Schedule (recent and imminent)

Date	**Reading**	**In class**
Fri, Nov 22	6 (whole chapter)	Sections 5.(4,6,7), ~~definitions from 6.(1 & 2)~~
Mon, Nov 25	Reread 6.1 Reread 6.2 through pg 282	6.1 Definition of class $\mathcal{P}$ etc 6.2 Reachability, HamCycle
Mon, Dec 2	Reread rest of 6.2 Reread 6.3	6.2 TSP, IndependentSet Clique, Partition 6.3 Boolean satisfiability
Wed, Dec 4	Reread 6.4 Read 7.2	6.4 The class NP 7.1 Polynomial-time reductions

Boolean Satisfiability (SAT) and family

- A **literal** is an occurrence of a variable or its negation: x or $\sim x$
- A **clause** is a disjunction of literals: $x_1 \vee x_2 \vee \sim x_3$
- A **formula** is a conjunction of clauses: $(x_1 \vee \sim x_2 \vee x_3) \wedge (x_2 \vee x_4 \vee \sim x_5)$
- A **truth assignment** is a mapping from variables to $\{\top, \bot\}$
- A truth assignment **satisfies** a formula is $\forall$ clauses $\exists$ a true literal

The **Satisfiability** problem is, given a formula, does a satisfying truth assignment exist?

2-SAT: Given a formula in which each clause has no more than two literals . . .

3-SAT: Given a formula in which each clause has no more than three literals . . .

Claim: This algorithm produces a satisfying truth assignment iff one exists.

Proof. *($\Rightarrow$) [If the algorithm returns a truth assignment, that assignment indeed satisfies the given formula.]*

In the original/initial call to `purge`*, any individual variable assignment that results must be part of any satisfying truth assignment, since the formula cannot be satisfied without the variable assignments done by* `purge`*.*

Invariant *for the main loop: The (partial) assignment to the variables is part of a satisfying (complete) truth assignment, iff one exists.*

Initialization: *Implied by what is said above.*

Maintenance: *Suppose the partial assignment at the beginning of an iteration is part of a complete satisfying truth assignment. This iteration assigns to one variable. If that assignment were not part of a CSTA that also includes the current partial assignment, it would be rejected by the call to purge. Hence the updated partial assignment is also part of a CSTA.*

Termination. *There is at most one iteration for each variable. Since there are a finite number of variables, the loop terminates. When the loop terminates, all variables are assigned, and, by the loop invariant, that assignment is "part of" a CSTA. There fore the assignment is a CSTA.*

($\Leftarrow$) [If a truth assignment exists, the algorithm returns one.]

Suppose a truth assignment exists, and suppose the algorithm doesn't find one. From that we can derive a contradiction. $\square$

Why don't the polynomial-time algorithms for 2-SAT work for 3-SAT?

The `purge` routine is based on the premise that if a guess doesn't fail, then it is safe.

$$(\sim x_1 \vee x_2 \vee \sim x_3) \wedge (\sim x_1 \vee \sim x_2 \vee \sim x_3) \wedge (\sim x_1 \vee x_2 \vee x_3) \wedge (\sim x_1 \vee \sim x_2 \vee x_3)$$

What if we guess $x_1 := \top$?

§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

LP pg 293

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).

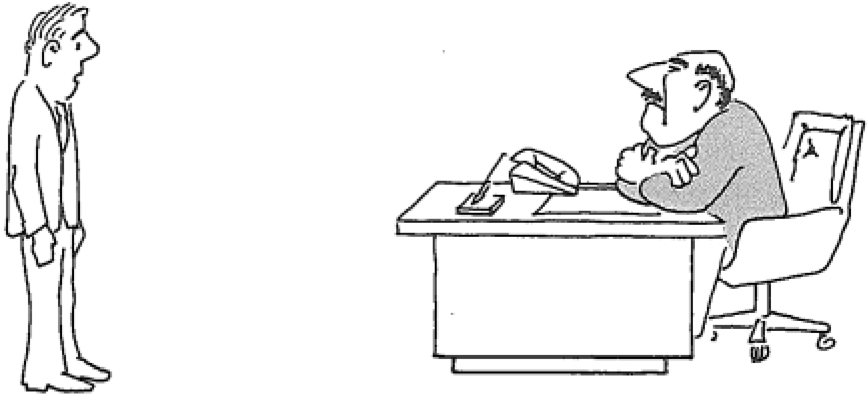
Alternative definition of $\mathcal{NP}$:

> $L \in \mathcal{NP}$ 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**.

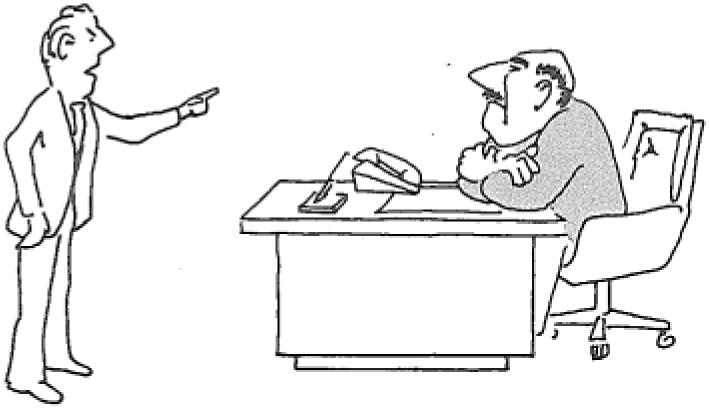
CLRS's definition of class $\mathcal{NP}$:

> *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*
>
> $$L = \{x \in \{0,1\}* \mid \begin{array}{l} \exists \text{ a certificate } y \text{ with } |y| = O(|x|^c) \\ \text{such that } A(x,y) = 1\} \end{array}$$
>
> *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.