COMP 330
NOV 12, 2010

PAUL HUSSMAN

Some other notation we often use

$A \Rightarrow B$ meaning $\neg A \vee B$

$A \Leftrightarrow B$ meaning $(\neg A \vee B) \wedge (\neg B \vee A)$

Example: $\forall x\,(Even(x) \wedge (x>2)) \Rightarrow \exists y \exists z\,(Prime(y) \wedge Prime(z) \wedge (x+y=z))$

Solution worth 1 million ↓ (GOLDBACH'S CONJECTURE)

**Definition:** An occurrence of a variable $x$ in a formula $A$ is called **BOUND** if it appears in a subformula of the form $\forall x\,B$ or $\exists x\,B$.
Otherwise this occurrence is called **FREE**

Example: In $\exists y\,(x=y+y)$ the occurrence of $x$ is FREE while both occurences of $y$ are BOUND.

A formula with no free variables is called a **SENTENCE**.

So far we talked about syntax, and now we discuss the INTERPRETATIONS of these formulas.

**Definition:** A **MODEL** $M$ for a vocabulary consists of:
- Universe: a set $U$ (possibly infinite), the variables range over $U$.
- $M$ assigns meanings to function symbols in the following way:
  For every $n$-ary function, $f$ and every $n$ elements $u_1,\dots,u_n$ in the universe, it assigns a value from $U$ to $f(u_1,\dots,u_n)$
- $M$ assigns values to relation symbols for every $n$-ary relation symbol $R$ and every $u_1,\dots,u_n \in U$ it assigns a true or false to $R(u_1,\dots,u_n)$.

Remark: The relation symbol $=$ always gets its natural meaning. (It holds if two elements are the same element in the universe)

Remark: Note that every model imposes a true or false value on every sentence.

Example: Consider the sentence: $\forall x\,\exists y\,(x=y+y)$

(i) Consider the model $M_1$ which has $\mathbb{Z}$ universe and $+,=$ are defined in the natural way.
This sentence is FALSE in the model $M_1$.

(ii) Consider the model $M_2$ with $\mathbb{R}$ as its universe and $+,=$ are defined normally.
This sentence is TRUE in the model $M_2$

---

COMP 330
NOV 15 2010

PAUL HUSSMAN

When a model is specified, every sentence becomes TRUE or FALSE.
eg: $\forall x\,\exists y\ x=y+y$ is false in $\mathbb{N}$ but true in $\mathbb{R}$.

**Definition:** The **THEORY** of a model is the set of all true sentences in that model
If $M$ is the model, the $T(M)$ denotes the theory of $M$.

**Definition:** A sentence is **VALID** if and only if it is true in every model.

eg: $\forall x\ P(x) \vee \neg P(x)$ is a valid sentence
eg: $(P(f) \vee \neg P(f)) \vee Q(1,2)$ is a valid sentence

Question: Is this a valid sentence?
$\forall x\,\exists y\ x < x+y \Rightarrow$ NO. Consider the model with universe $\mathbb{N}$ and $+$ is the usual sum and $x<y$ is always false.

PROOF SYSTEMS

Notation: Let $A_1,\dots,A_n$ and $B_1,\dots,B_m$ be sentences in some vocabulary. Fix a model $M$, we write $A_1,\dots,A_n \longrightarrow B_1,\dots,B_m$ meaning that $(A_1 \wedge \dots \wedge A_n) \Rightarrow (B_1 \vee \dots \vee B_m)$ is TRUE in the model. In otherwords if $A_1,\dots,A_n$ are all true then at least one of $B_1,\dots,B_m$ is true.

Some Inference Rules: Let A be a sentence

Weakening Rule: $\dfrac{A_1,\dots,A_n \to B_1,\dots,B_m}{A, A_1,\dots A_n \to B_1,\dots,B_m}$ implies [can add A to left side]

$\dfrac{A_1,\dots,A_n \to B_1,\dots B_m}{A_1,\dots,A_n \to B_1,\dots,B_m, A}$ [can add A to right side]

Exchange Rule: $\dfrac{A_1,\dots A_i, A_{i+1},\dots,A_n \to B_1,\dots,B_m}{A_1,\dots,A_{i-1},A_{i+1},A_i,A_{i+2},\dots A_n \to B_1,\dots,B_m}$, and same goes for the right side.

