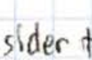
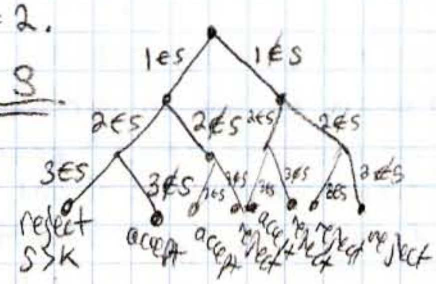
Example: Show that the following language is in NP: $L = \{\langle G, k\rangle \mid G \text{ has a vertex cover of at most } k\}$

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 $i = 1, 2, \dots, m$
  - 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 $\underline{\underline{3}}$.
(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 <u>P vs NP question</u>:

<u>Interpretation 1</u>: 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?

- <u>NP via verifiers:</u>

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 $|S| \le k$
2. Check that every edge is covered by S
3. If both conditions are passed, accept otherwise reject.

<u>Definition:</u> A $\boxed{\text{VERIFIER}}$ V for the language L is a deterministic TM that always halts (decider) satisfying: $L = \{w \mid V \text{ accepts } \langle w, y\rangle \text{ for some } y\}$ [$\boxed{\text{CERTIFICATE}}$ for w]

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 $|w| = n$.

<u>Theorem</u> A language L is in NP $\Leftrightarrow$ There is a verifier with running time $O(n^k)$ for L (for some $k \ge 0$).
(Proof next class)

COMP 330
NOV 26, 2010

PAUL HUSSMAN

<u>Example:</u> Construct a polynomial time verifier for

$L = \{\langle S, t\rangle \mid S \text{ is a finite set of numbers which contains a subset that adds up to } \underline{t}.\}$

eg: $\langle \{6, 5, 3, 2\}, 13\rangle \in L$ since $6 + 5 + 2 = 13$

PAUL HUSSMAN

Answer:

$V=$ "on input $\langle\langle S,t\rangle, y\rangle$

① Test whether y is a collection of numbers that add up to t.
② Test whether S contains every number in y
③ If both steps pass, accept
otherwise reject."

So: $\langle S,t\rangle \in L \Rightarrow$ There is some y that passes steps ① & ② $\Rightarrow V$ accepts $\langle\langle S,t\rangle, y\rangle$ for some y.
$\langle S,t\rangle \notin L \Rightarrow$ There is no y that passes steps ① & ② $\Rightarrow V$ does not accept $\langle\langle S,t\rangle, y\rangle$ for ALL y.

Proof of Theorem: A language L is in NP $\Leftrightarrow$ There is a polynomial-time verifier for L.

$\Rightarrow$ There is a non-deterministic Turing Machine N that decides L in time $f(n)=O(n^k)$.
We want to construct a polynomial-time verifier for L.
Let b be the max # of choices that we might face in any step.

$V=$ "on input $\langle w,y\rangle$ where $y \in \{1,\dots,b\}^K$ where $K=f(n)$
① Simulate N on w for at most $f(n)$ steps where at every step we make the choice according to y.
② If this leads to accept, then accept.
otherwise reject."

$w \in L \Rightarrow$ There is a branch leading to accept $\Rightarrow \exists y \mid V$ accepts $\langle w,y\rangle$
$w \notin L \Rightarrow$ Every branch leads to reject $\Rightarrow \nexists y \mid V$ accepts $\langle w,y\rangle$.
$\Rightarrow V$ is a verifier. (with polynomial runtime)

$\Leftarrow$ Suppose V is a verifier for L with running time $f(n)=O(n^k)$.
We want to construct a non-deterministic Turing machine N that decides L.

$N=$ "on input w:
Generate a string y of length at most $f(n)$ non-deterministically.
Run V on $\langle w,y\rangle$
if it accepts $\rightarrow$ accept
if it rejects $\rightarrow$ reject"

$w \notin L \Rightarrow \forall y$, V rejects $\langle w,y\rangle \Rightarrow N$ rejects w.
$w \in L \Rightarrow \exists y_0$ such that V accepts $\langle w,y\rangle \overset{*}{\Rightarrow} \exists y_0'$ of length at most $f(n)$ such that V accepts $\langle w,y_0'\rangle$
$\Rightarrow N$ accepts w.

