq$ verifying the problem

A language A is polynomially verifiable if it has a verifier that runs in polynomial time in the length of the input w.

$\overline{HC} = \{ G \mid G$ has no hamiltonian cycle $\}$



G_1



G_2

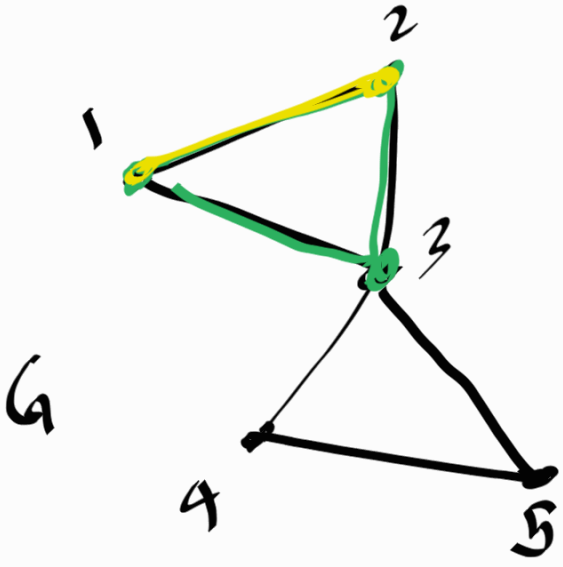
for $\overline{HC}$ we cannot create a polynomial time verifier.

Definition

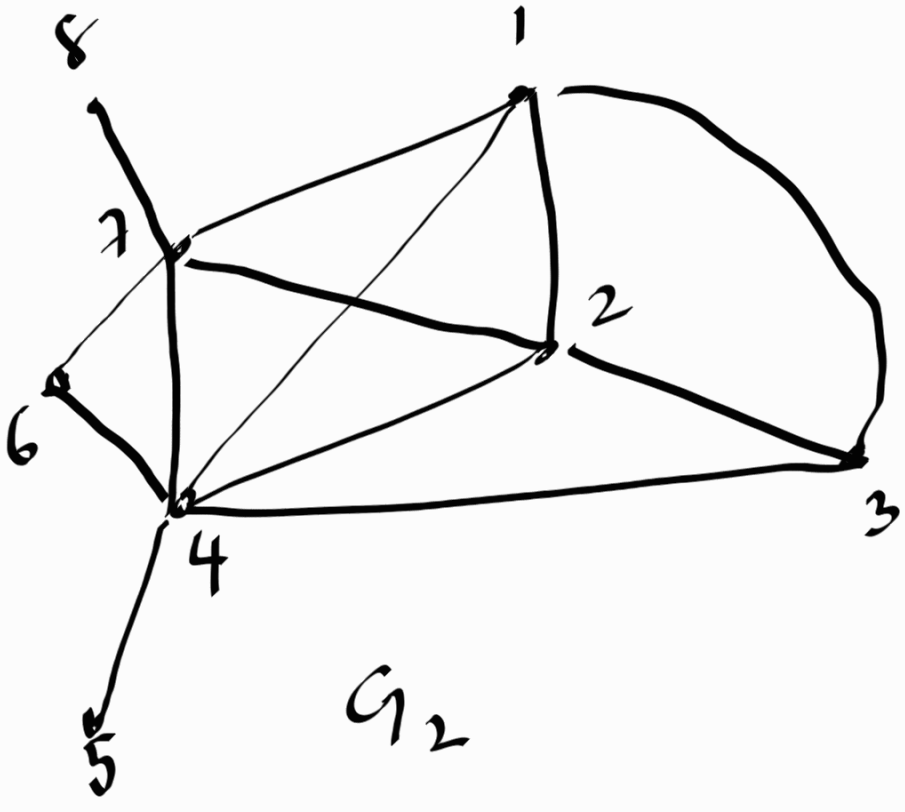
NP is the class of languages that has polynomial time verifiers.

The term NP comes from nondeterministically polynomial time.

CLIQUE = $\{\langle G, k\rangle \mid$ G is an undirected graph of n vertices which contains a k-clique or k-complete-subgraph$\}$.