Contraction: $\dfrac{A_1,\dots,A_n,A,A \to B_1,\dots,B_m}{A_1,\dots,A_n,A \to B_1,\dots,B_m}$ $\qquad$ $\dfrac{A_1,\dots,A_n \to B_1,\dots,B_m,A,A}{A_1,\dots,A_n \to B_1,\dots,B_m,A}$

¬ Introduction: Let $\Gamma = A_1,\dots,A_n$ and $\Delta = B_1,\dots,B_m$

$\dfrac{\Gamma \to \Delta, A}{\neg A, \Gamma \to \Delta}$ $\qquad$ $\dfrac{A,\Gamma \to \Delta}{\Gamma \to \neg A, \Delta}$

∧ Introduction:

$\dfrac{A,B,\Gamma \to \Delta}{(A \wedge B),\Gamma \to \Delta}$ $\qquad$ $\dfrac{\Gamma \to \Delta, A \text{ and } \Delta \to \Gamma, B}{\Gamma \to \Delta, A \wedge B}$

∨ Introduction:

$\dfrac{A,\Gamma \to \Delta \quad B,\Gamma \to \Delta}{A \vee B, \Gamma \to \Delta}$ $\qquad$ $\dfrac{\Gamma \to \Delta, A, B}{\Gamma \to \Delta, A \vee B}$

Cut Rule:

$$\frac{\Gamma \to \Delta, A \quad A, \Gamma \to \Delta}{\Gamma \to \Delta}$$

∀

$$\frac{A(\overset{\text{term}}{t}), \Gamma \to \Delta}{\forall x A(x), \Gamma \to \Delta}$$

∃

$$\frac{\Gamma \to \Delta, A(\overset{\text{term}}{t})}{\Gamma \to \Delta, \exists x A(x)}$$

Theorem: GODEL'S COMPLETENESS THEOREM

For every vocabulary, every valid sentence has a proof using the above rules strictly from axioms $A \to A$ (where A is a sentence.

Remark: We start from $A \to A$ for sentences A and apply the rules, and we will be able to prove $\to B$ for every valid sentence B.

Corollary: The following set is decidable:

$\{A \mid A \text{ is a valid sentence}\}$

Proof: The following TM will decide it:

N = "on every sentence A

for $i = 1, 2, \dots$

compute every proof of length $i$

if A is proved accept, if ¬A is proved, reject"

COMP 330
NOV 15 CONT'D

PAUL HUSSMAN

Theorem: GODEL'S INCOMPLETENESS THEOREM
Consider the model of natural Numbers $(\mathbb{N}, \{+, \times, S, 0, \circ, =\})$. Given any recursively enumerable axioms H and any recursively enumerable set of inference rules, there will be statements in the corresponding theory with no proof.

Peano Arithmetic

① $\forall x \; S(x) \neq 0$
② $\forall x \forall y \; S(x) = S(y) \Rightarrow x = y$
} define $S$ (successor)

③ $\forall x \; (x+0) = x$
④ $\forall x \forall y \; (x + S(y)) = S(x+y)$
} define $+$

⑤ $\forall x \; x \cdot 0 = 0$
⑥ $\forall x \forall y \; x \cdot S(y) = x + x \cdot y$
} define $\cdot$ (multiplication)

⑦ $\forall y_1, \ldots, y_k \; (A(0) \wedge A(x) \Rightarrow A(S(x))) \Rightarrow \forall x \, A(x)$ (induction)
where $A$ is a formula with free variables $y_1, \ldots, y_k$.
Induction Hypothesis.
Peano arithmetic is the set of axioms of $\{1,2,3,4,5,6\} \cup \{7\}$

Note: These two lectures are not on the final. (This one & last one)

COMP 330
NOV 17 2010

PAUL HUSSMAN

Computability: Is there an algorithm that decides L?
Complexity?: Is there an efficient algorithm that decides L?
Efficient: ① Algorithm is fast (halts after not too many steps)
② The algorithm does not use much memory.
③ The algorithm can be implemented using small circuits.

Time complexity ✓
Space complexity
Circuit complexity ✓

$L = \{P \mid P$ is a polynomial w/ integer coefficients and $P$ has a rational root$\}$
Is L decidable?

Definition: Let M be a TM that decides a language L. The RUNNING TIME or TIME COMPLEXITY of L is a function $f: \mathbb{N} \to \mathbb{N}$ such that M takes at most $f(n)$ steps on any input of size $n$.

