Cpts 515. 11/20/2020

Last time:

f is $\#\mathbb{P}$ iff there is an NP-predicate Q such that $\forall x$.

$$f(x) = |\{y : Q(x, y)\}|$$

Proof. ($\Rightarrow$ easy). ($\Leftarrow$):

Suppose that $\hat{M}$ is a det. poly-time TM that accepts Q. That is, $\forall x, y$

$\hat{M}$ says yes on (x, y) in poly-time $\hat{p}(|x|+|y|)$

iff $Q(x, y)$ holds.

Now, we construct a NTM M s.t. M works on input $x \in \Sigma^*$:

input x

guess a y with $|y| \leq p(|x|)$

↑ the polynomial to bound running time of the det. TM in defining the Q.

Run the det. TM $\hat{M}$ on (x, y)

ret yes/No according to the $\hat{M}$.

Clearly, the # of accepting runs of M is exactly the # of y that makes $\hat{M}$ say yes on (x, y), for any input x. (That is $\Leftarrow$).

Relationship between #P and NP.

#P is at least harder than NP. Recall that SAT is NP-complete. Then, we can define #SAT.

Definition of #P-complete.

Consider two functions

$$f_1, f_2 : \Sigma^* \longrightarrow \mathbb{N}.$$

We say $f_1 \leq_m f_2$ if there is a poly-time computable function g s.t. $\forall x \in \Sigma^*$.

$$f_1(x) = f_2(g(x)).$$

That is, f_1 can be computed through f_2, after applying transform g.

f is #P-complete if

① $f \in \#P$

② $\forall f' \in \#P$, $f' \leq_m f$.

Given: a Boolean formula F

Count: What is the # of satisfying assignments in F?

Clearly, #SAT is to compute the following function:

$$\#SAT: F \longrightarrow \#\text{ of "solutions" to } F.$$

Notice that F is satisfiable iff $\#SAT(F) > 0$.

We see: #SAT is at least "harder" than SAT which is NP-complete.

We can have many #P problems, naturally derived from those NP-complete problems. For instance,

#HC: Given a graph G

Count: What is the # of Hamiltonian circuits in G?

Theorem. #SAT is #P-complete.

Proof. (difficult).

(1). #SAT $\in$ #P. (obvious).

// why? you need define an NP-predicate $Q(F, y)$ holds if Boolean formula F has sat. assgmt y.

(2). Consider a functn $f' \in$ #P. By definition, f' is the # of accepting runs of a TM M' runs in poly-time. To show $f' \leq_m$ #SAT, we notice that

$f'(x) =$ the # of y s.t. y is an accepting run of M' on input x.

$= Q(x, y)$

$= $ the # of y sat. $Q(x, y)$.

Notice that $Q \in \underline{P}$. Using the classic proof of Cook's reduction technique (that is, computation in NTM can be "encoded" in Boolean formula), there is a Boolean formula $\hat{Q}(\hat{x}, \hat{y})$ s.t.

$Q(x,y)$ iff $\hat{Q}(\hat{x}, \hat{y})$ and

the mapping from Q to $\hat{Q}$ is det. poly-time.

In particular, the translation from x, y to $\hat{x}, \hat{y}$ is 1-1. That is, $\forall x$,

$$|\{y : Q(x,y)\}| = |\{\hat{y} : \hat{Q}(\hat{x}, \hat{y})\}|.$$

Then, $f'(x)$ can be computed in terms of

$$\#SAT\big(\underbrace{\hat{Q}(\hat{x},\cdot)}_{\text{is a Boolean fnla on } \hat{y}.}\big) = \#SAT(g(x))$$

where $g(x) = \hat{Q}(\hat{x},\cdot)$. Where g is def. poly-time computable.

Let M be a NTM in poly-time. Then, $\forall x \in \Sigma^*$, define $f_M(x) =$ The # of accepting runs of M on x.

f_M is in $\#P$. Also, from the proof,

$$f_M \leq_m \#SAT.$$

However, the use of "$\leq_m$" fun

$$f_1 \leq_m f_2$$

makes a lot of #P-completeness proofs impossible. (Since you have cashed that fucking g in the def. $f_1 \leq_m f_2$).

New def. of $f_1 \leq_m f_2$:

$f_1 \leq_m f_2$ if there is a poly-time det. TM M that "calls" f_2 (M is called oracle TM) such that

$$f_1(x) = M^{f_2}(x).$$

Exaple:

$M^{f_2(x)}$: M runs on x while M can query to f_2 to obtain $f_2(y)$ where y is a query word generated by M.

$f_1 \leq_m f_2$ where

$f_1(x) = x^2 + x^2$

$f_2(x) = x^2$.

Obviously, computing f_1 can be:

int x

Call $f_2(x)$, set $y := f_2(x)$.

takes unit time ← Call $f_2(x)$, set $y := y + f_2(x)$.

ret y.

The query words can be very complex.

Example: $f_1(x) = 2 \cdot (x^2 - 4)^2$

$f_2(x) = x^2$

$f_1 \leq_m f_2$? yes.

$f_1(x)$ can be computed as:

input x

Call $f_2(x)$ with $y = f_2(x)$;

Compute $y-4$ as the new y,

Call $f_2(y)$ with $y := f_2(y)$,

ret $2 \cdot y$;

Clearly, we still have (under the new def of $\leq_m$), #SAT is #P-complete.

Surprise:

It appears that counting problems are closely related to NP. Actually, they are not.

A counting problem can be #P-complete but deciding the count is 0 or not can be very simple!

↑ decision problems.

Recall: CNF — conjunctive Normal form of Boolean formulas:

$(\_ \vee \_ \vee \_ \ldots) \wedge (\_ \vee \_ \vee \_ \ldots) \wedge \ldots$

each "_" is a literal

Example of CNF:

δ: $(\bar{x}_1 \vee x_2 \vee x_4) \wedge (x_1 \vee \bar{x}_2 \vee x_3 \vee x_4) \wedge \ldots$

DNF: Disjunctive Normal form.

$(\_ \wedge \_ \wedge \_ \wedge \_ \ldots) \vee (\_ \wedge \_ \wedge \_ \wedge \ldots) \vee \ldots$

Example of DNF:

φ: $(x_1 \wedge \bar{x}_2 \wedge \bar{x}_4) \vee (\bar{x}_1 \wedge x_2 \wedge \bar{x}_3 \wedge x_4) \vee \ldots$

Can you see: $\overline{\delta} = \varphi$.

taking negation

(CNF $\longleftrightarrow$ DNF).

Surprise is:

DNF satisfiability is easy, $\in \mathbb{P}$.

#DNF is #P-complete !!!