Soham Chatterjee
Email: soham.chatterjee@tifr.res.in
Course: Complexity Theory

Assignment - 1
Dept: STCS, TIFR
Date: February 25, 2025

Problem 1 Classification of problems

For each of these problems, mention (with justification) if they are in P, or not (known to be) in P, or in NP, or in coNP, or is NP-hard, or is coNP-hard etc.

(a) Factoring $= \{(1^n, 1^k) \colon n$ has a prime factor less than $k\}$

(b) Contradiction $= \{\langle\varphi\rangle \colon \langle\varphi\rangle$ encodes a formula that is false for every assignment$\}$
(For example, $\langle x_1 \land \neg x_1\rangle$ is in the language)

(c) EquivalentFormulas$= \left\{(\langle\varphi_1\rangle, \langle\varphi_2\rangle) \colon \begin{array}{c}\langle\varphi_1\rangle, \langle\varphi_2\rangle \text{ encode two formulas such that} \\ \text{for all } x \text{ we have } \varphi_1(x) = \varphi_2(x).\end{array}\right\}$

Solution:

(a) Factoring is in Psince the numbers n and k are given in unary form the algorithm can go over every number from 1 to k and test if there is a prime factor which divides n. Since division can be done in polynomial time this algorithm is polynomial time.

(b) Contradiction is in coNP since the complement of Contradiction is the set of formulas which not false for all assignment i.e. there exists at least one assignment for which formula is not false which is basically SAT and SAT is in NP. Since SAT is NP-hard Contradiction is coNP-hard. Therefore Contradiction is coNP-complete.

(c) Let $\varphi_1(x) = \varphi_2(x)$ for all x. Then consider the formula $\varphi_1 \oplus \varphi_2$ where $\oplus$ is the XOR operation and

$$x_1 \oplus x_2 = (x_1 \land \neg x_2) \lor (\neg x_1 \land x_2)$$

Then we have

$$\varphi_1(x) = \varphi_2(x) \;\forall\; x \iff (\varphi_1 \oplus \varphi_2)(x) = \text{False} \;\forall\; x$$

So we have reduced EquivalentFormulas to Contradiction. Therefore EquivalentFormulas is in coNP. Now we will show a polynomial time many-one reduction from TAUT to EquivalentFormulas. Let $\langle\varphi\rangle \in$TAUT. Then consider $(\langle\varphi\rangle, \langle 1\rangle)$ where $\langle 1\rangle$ is the boolean formula which always returns true. Then we have

$$\langle\varphi\rangle \in \text{TAUT} \iff (\langle\varphi\rangle, \langle 1\rangle) \in \text{EquivalentFormulas}$$

Therefore EquivalentFormulas is coNP-hard. So it is coNP-complete.

■

Problem 2 Properties of reductions

We'll use $L_1 \le_m^{\text{poly}} L_2$ to denote that there is a polynomial-time many-one reduction from L_1 to L_2. Answer each of the questions with True/False with brief justifications

(a) If $L_1 \le_m^{\text{poly}} L_2$, then $\overline{L_2} \le_m^{\text{poly}} \overline{L_1}$

(b) If $L_1 \le_m^{\text{poly}} L_2$, then $\overline{L_1} \le_m^{\text{poly}} \overline{L_2}$.

(c) If L is NP-hard, then $\overline{L}$ is coNP-hard

(d) If $L_1 \le_m^{\text{lin}} L_2$ and $L_2 \le_m^{\text{lin}} L_3$. (Here, $\le_m^{\text{lin}}$ refers to linear-time many-one reductions).

(e) If $L_1 \le_m^{\text{quad}} L_2$ and $L_2 \le_m^{\text{quad}} L_3$, then $L_1 \le_m^{\text{quad}} L_3$. (here $\le_m^{\text{quad}}$ refers to quadratic time many-one reductions.)

Solution:

(a) False. For example take L_1 to be any language in P and L_2 to be any NP-complete language (SAT). Then $\overline{L_1} \in \mathsf{P}$ and $\overline{L_2}$ is coNP-complete (We proved in Question 1.b). So if $\overline{L_2} \le_m^{\text{poly}} \overline{L_1}$ that means $\mathsf{P} = \text{coNP}$ which is unlikely to happen.

(b) True. Since $L_1 \le_m^{\text{poly}} L_2$ there exists a polynomial time computable function $f : \Sigma^* \to \Sigma^*$ such that $x \in L_1 \iff f(x) \in L_2$. Hence we can also write this as $x \notin L_1 \iff f(x) \notin L_2$ which is equivalent to $x \in \overline{L_1} \iff f(x) \in \overline{L_2}$. Therefore the same function f works in reducing $\overline{L_1}$ to $\overline{L_2}$. Therefore $\overline{L_1} \le_m^{\text{poly}} \overline{L_2}$

(c) True. Let $L' \in$ coNP. Then $\overline{L'} \in$ NP. Since L is NP-hard we have $\overline{L'} \le_m^{\text{poly}} L$. By using the above result we have $L' \le_m^{\text{poly}} \overline{L}$. Since L' is any arbitrary language of coNP, L is coNP-hard.