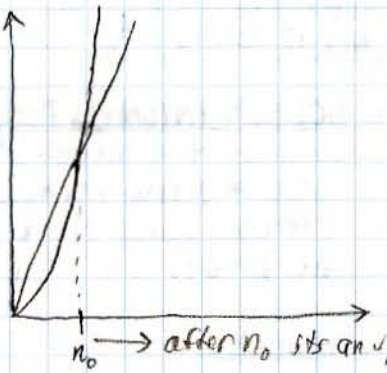
Remark: An algorithm of running time $n^2$ does not have much advantage over an algorithm with running time $n^2 + n$. Moreover, $f(n)$ is usually a complicated expression. So we are only interested in ESTIMATING the ASYMPTOTICS of $f(n)$ as $n$ grows.

Big O and small o

Definition: Let $f$ and $g$ be two functions from $\mathbb{N} \to \mathbb{R}^+$. We say $f = O(g)$ if and only if $\exists c, n_0 > 0$ such that $\forall n \geq n_0$, $f(n) \leq c \cdot g(n)$.

Intuitively, $f = O(g)$ means that growth rate of $f$ is not larger than the growth rate of $g$, (ignoring constant coefficients).

Example
Let $f(n) = 100n$ and $g(n) = n^2$ then $f = O(g)$.
Proof: If $c = 100$ and $n_0 = 1$, then $\forall n > n_0$, $f(n) = 100n \leq 100n^2 = c \cdot f(n)$.
So we showed $\exists c, n_0 > 0 \; \forall n > n_0 \; f(n) \leq c \cdot g(n)$.

Graph of these functions: 


**Example:** $f(n) = 10n$
$g(n) = n$
Then $f = O(g)$
**Proof:** Take $n_0 = 1$ and $c = 10$ then $\forall n > n_0$
$f(n) = 10n \leq c\,g(n) = 10n$.

**Example:** Let $f(n) = n^2$ and $g(n) = 2^n$, then $f = O(g)$.
**Proof:** take $c = 1$ and $n_0 = 10$.
we want to show that $\forall n > n_0$, $n^2 \leq 2^n$. We prove this by induction.
- Induction basis: $(n_0)^2 = 100 \leq 2^{n_0} = 2^{10} = 1024$
- Induction Hypothesis: $n^2 \leq 2^n$ (here $n \geq 10$)
- Induction Step: $(n+1)^2 \leq 2^{n+1}$ ← we prove this.

$$(n+1)^2 = n^2 + 2n + 1 \leq n^2 + \left(\frac{n}{5}\right)n + \frac{n^2}{100} \leq \left(1 + \frac{1}{5} + \frac{1}{100}\right)n^2 \leq 2n^2 \leq 2 \times 2^n = 2^{n+1}.$$

($\frac{n}{5}$ is at least 2) ↑

(This level of detail is needed for the assignment). □

**Example:** Let $f(n) = 3n^3 + 10n^2 + n + 6$ and $g(n) = \frac{n^3}{10}$
Show that $f = O(g)$
**Answer:** Take $n_0 = 1$ and $c = 1000$.

$$\forall n > n_0 \quad f(n) = 3n^3 + 10n^2 + n + 6 \leq 3n^3 + 10n^3 + n^3 + 6n^3 \leq 20n^3 \leq c \cdot g(n) = 1000\left(\frac{n^3}{10}\right) = 100n^3.$$

**Definition:** Let $f, g : \mathbb{N} \to \mathbb{R}^+$. Then we say $f = o(g)$ iff $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$.
equivilently, $\forall \epsilon > 0 \ \exists n_0 > 0 \ \forall n > n_0 \ \frac{f(n)}{g(n)} \leq \epsilon$. equivalently $f(n) \leq \epsilon g(n)$.

**Example:** show that $f = o(g)$ where $f(n) = \sqrt{n}$ and $g(n) = n$
**Answer:** Take any $\epsilon > 0$. set $n_0 = \frac{1}{\epsilon^2}$. Then $\forall n \geq n_0$, $f(n) = \sqrt{n} \leq (\epsilon\sqrt{n})\sqrt{n}$
Note that $\forall n > \frac{1}{\epsilon^2}$, $\sqrt{n} > \frac{1}{\epsilon}$ equivalently $\epsilon\sqrt{n} > 1$.