★ Since V runs at most $f(n)$ steps, it will not be able to read anything beyond the first $f(n)$ letters of y_0, so we can replace them with blanks without affecting the performance of V.

P vs NP? Interpretation #2:

Is it harder to solve a problem by yourself than verifying someone elses solution? (probably)
(is doing an assignment harder than grading it?)
If we had a proof for P vs NP, we could easily verify it but we can't find a proof $\Rightarrow P \neq NP$

Example: Consider a finite set of axioms, and a finite set of inference rules.
Show that the following language is in NP:

$$L=\{\langle S,1^n\rangle \mid S \text{ has a proof of length at most } n \text{ using the hypothesis \& inference rules}\}$$

Answer:

$V=$ "on input $\langle\langle S,1^n\rangle, y\rangle$
Check whether y is a valid proof of length at most n
Check whether the last statement obtained in y is S
If both steps pass, accept
otherwise reject."

$\Big\} O(|S|+n)$ which is polynomial time.

P vs NP? Interpretation #3:

Are there mathematical statements that have short proofs, but it is not possible to find these proofs in short time.

COMP 330
Nov 29 2010

A5 returned avg 70% office hours wed @ 3pm see website. — PAUL HUSSMAN
Hint for A6 → see website.

Recall: Thm: Having polytime nondeterministic decider TM ≡ Having polytime verifier ≡ Being in NP.
mapping reduction: $A \leq_M B \iff \exists$ a computable f such that $\forall w$ $w \in A \iff f(w) \in B$
(Accepting B by a TM is at least as hard as accepting A)
(If B is R.E. then A is RE.)

Deciding B in polytime is at least as hard as deciding A in polytime

Definition: A function $f: \Sigma^* \to \Sigma^*$ is $\boxed{\text{polytime computable}}$ $\iff$ There is a TM M that on every input w takes at most a polynomial number of steps and halts with only $f(w)$ on the tape.

Definition: We say that a language A is $\boxed{\text{polytime reducible}}$ to B, written $A \leq_p B$, if and only if there is a polytime computable function f such that $w \in A \iff f(w) \in B$ for all $w \in \Sigma^*$

Theorem: If $A \leq_p B$ and $B \in P$ then $A \in P$

Proof: Let M be an algorithm that decides B in polytime, and f be a polytime reduction from A to B.
Consider the following algorithm:

N = "on input w
1. Compute $f(w)$
2. Run M on $f(w)$. If it accepts, accept
   if it rejects reject."

Since f is polytime computible & M has polynomial running time, N also has polynomial running time.

$w \in A \Rightarrow f(w) \in B \Rightarrow$ M accepts $f(w) \Rightarrow$ N accepts w.
$w \notin A \Rightarrow f(w) \notin B \Rightarrow$ M rejects $f(w) \Rightarrow$ N rejects w.

Proposition: If $A \leq_p B$ and $B \leq_p C \Rightarrow A \leq_p C$.

Definition: A $\boxed{\text{BOOLEAN VARIABLE}}$ is a variable that takes TRUE or FALSE values.

A $\boxed{\text{LITERAL}}$ is either a Boolean variable OR its negation. (eg: $x_1, \neg x_2, \neg x_1, \ldots$)

A $\boxed{\text{CLAUSE}}$ is an OR of literals (eg: $(x_1 \vee x_2 \vee \neg x_3)$ $(x_1 \vee x_2)$ $(\neg x_3)$)

A $\boxed{\text{CONJUNCTIVE NORMAL FORM}}$ (CNF) formula is an AND of clauses.
eg: $(x_1 \vee x_2 \vee \neg x_3) \wedge (x_2) \wedge (x_1 \vee x_3)$

Definition: A CNF formula is called $\boxed{\text{SATISFIABLE}}$ $\iff$ there is an assignment of True-False values to its variables that makes the formula TRUE.
eg: $(x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2) \wedge (x_3)$ is satisfiable.

$x_1 \leftarrow$ True
$x_2 \leftarrow$ False ← This assignment makes the formula true.
$x_3 \leftarrow$ True ← This must be true.

