§6.4. The class $\mathcal{NP}$ defined

Our **aspiration**: We want to identify problems that are *not* in class $\mathcal{P}$.

We *suspect* Ham-Cycle, TSP, Indep-Set, Partition, SAT, and 3-SAT are *not* in class $\mathcal{P}$. They all happen to be in class $\mathcal{NP}$.

A language L is in class $\mathcal{NP}$ if there exists a nondeterministic Turing machine M such that

- All computations are bounded by a polynomial in the size of the input (and hence halt)
- There are no false positives:
  If $w \notin L$ then all computations of M on w halt n
- There may be some false negatives, but there must be at least one true positive
  If $w \in L$, then $\exists$ a computation of M on w that halts y

Notice how cleverly the nodeterministic "algorithms" of [Examples 6.4.(1&2)] exploit the *fundamental asymmetry* in the definition of nondeterministic time-bounded computation. They try out all possible solutions to the problem in hand in independent computations, and accept as soon as they discover one that works—oblivious of the others that do not. LP pg 295

- $\mathcal{P} \subseteq \mathcal{NP}$, just as $R \subseteq RE$.
- $\mathcal{P} \subseteq \mathcal{EXP}$, but $\mathcal{P} \neq \mathcal{EXP}$.
  (since $E \in \mathcal{EXP}$ but $E \notin \mathcal{P}$, Theorem 6.1.2)
- $\mathcal{NP} \subseteq \mathcal{EXP}$. (Theorem 6.4.1)
- These imply that $\mathcal{P} \subseteq \mathcal{NP} \subseteq \mathcal{EXP}$,
  but also that $\mathcal{P} = \mathcal{NP}$ and $\mathcal{NP} = \mathcal{EXP}$ cannot *both* be true.
- We don't know whether $\mathcal{P} \neq \mathcal{NP}$ or $\mathcal{NP} \neq \mathcal{EXP}$ (possibly both are true).

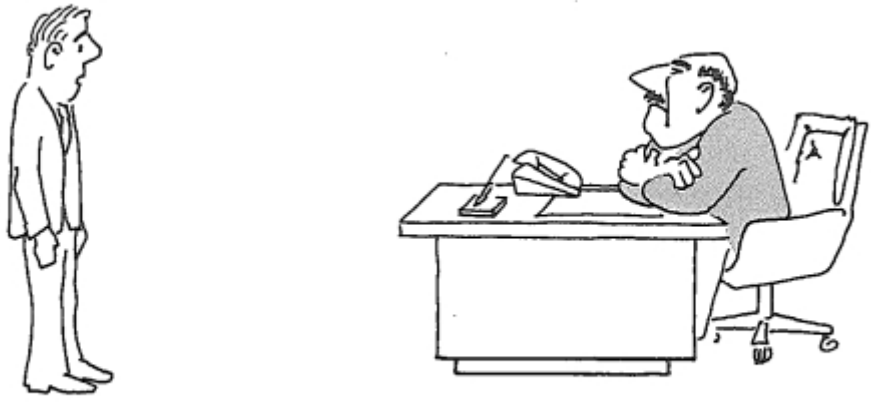
Alternative definition of $\mathcal{NP}$:

> $L \in \mathcal{NP}$ if there exists a Turing machine M such that for all $w \in L$ there exists a string y such that $|y|$ is polynomial in $|w|$ and M computes whether $w \in L$ in polynomial time when given $w; y$ as input.

y is a **succinct certificate**.

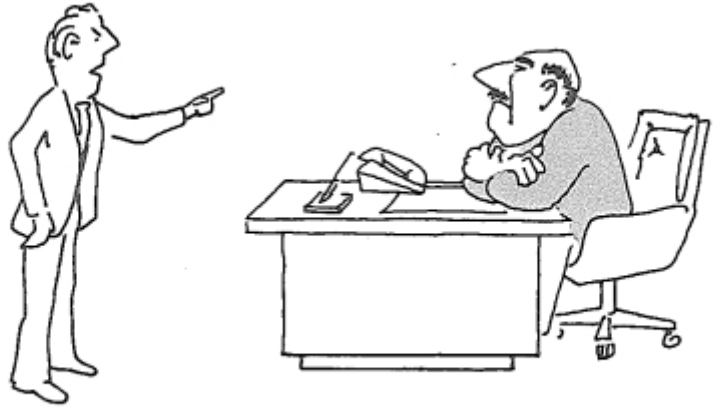
CLRS's definition of class $\mathcal{NP}$:

