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:**

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

**Algorithm:**

- **BFS**
  - If gets to $t$ return "YES, path exists"
  - Otherwise return "NO"

**Running time:**

1. $\#\text{iterations} \leq \#\text{vertices}$
2. In every iteration go through all neighbors $\leq \#\text{vertices}$

**Overall:** $O\left(\left(\#\text{vertices}\right)^t\right) = O(n^3)$

**Remark:**

BFS can be implemented faster, in time $O(\#\text{vertices} + \#\text{edges}) = O(n)$
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:

$\text{RELPRIME} = \{ <x,y> \mid x \text{ and } y \text{ are relatively primes} \}$

\begin{itemize}
  \item \text{idea 1:} for $i \in \{2 \to \sqrt{\text{min}(x,y)} \}$
    \begin{itemize}
      \item if $i$ divides both $x$ and $y$, then return "NO, not rel. primes"
      \item return "YES, rel. primes"
    \end{itemize}
    \begin{itemize}
      \item running time: $O(\sqrt{\text{min}(x,y)}) \approx O(\sqrt{n}) = O(\text{exp.})$
    \end{itemize}

  \item \text{idea 2:} Euclid Alg. for finding $\gcd(x,y)$
    \begin{itemize}
      \item if $\gcd = 1$ return "YES"
      \item else return "NO"
    \end{itemize}
    \begin{itemize}
      \item needs $O(\log x + \log y)$ steps
      \item $= O(n)$
    \end{itemize}
\end{itemize}

$\text{RELPRIME} \in P$
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:

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

Idea:

\[
\begin{array}{l}
\text{for } i = 2 \text{ to } \sqrt{x} : \\
\quad \text{if } i \text{ divides } x, \text{ return "NO, not a prime"} \\
\quad \text{return "YES"}
\end{array}
\]

There exists a (fast) randomized algo for testing primality, due to Miller-Rabin.

Since 2002, we know that $\text{PRIME} \in P$, Agrawal-Kayal-Saxena.
The Class NP

Example:

$$\text{HAMPATH} = \{ <G, s, t> \mid G \text{ is digraph with Hamiltonian path from } s \text{ to } t \}$$

we do not know whether HAMPATH ∈ P

but we do have a nondet. poly-time algo:

1. nondeterministically generate a sequence of vertices of length $n$ vertices
2. deterministically verify whether:
   - every vertex appears exactly once
   - every pair of adjacent vertices is connected by an edge
     starts at $s$ and ends at $t$

formal:

- $O(n)$ running time (nondeterministic)
The Class NP

Example:

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

handel. poly-time:

1. nondeterministically guess $p$
2. deterministically verify:
   - $p$ divides $x$
   - $p > 1$ and $p < x$

poly-time $O(n)$
by e.g. Euclid Alg
The Class NP

Def 7.18: A *verifier* for a language \( A \) is an algorithm \( A \), where

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

*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 \( A \).

\[\begin{align*}
&\text{nondet. polynomial-time:} \\
&1. \text{ guess the solution (nondet. part) } \rightarrow \text{ poly-time} \\
&2. \text{ deterministically verify } \rightarrow \text{ poly-time}
\end{align*}\]
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 $A$.

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

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

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

Example:

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

\( k = 3 \)

YES, there is a 3-clique

\( k = 4 \)

YES, there is a 4-clique

\( k = 5 \)

NO

\( k \) vertices, every pair connected by an edge

mondet: poly-time

1. wonder guess k vertices
2. check verify if they form a clique
The Class NP

Example:

\[ SAT = \{ \langle \phi \rangle \mid \phi \text{ is a satisfiable Boolean formula} \} \]

\[ \phi = (x_1 \lor x_2 \lor \neg x_3) \land (x_2 \lor x_4) \land (\neg x_1 \lor \neg x_4) \]

Can be made true (i.e., it is satisfiable) e.g. take:

\begin{align*}
& x_1 = \text{false} \\
& x_2 = \text{true} \\
& x_3 = ? \\
& x_4 = ?
\end{align*}

In NP:

1. nondet. guess the T/F assignment to the literals/variables
2. det. verify whether \( \phi \) is true
The Class NP

Wrapping up:
- P - exists polynomial-time algorithm
- NP - exits 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.