**Example:** True or False
(1) $\sqrt{n} = o(n^3)$ TRUE. $\frac{n}{n^3}$ as $n \to \infty$ goes to 0.
(2) $n = O(\log n)$ FALSE. ($\log n$ grows really slow)
(3) $n^{10} = O(2^n)$ TRUE.
(4) $n \log n = o(n \log(\log n))$ FALSE. (false for big O too).
(5) $n = O(n)$ but $n \neq o(n)$ TRUE.
(6) $n = O(2^{\sqrt{n}})$ TRUE (tricky one).
(7) $1000 = O(1)$ TRUE ⇒ constants are all $O(1)$.
(8) $n = O(2^{(\log n)^2})$ TRUE ⇒ $2^{(\log n)^2} = 2^{(\log n)^{\log n}} = n^{\log n}$
(9) $n = O\left(2^{\sqrt{\log n}}\right)$ FALSE ⇒ $2^{\sqrt{\log n}} = 2^{(\log n)^{\frac{1}{\sqrt{\log n}}}} = n^{\frac{1}{\sqrt{\log n}}}$ $n = 1000 \quad 1000$
(10) $a_k n^k + a_{k-1} n^{k-1} + \ldots + a_1 n + a_0 = O(nk)$ where $a_k, \ldots, a_0$ are constants.

COMP 330
NOV 19 2010

PAUL HUSSMAN

① Example: consider the language: $L = \{ "a+b=c" \mid a,b,c \text{ are binary positive integers where } a+b=c \}$

eg: $1+11=100 \in L$

Q: what is the running time of the following Turing machine with language L.

M = "on input <a+b=c>:

$O(n^2)$ { $O(n)$ → Repeat the following until all digits of a and b are crossed, and no "carry-over" digit is left:

$O(n)$ { $O(n)$ → Find the least important digit of a that is not crossed, and cross it.
$O(n)$ → Find the least important digit of b that is not crossed, and cross it.
$O(n)$ → Add these two numbers with the possible carry-over from the previous round.
$O(n)$ → Compare this number with the least significant digit of c that is not crossed, and cross it.
If the number didn't match, reject (also record possible carry-over).

If all digits of a b and c are crossed, accept."

A: Let n be the number of symbols in the input.

$= O(n) \times O(n)$

$= O(n^2)$.

*Careful → Q: Let a, b, c be numbers satisfying $a+b=c$. How many steps does the algorithm take on "$a+b=c$"? (In terms of a and b).

A: The length of the input is: $\lceil \log_2 a \rceil + \lceil \log_2 b \rceil + \lceil \log_2 (a+b) \rceil + 2$ (+, =) $= O(\log_2 a + \log_2 b)$

so the algorithm takes: $O((\log_2 a + \log_2 b)^2)$

★ THIS ALGORITHM RUNS IN QUADRATIC RUNNING TIME, NOT LOG TIME!

② Example: Q: What is the running time of the following Turing machine which decides $L = \{ \langle G,s,t \rangle \mid s \text{ \& } t \text{ are vertices in a graph } G \text{ and there's a path } (s,t) \text{ in } G \}$

M = "On input <G,s,t>

① mark the vertex s

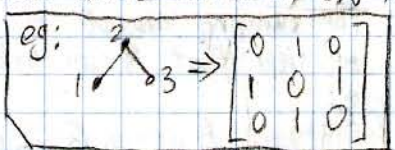
$O(n^{3/2})$ { $O(\sqrt{n})$ → ② repeat until no ne vertex is marked:

$O(n)$ → ③ Go over all edges i,j and if i was marked and j was not, then mark j

④ if t is marked, accept, otherwise reject."

Remark: $n = O(k^2)$ (n: size of input; k: # of vertices)

A: Here, the graph G has an adjacency matrix. The adjacency matrix of a graph with K vertices is a KxK zero-one matrix where the entry i,j is equal to 1 ⟺ There is an edge from i to j.

eg: 


The running time will be $O(\sqrt{n}) \times O(n) = O(n^{3/2})$

③ Example: Recall that we introduced a Turing machine that decides $\{0^n 1^n \mid n \geq 0\}$. It was moving the head back & forth and crossing one 0 for a 1 until all 0's and 1's are finished.

Q: What is the running time of this Turing machine?

A: It's $O(n)$ for each round of crossing and there's at most $O(n)$ rounds

∴ Running time is $O(n^2)$.

④ Example: Design a 2-tape Turing machine that decides $\{0^n 1^n \mid n \geq 0\}$ in $O(n)$.

Answer: M = "on input w,
scan the 0's in the beginning and copy them to the second tape.
Scan the 1's, and for each one we move the tape head on the second tape one to the left.
When scanning is finished, we will be able to say if we saw the same number of 0's as 1's."