G has a 3-clique
2-clique



$\langle 1, 7, 4, 2\rangle$

$\langle 1, 2, 3, 4\rangle$

$\langle 1, 7, 4, 3\rangle$ x

Let us define a polynomial time verifier for CLIQUE

A = " on input $\langle \langle G, k \rangle, c \rangle$

① Test whether c is a set of k nodes in G.

② Test if G contains all $\binom{k}{2}$ edges containing nodes in c.

③ If both passes, accept
otherwise, reject

Run time complexity $O\left(nk + \binom{k}{2}\right)$

$= O\left(\left(\binom{k}{2}\right)\right)$

There is a different definition for NP.

Theorem 7.22

A language is NP itf it is decided by some nondeterministic polynomial time Turing machine.

Def of nondeterministic TM.

In a NTM, at any point in a computation, machine may proceed according to several possibilities

$$\delta : Q \times \Gamma \longrightarrow P(Q \times \Gamma \times \{L, R\})$$

Theorem 3.16

Every NTM has an equivalent TM.

- refer Theorem 3.16 for the proof

Proof Idea:

Proof Idea: We show that we can convert a poly time verifier to an equivalent polynomial time NTM and vice versa.

NTM simulates the verifier by guessing all possible certificates. The verifier simulates the NTM by using the accepting branch as the certificate.

Side Note!

Think of a NTM as a TM that can compute multiple choices simultaneously.

Analogy

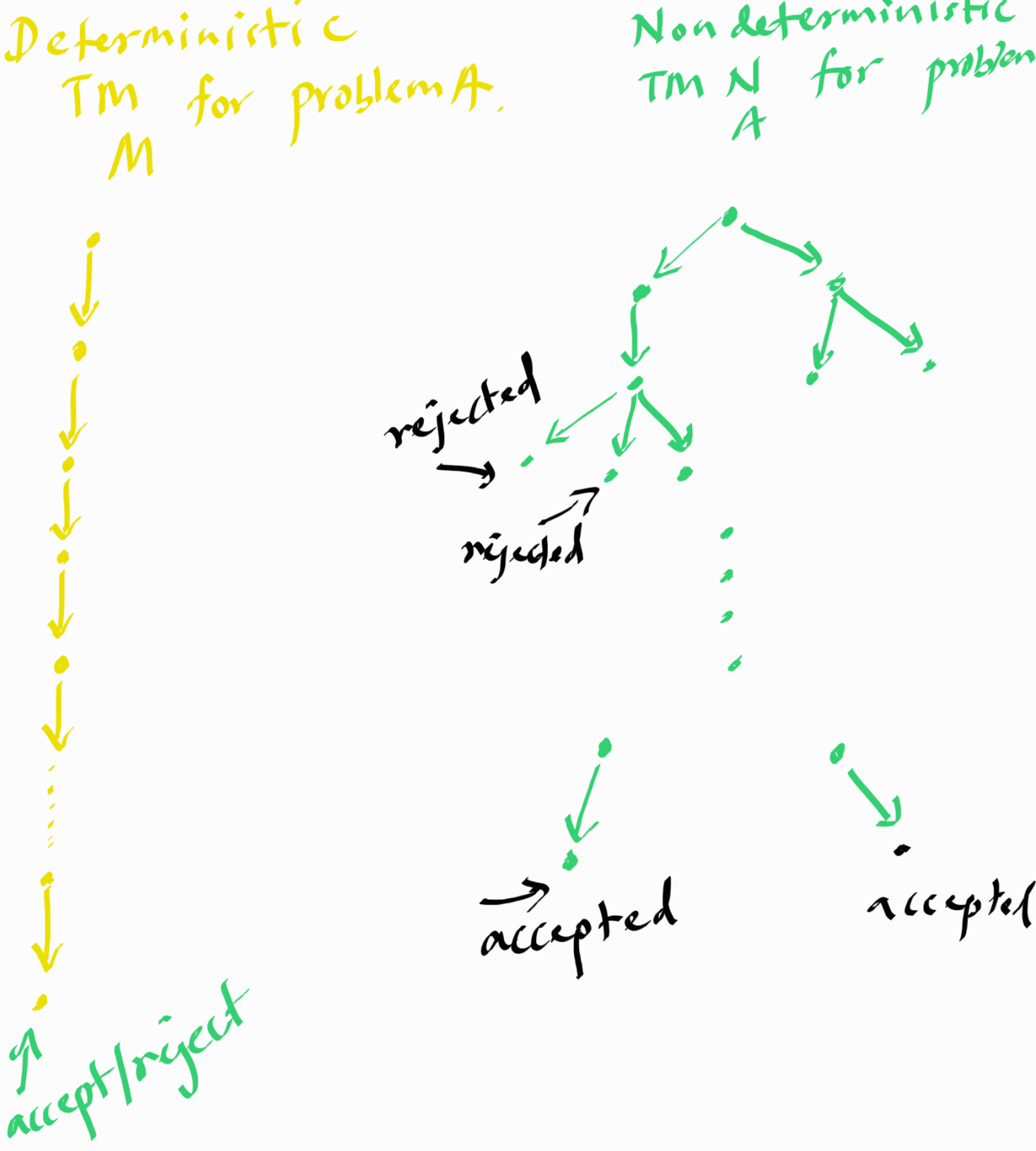
Model	Real-world analogy
TM	following recipes one by one.
NTM	Instantly checking all possible recipes and picking the right one.