COMP 830
NOV 29 CONT'D

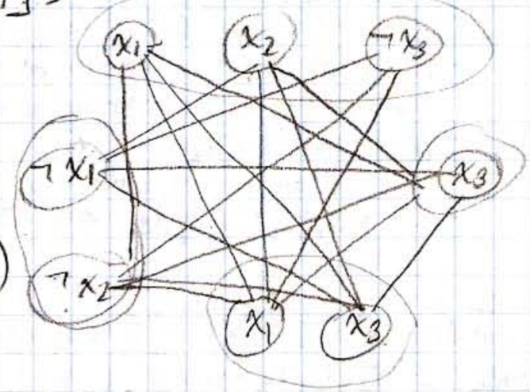
PAUL HUSSMAN

Example: Show that the language $SAT = \{\langle\phi\rangle \mid \phi$ is a satisfiable CNF$\}$ belongs to NP.
Answer: The true-false assignment that satisfies ϕ is a certificate and it can be verified in polytime.

Example: show that $(SAT \leq_p Clique)$ where $Clique = \{\langle G,K\rangle \mid G$ is a graph that contains K pairwise adjacent vertices$\}$

(ex:  $\in$ Clique)

Answer:
- Given a CNF ϕ, we will construct in polytime a pair $\langle G,K\rangle$ such that ϕ is satisfiable $\iff \langle G,K\rangle \in$ Clique.
- For every literal in every clause of ϕ, put a vertex in G
  [say $\phi = (x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2) \wedge (x_3)$] $\rightarrow$
- Put an edge between two literals in different clauses $\iff$ They are not contradictory (ie They are not x_i and $\neg x_i$)
- Claim: ϕ is satisfiable $\iff$ G contains a clique of size K.



$$\phi = (\overset{T}{x_1} \vee \overset{F}{x_2} \vee \overset{F}{\neg x_3}) \wedge (\overset{F}{\neg x_1} \vee \overset{T}{\neg x_2}) \wedge (\overset{T}{x_3}) \wedge (\overset{T}{x_1} \vee \overset{T}{x_3})$$

$x_1 \leftarrow T$
$x_2 \leftarrow F$
$x_3 \leftarrow T$

Proof of $\Rightarrow$

For every assignment that satisfies ϕ pick a true literal from every clause and they form a clique of size K in G

Proof of $\Leftarrow$

Consider a clique of size K in G.
- Let $v_1, \dots, v_K$ be the vertices of the clique. Every clause contains exactly one vertex from this clique.
- So these K vertices correspond to K literals such that no two of them are contradictory, and every class contains exactly one of them.
- Assign values to variables to make these literals true. The rest of the variables get arbitrary values
- This will satisfy ϕ. $\square$

(Important)

Definition: A language A is $\boxed{\text{NP-complete}}$ iff it satisfies both conditions:
1) $A \in NP$
2) $B \leq_p A$ for every $B \in NP$.

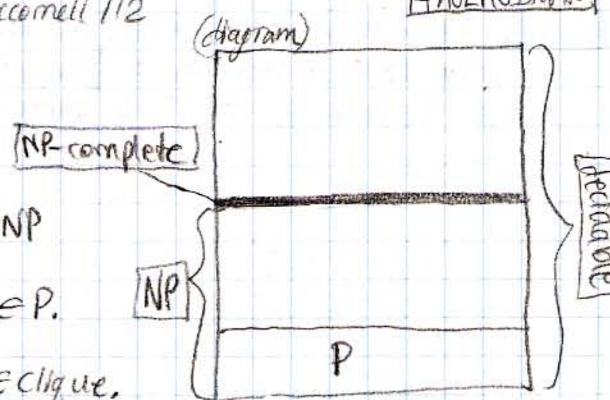
COMP 830
DEC 1, 2010

PAUL HUSSMAN

office hours for assignment 5 today 5-6 in mcconnell 112

Remark: if $P \neq NP$ then there are $A, B \in NP$ such that $A \not\leq_p B$ and $B \not\leq_p A$.

(diagram)



Theorem: (Cook 1971): SAT is NP-complete.
Corollary: If $SAT \in P \iff P = NP$
Proof: Since SAT is NP complete, any language $A \in NP$ is polytime reducible to SAT
so $SAT \in P \Rightarrow A \in P \Rightarrow$ every language in $NP \in P$.