The running time is $O(n)$

Recall:
Church Turing Thesis: An algorithm is something that can be simulated with a decider.

Question: What should we consider a FAST algorithm?

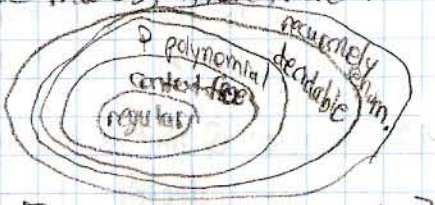
Answer: Roughly speaking, we say that an algorithm is fast if there is a positive $k$ such that running time is $O(n^k)$

---

COMP 330
NOV 22, 2010

PAUL HUSSMAN

- Intuitively, P is the set of all languages that can be decided using a fast algorithm.
- All reasonable computational models (Turing equivalent ones) which are deterministic are polynomially equivalent.
- That is, any of them can simulate the others with only a polynomial increase in the running time.
  For example, single-tape TM ≡ K-tape TMs ≡ RAM model (≡ means polynomially equivalent)
- So if there is a polynomial algorithm in any one of these models, then there is also a polynomial time algorithm in all other models.

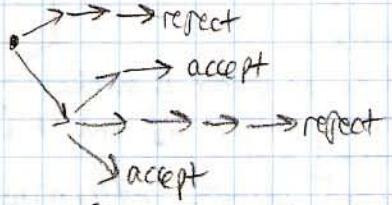



Theorem: Every context-free language belongs to P.

Definition: NON-DETERMINISTIC RUNNING TIME

Picture: Deterministic: • → • → ... → → accept/reject [# steps = running time $f(n)$]

Non-Deterministic:

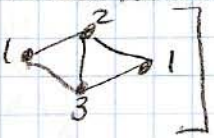



[Recall: a non-deterministic Turing machine accepts $w$ if some computation path leads to accept, rejects $w$ if all computation paths lead to reject.]

So the running time for a non-deterministic TM would be the # of steps in the longest path.

Let N be a non-deterministic TM which is a decider.
The running time of N is a function $f: \mathbb{N} \to \mathbb{N}$ such that $f(n)$ is the maximum number of steps that N can take on any computation path on any input of length $n$.

Example:
Definition: A graph G is called 3-COLORABLE if & only if it is possible to assign colors 1, 2, 3 to its vertices such that no two adjacent vertices recieve the same color.

[eg: 

]

- Consider the following language: $L = \{\langle G \rangle \mid G \text{ is a 3-colorable graph}\}$
- The non-deterministic TM N decides L, where:

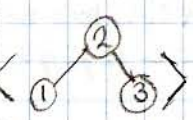
N = "On input $\langle G \rangle$ where G is a graph on $m$ vertices:

$O(m)$ { For $i = 1, 2, 3, \ldots, m$
color vertex $i$ with one of the colors 1, 2, 3 undeterministically.

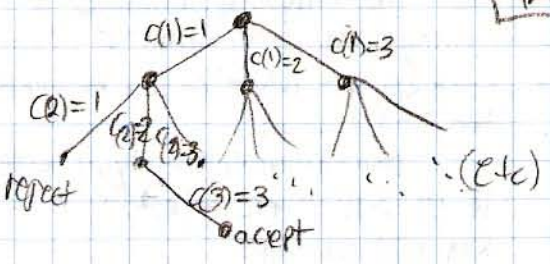
$O(m^2)$ { For every edge $(u, v)$ of G, if u and v recieve the same color, REJECT.
otherwise accept."

COMP 330
NOV 22 CONT'D

PAUL HUSSMAN

For G = 


Running time on this tree is 3 (the maximum path length).




Question: What is the running time of N in general?

Answer: - What is the input size? $\Rightarrow O(m^2)$ (# of things in $m \times m$ adjacency matrix).

The running time is $O(m^2 + m) = O(m^2) = O(n)$.

Example: Deterministic TM for the same problem

M = "on input <G> where G has m vertices

Generate each one of all $3^m$ possible colorings.

If for every edge (u,v) if u and v are colored differently, ACCEPT

If none of the $3^m$ colorings is accepted, REJECT"

Running time of M?

$= O(3^m \times m^2) = O(3^m) = O(3^{\sqrt{n}})$ (much slower).

Theorem: Let $t: \mathbb{N} \to \mathbb{N}$ satisfy $t(n) \geq n$. Then every non-deterministic single-tape TM with running time $t(n)$ can be satisfied by a deterministic single-tape TM of time $2^{O(t(n))}$.

