Def 7.12: The class $P$ consists of languages that are decidable in polynomial time, i.e.,

$$P = \bigcup_k \text{TIME}(n^k)$$

Example:

$\text{PATH} = \{ <G,s,t> \mid G \text{ is a digraph that has a path from } s \text{ to } t \}$

- $n = \text{length of the input } <G,s,t>$ - we need to encode vertices, edges, $s$, $t$
- $\geq \text{#vertices}$
- Crude running time bound $\leq O(\text{(#vertices)}^2) = O(n^2)$
- Every iteration, look through all neighbors $\leq \text{#vertices}$
- #iterations $\leq \text{#vertices}$

Remark 2:

BFS can be implemented in time $O(\text{#vertices} + \text{#edges}) = O(n)$
The Class P

Def 7.12: The class P consists of languages that are decidable in polynomial time, i.e.,

\[ P = \bigcup_k \text{TIME}(n^k) \]

Example:

RELPRIME = \{ <x,y> | x and y are relatively primes \}

Euclidean algo to find the gcd
if gcd = 1 return "relatively prime"
else return "not rel.pr."

\[ \rightarrow \text{needs } O(\log x + \log y) \text{ steps} \]
\[ = O(n) \]

what is \( n \), the length of the input?

\[ n = \lceil \log x + 1 \rceil + \lceil \log y + 1 \rceil + 1 \]

length of \( x \) length of \( y \)

\[ \log \text{ is base } 10 \]
Def 7.12: The class $P$ consists of languages that are decidable in polynomial time, i.e.,

\[ P = \bigcup_k \text{TIME}(n^k) \]

Example:

PRIME = \{ <x> \mid x \text{ is a prime} \}

\[
\begin{align*}
\text{for } i=2 \text{ to } \sqrt{x}: \\
& \quad \text{if } i \text{ divides } x, \text{ return "not prime"} \\
& \quad \text{return "prime"}
\end{align*}
\]

\[
\text{takes } O(\sqrt{n}) = O(n^{1/2}) = O(10^{n/12}) = O(\sqrt[12]{10^n})
\]

\[ n \propto \log x \quad \text{thus} \quad x \approx 10^n \]

\[ \text{exponential time} \]

BUT:
- in 2002 Agrawal-Kayal-Saxena showed PRIMES $\in P$
- we are using a randomized poly-time (fast) algo by Miller-Rabin in practice
The Class NP

Example:

\[ \text{HAMPATH} = \{ <G,s,t> \mid G \text{ is digraphs with Hamiltonian path from } s \text{ to } t \} \]

We do not know if this can be done in poly-time.

\[
\begin{align*}
\text{nondet., poly-time edge:} & \\
\{ & \text{nondeterministically generate a sequence of vertices of length } \#\text{vertices} \\
\{ & \text{deterministically verify:} \\
& \text{every vertex visited, start at } s, \text{ end at } t \\
\end{align*}
\]

\[ \text{takes } O(\#\text{vertices}) \text{ steps} \]

\[ \text{takes } O(\#\text{vertices}) \text{ steps (or } O(\#\text{vertices})^2 \text{ if a bit inefficient)} \]
The Class NP

Example:

$\text{COMPOSITES} = \{ x \mid x = pq, \text{ for some } p, q > 1 \}$

\[ \text{nondet. poly-time:} \]

1. nondeterministically generate $p$ \hspace{1cm} \leftarrow \text{takes } O(n) \text{ steps}
2. deterministically verify if $p$ divides $x$ \hspace{1cm} \leftarrow \text{takes } O(n) \text{ steps}
   \hspace{1cm} (p \neq 1, x) \hspace{1cm} (e.g. \text{by Euclid algorithm})
The Class NP

Def 7.18: A **verifier** for a language $A$ is an algorithm $A$, where

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

**Polynomial-time verifier** runs in (deterministic) time polynomial in the length of $w$. The string $c$ is called the **certificate**, or **proof**, of the membership in $A$.

**verifier** → the part 2 of our nondet. algorithms

need for NP: **verifier** runs deterministically in poly-time

part 1 ("guess the solution" part takes a polynomial # steps)
Def 7.18: A **verifier** for a language $A$ is an algorithm $A$, where

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

**Polynomial-time verifier** runs in (deterministic) time polynomial in the length of $w$. The string $c$ is called the **certificate**, or **proof**, of the membership in $A$.

Def 7.19: **NP** is the class of languages that have polynomial time verifiers.
The Class NP

Thm 7.20: A language is in NP iff it is decided by some nondeterministic polynomial-time TM.

Def 7.21: NTIME(t(n)) = \{ L \mid L \text{ is a language decided by a } O(t(n))-\text{time nondeterministic TM} \}.

Thus, \( \text{NP} = \bigcup_k \text{NTIME}(n^k) \)
The Class NP

Thm 7.20: A language is in NP iff it is decided by some nondeterministic polynomial-time TM.

Def 7.21: NTIME(t(n)) = \{ L \mid L \text{ is a language decided by a } O(t(n))-\text{time nondeterministic TM} \}.

Thus, \[ NP = \bigcup_k NTIME(n^k) \]
The Class NP

Example:

CLIQUE = \{ <G,k> \mid G \text{ is undirected graph with a } k\text{-clique} \}

G:

- a clique of size 3
- k = 3, YES, 3-clique exists
- k = 4, YES, 4-clique exists

in NP:

1. guess k vertices
2. deterministically check if form a clique
The Class NP

Example:

\[ SAT = \{ <\phi> \mid \phi \text{ is a satisfiable Boolean formula} \} \]

\[
\phi = (x_1 \lor \overline{x}_2) \land (x_1 \lor \overline{x}_2 \lor \overline{x}_3) \land (\overline{x}_1 \lor \overline{x}_2 \lor \overline{x}_3)
\]

\[
\text{e.g., can be made true, we call it satisfiable}
\]

\[
\text{e.g., if } x_1 = \text{false, } x_2 = \text{true, } x_3 = \text{true}
\]

in NP:

1. guess the assignment of the variables \((x_1, x_2, x_3, \ldots)\)
2. determine if \(\phi\) for the given assignment is true?
The Class NP

Wrapping up:
- P - exists polynomial-time algorithm
- NP - exists polynomial-time verifier

BIG open problem:

Is P = NP ???

Note: also exists a class coNP, the class of complements of problems in NP (e.g. CLIQUE\(^c\), “is every clique of a given graph of different size than k?”). We do not know if NP = coNP.