Corollary: Clique is NP-complete
Proof: Clique is in NP indeed for every $\langle G,K\rangle \in$ Clique. The clique of size K in G is the certificate, and it can be verified in polytime.

google: "complexity zoo"

verifies second condition: We also show $SAT \leq_p$ Clique since $A \leq_p SAT$ for all $A \in NP$, we conclude that $\forall A \in NP$ we have $A \leq_p$ Clique.

Proposition: If A is NP-complete, and $B \in NP$ such that $A \leq_P B$, then B is NP-complete.

So far:

List of NP-complete Problems:

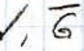
- SAT
- Clique,

Next: (easy example)

Show that $IND = \{\langle G, K\rangle \mid G$ contains K vertices with no edges between them$\}$ is NP-complete.

Answer: Trivially, IND is in NP. Indeed if $\langle G,K\rangle \in IND$ then k vertices with no edges between them is a verifiable certificate.

We prove NP completeness by reducing Clique to IND.

- Given an instance $\langle G,K\rangle$ for Clique problem, we want to construct $f(\langle G,K\rangle)$ such that $\langle G,K\rangle \in Clique \iff f(\langle G,K\rangle) \in IND$.
- Let $\overline{G}$ be the complement of G — That is every edge is converted to a non-edge, and every non-edge is converted to an edge. [eg: $G =$ , $\overline{G} =$ ]
- Note that a Clique in G is an independent set in $\overline{G}$ and vice-versa
- So $\langle G,K\rangle \in Clique \iff \langle \overline{G}, K\rangle \in IND$. [$\langle \overline{G},k\rangle$ is $f(\langle G,k\rangle)$]

Example

Show that $3SAT = \{\langle \phi\rangle \mid \phi$ is a satisfiable 3-CNF$\}$ is in NP. (3-CNF ⇒ every clause contains exactly 3 literals.)

Answer: 3SAT is trivially in NP for the same reason as SAT.

So we reduce SAT to 3-SAT. That is: given any CNF Ψ we construct a 3-CNF ϕ such that: Ψ is satisfiable $\iff \phi$ is satisfiable.

eg: $\Psi = (x_1 \vee x_2 \vee x_3 \vee x_4 \vee \overline{x_5}) \wedge (\overline{x_8})$

① $\Downarrow$

$\Psi = (x_1 \vee x_2 \vee x_3 \vee x_4 \vee \overline{x_5}) \wedge (\overline{x_8} \vee \overline{x_3} \vee \overline{x_3})$

② $\Downarrow$

$\Psi = (x_1 \vee x_2 \vee z_1) \wedge (\overline{z_1} \vee x_3 \vee z_2) \wedge (\overline{z_2} \vee x_4 \vee \overline{x_5}) \wedge (\overline{x_3} \vee \overline{x_3} \vee \overline{x_8})$.

① If there is a clause with less than 3 literals, pick one of the literals in the clause and repeat it until there's 3:

② If there are clauses with more than 3 literals, $(a_1 \vee a_2 \vee a_3 \vee \ldots \vee a_K)$, replace them with $(a_1 \vee a_2 \vee z_1) \wedge (\overline{z_1} \vee a_3 \vee z_2) \wedge (\overline{z_2} \vee a_4 \vee z_3) \wedge \ldots \wedge (\overline{z}_{K-3} \vee a_{K-1} \vee a_K)$ where $z_1, \ldots, z_{K-3}$ are new variables.

Note: that if we set all of $a_1, \ldots, a_K$ to be false, then we cannot satisfy the new 3-clauses. (the last set fails)
Also if at least one of $a_1, a_2, \ldots, a_K$ is set to true, there is a way of assigning values to $z_1, \ldots, z_{K-3}$ so that all new clauses are satisfied.

So Ψ is satisfiable $\iff \phi$ is satisfiable.

$\Rightarrow \Psi \in SAT \iff \phi \in 3SAT$ where $\phi = f(\langle\Psi\rangle)$

★Example (GOOD EXERCISE to PRACTICE POLYTIME REDUCTIONS),

Show that $VertexCover = \{\langle G,k\rangle \mid$ there are K vertices that touch every edge$\}$ 

Answer: By reducing 3SAT to VertexCover. (non-trivial).