> *The* **complexity class** $\mathcal{NP}$ *is the class of languages that can be verified by a polynomial-time algorithm. More precisely, a language L belongs to* $\mathcal{NP}$ *if and only if there exist a two-input polynomial-time algorithm A and a constant c such that*
>
> $$L = \{x \in \{0,1\}* \mid \exists \text{ a certificate } y \text{ with } |y| = O(|x|^c) \text{ such that } A(x,y) = 1\}$$
>
> *We say that algorithm A* **verifies** *language L* **in polynomial time**. *CLRS pg 1064*



"I can't find an efficient algorithm, I guess I'm just too dumb."

Garey and Johnson, *Computers and Intractability*, Freeman, 1979; pg 2



"I can't find an efficient algorithm, because no such algorithm is possible!"

Garey and Johnson, *Computers and Intractability*, Freeman, 1979; pg 2

"I can't find an efficient algorithm, but neither can all these famous people."

Garey and Johnson, *Computers and Intractability*, Freeman, 1979; pg 3

Revisiting the **nature of a reduction**:

- A reduction from A to B uses a solution to B to build a solution to A.
  "If we can solve B [within constraints], then we can solve A [within analogous constraints]."
- To show a polynomial reduction from L_1 to L_2 requires us to
  - Describe a function τ from L_1-candidates to L_2-candidates
  - Show that τ is computed in polynomial time.
  - Show that $\forall\ x \in L_1$-candidates, $x \in L_1$ iff $\tau(x) \in L_2$.

  So the reduction turns an instance of "problem" L_1 to an instance of "problem" L_2.
- A reduction from A to B is evidence that B is at least as hard as A.

We need to show there exists a Hamiltonian cycle in $G = (V, E)$ iff there exists a satisfying truth assignment to the formula.

Proof *($\Rightarrow$) Suppose T satisfies the formula. Then for each $v_i \in V$ (that is, each $i \in [1, n]$), exactly one x_{ij} is true. For each $j \in [1, n]$ (that is, for each position in the cycle), exactly one x_{ij} is true. If x_{ij} and x_{kj+1} are both true, then $(v_i, v_k) \in E$.*

($\Leftarrow$) Conversely, suppose there exists a Hamiltonian cycle for G. Then the truth assignment T where $T(x_{ij}) = \top$ iff v_i is in the jth position in the cycle satisfies the formula. $\square$

This is **bad news** for SAT.

- If we could solve SAT in polynomial time [or any other time category], then we could solve HamCycle in polynomial time [or whatever category]
- If we prove HamCycle can't be solved in polynomial time, then SAT also can't.
- If we prove SAT can't be done in polynomial time, then the story still isn't over for HamCycle.

Example 7.1.2: Reducing Knapsack to Partition

Knapsack: Given a set S of n integers and capacity k, is there [find] a subset of S that sum exactly to k?
Partition: Given a set S of n integers, can they be partitioned exactly in half (in terms of their sum)?

Let $S = \{a_1, a_2, \ldots a_3\}, k$ be an instance of Knapsack.

Let $H = \frac{1}{2}\sum_{a_i \in S} a_i$ and make set $S_2 = S \cup \{2H + 2k, 4H\}$. This is an instance of Partition.
Suppose a partition exists for S_2, call it $P \cup \{4H\}$ and $(S - P) \cup \{2H + 2k\}$ for some $P \subseteq S$. Then

$$\begin{array}{rcl} 4H + \sum_{a_i \in P} a_i & = & 2H + 2k + \sum_{a_i \in S-P} a_i \\ 4H + 2\sum_{a_i \in P} a_i & = & 2H + 2k + \sum_{a_i \in S} a_i = 2H + 2k + 2H = 4H + 2k \\ \sum_{a_i \in P} a_i & = & k \end{array}$$

And so P is our solution to Knapsack.

Conversely, suppose there exists $P \subseteq S$, a solution to Knapsack, that is, $\sum_{a_i \in P} a_i = k$. Work backwards algebraically ...