NTM can be deciders as well.

Definition

Let N be a nondeterministic turing machine that is a decider. The running time of N is the function $f: \mathbb{N} \longrightarrow \mathbb{N}$, where $f(n)$ is the maximum number of steps that N uses on any branch of its computation on any input of length n.

If a NTM is a decider, then every computation path must halt.



Example: CLIQUE ∈ NP

suppose NTM A decides CLIQUE

A = "on input $\langle G, k \rangle$, G is a undirected graph, $k \leq n$ = # of vertices in the graph.

1. Nondeterministically select a set C of k nodes from G.
2. Test if G contains all $\binom{k}{2}$ edges connecting nodes in C.
3. If the test passes, accept. Otherwise, reject.

N runs in $O\left(\binom{k}{2}\right) = O(n^2)$

$NTIME(t(n)) = \{L \mid L$ is a language decided by $O(t(n))$ time nondeterministic TM$\}$.

$NP = \bigcup_k NTIME(n^k)$, k is a constant

∴ CLIQUE $\in$ NP

Recap

P = the class of languages for which membership can be decided quickly.

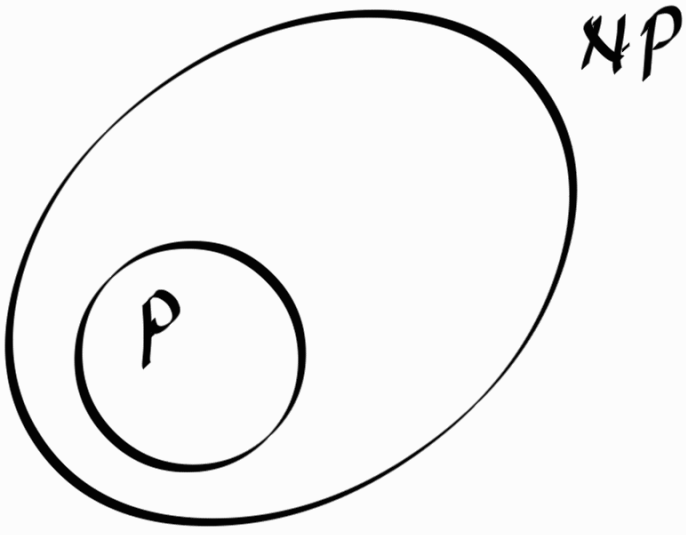
NP = the class of languages for which membership can be verified quickly.

Languages like CLIQUE are in NP, but we donot know whether they are in P.

$P \stackrel{?}{=} NP$

$P \subseteq NP$ ✓

$P \stackrel{?}{\supseteq} NP$



In NP and P, we only have decison problems.

- No optimization problems.