Proof: Let $b$ be the maximum # of choices that we should make at every step.

We can use a string $y \in \{1, 2, 3, \ldots, b\}^k$ to decide which choices to pick in the first k steps. That is, in the $i^{th}$ step, we pick the choice that corresponds to the $i^{th}$ symbol in y.

Consider the following TM:

M = "on input w,

For K = 1, 2, 3, ...

$b^k \leftarrow$ generate all the $b^k$ possible strings $y \in \{1, 2, \ldots, b\}^k$

k steps { we simulate N for at most k steps deterministically using y.

If any computation leads to an accept, ACCEPT

If all computations lead to reject, REJECT."

Running time: $O(1 \times b^1 + 2b^2 + 3b^3 + \ldots + t(n)\, b^{t(n)}) = 2^{O(t(n))}$.

---

COMP 330
NOV 24 2010

PAUL HUSSMAN

Time complexity classes:

- P is the set of languages L such that $\exists k$ $\exists$ deterministic TM of running time $O(n^k)$ deciding L.
- NP : NON-DETERMINISTIC running time $O(n^k)$
- EXPTIME: DETERMINISTIC running time $O(2^{n^k})$
- NEXPTIME: NON-DETERMINISTIC running time $O(2^{n^k})$

Note: $P \subseteq NP$

$EXPTIME \subseteq NEXPTIME$

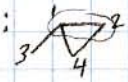
and by the thm last class: $NP \subseteq EXPTIME$.

So; $P \subseteq NP \subseteq EXPTIME \subseteq NEXPTIME$.

Question: Is it true that $P \neq NP$? (one of the most important questions in Compsci & Math).

We know: $P \neq EXPTIME$, and $NP \neq NEXPTIME$

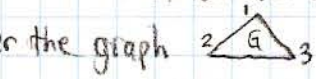
Example:

A vertex cover for a graph G is a set of vertices S such that every edge has at least one endpoint in S. Eg: {1,2} is a vertex cover for: 

Example: Show that the following language is in NP: $L = \{\langle G, k\rangle \mid G \text{ has a vertex cover of at most } k\}$

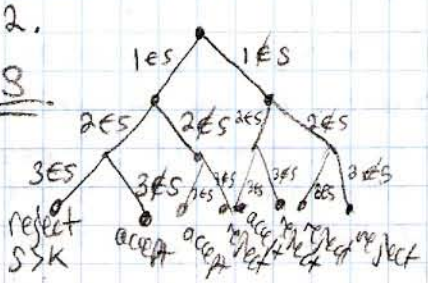
Answer: The following non-deterministic Turing machine decides $L$ in polynomial time.

N = "on input $\langle G, k\rangle$ where $G$ has $m$ vertices,
  for $i = 1, 2, \ldots, m$
    Non-deterministically, choose one of the choices $i \in S$ or $i \notin S$.
  If $|S| > k$ then reject.
  Run over all edges.
    If any of them is not covered, reject
    otherwise accept"

Consider the graph 

with $k = 2$.

Running time of this instance is 8.
(max # of steps taken)




- This non-deterministic algorithm uses non-determinism to simulate a brute-force search. That is it checks all the possible choices of picking S using its computation paths, and accepts if any of them is a proper vertex cover.

- One interpretation of the P vs NP question:
  Interpretation 1: Are there situations where brute-force search (that is trying an exponential number of possibilities one-by-one until we find a solution that satisfies all the constraints) — is essentially the best possible algorithm?

- NP via verifiers:
  In both vertex cover and 3-colorability examples, although we don't know how to solve the problems in polynomial time, if someone provides us a proper vertex cover, or a proper 3-coloring of the graph, we can easily verify it in polynomial time.
  For example, for vertex cover, given a set S:
  ① Check $|S| \leq k$
  ② Check that every edge is covered by S
  ③ If both conditions are passed, accept
  otherwise reject.

Definition: A **VERIFIER** V for the language L is a deterministic (decider) TM that always halts satisfying: $L = \{w \mid V \text{ accepts } \langle w, y\rangle \text{ for some } y\}$

The running time of V is measured with respect to the size of w
running time of V: $f(n)$ = max # steps that V takes on any $\langle w, y\rangle$ where $|w| = n$.

Theorem: A language L is in NP $\Leftrightarrow$ There is a verifier with running time $O(n^k)$ for L (for some $k \geq 0$).