(d) True. Since $L_1 \le_m^{\text{lin}} L_2$ there exists a linear-time computable function $f : \Sigma^* \to \Sigma^*$ such that $x \in L_1 \iff f(x) \in L_2$. Since $L_2 \le_m^{\text{lin}} L_3$ there exists a linear-time computable function $g : \Sigma^* \to \Sigma^*$ such that $x \in L_2 \iff f(x) \in L_3$. Hence

$$x \in L_1 \iff g \circ f(x) \in L_3$$

Now since f and g are both linear time computable function their composition is also a linear time computable function. Therefore $L_1 \le_m^{\text{lin}} L_3$.

(e) False. Since $L_1 \le_m^{\text{quad}} L_2$ there exists a quadratic-time computable function $f : \Sigma^* \to \Sigma^*$ such that $x \in L_1 \iff f(x) \in L_2$. Since $L_2 \le_m^{\text{quad}} L_3$ there exists a quadratic-time computable function $g : \Sigma^* \to \Sigma^*$ such that $x \in L_2 \iff f(x) \in L_3$. Since composition of two quadratic functions not necessarily a quadratic function. I.e for example if $|f(x)| = |x|^2$ and $|g(x)| = |x|^2$ then $|g \circ f(x)| = |x|^4$ which is not quadratic with respect to $|x|$. Hence there may not be any quadratic-time many-one reduction from L_1 ot L_3.

■

Problem 3 Verifier perspective of NP

In class, we gave a *machine-based* definition of the class NP. Often times, the following alternate definition is used..

> A language $L \subseteq \Sigma^*$ is said to be in NP if and only if there is a language $V_L \in \mathsf{P}$, and constants c, n_0 such that for any $x \in \Sigma^*$ with $|x| > n_0$, we have $x \in \mathrm{L}$ if and only if there is a string $w \in \Sigma^*$ with $|w| \leq |x|^c$ such that $(x, w) \in V_L$

Formally prove that the above definition is equivalent to the definition we saw in class. (That is, any language that satisfies the above property is computable by a non-deterministic TM running in poly(n) time, and vice-versa.)

Solution: Suppose N is a nondeterministic turing machine N and $L := L(N)$. Then there exists constants $n_0, c \in \mathbb{N}$ such that $x \in L \iff N(x) = \text{Accept in } |x|^c$ time for all $|x| > n_0$. Now N has two transition functions Γ_0 and Γ_1. N accepts x if and only if there is a sequence of use of Γ_0 and Γ_1 which accepts x. Let $w \in \{0,1\}^{p(|x|)}$ where i^{th} bit of w denotes that at i^{th} step in N the Γ_i transition function was used. Then consider the deterministic turing machine M which on the input (x, w) does a deterministic simulation of N on x following w applying the transition functions Γ_0 and Γ_1 accordingly.

Consider the language

$$V_L = \{(x, w) \mid x \in L,\ w \in \{0,1\}^c,\ N \text{ accepts } x \text{ following } w\}$$

The way constructed M we have

$$(x, w) \in V_L \iff M(x, w) = \text{Accept}$$

Therefore $V_L \in \mathsf{P}$.

Now

$$\begin{aligned} x \in L &\iff \exists\ w \in \{0,1\}^c \text{ such that } N \text{ accepts } x \text{ following transition } w \\ &\iff M \text{ accepts } (x, w) \\ &\iff (x, w) \in V_L \end{aligned}$$

Now suppose $L \subseteq \Sigma^*$ such that there is a language $V_L \in \mathsf{P}$, and constants c, n_0 such that for any $x \in \Sigma^*$ with $|x| > n_0$, we have $x \in \mathrm{L}$ if and only if there is a string $w \in \Sigma^*$ with $|w| \leq |x|^c$ such that $(x, w) \in V_L$. Since $V_L \in \mathsf{P}$ there exists a polynomial time turing machine M which decides V_L. Then there exists a polynomial time deterministic turing machine M deciding V_L.

Consider the nondeterministic turing machine N with two transition functions Γ_0 and Γ_1. For the first c steps if Γ_0 is used then it writes 0 in the tape and if Γ_1 is used it writes 1 in the tape. Suppose in the first c steps the binary string written on the tape in w. After the first c steps both the transition functions becomes equal and they run M on the input (x, w). If x is in L then there exists $w \in \{0,1\}^c$ such that $(x, w) \in V_L$. So N in the first c steps guesses w and then runs (x, w). Hence if

$$\begin{aligned} x \in L &\iff \exists\ w \in \{0,1\}^c \text{ such that } (x, w) \in V_L \\ &\iff N(x) = \text{Accept} \end{aligned}$$

Therefore the two definitions of NP are equivalent to each other.

■

Problem 4 Operations on languages

(a) If L_1, L_2 are two languages in NP, show that the languages $L_1 \cap L_2$ and $L_1 \cup L_2$ are in NP as well.

(b) For any three languages L_1, L_2, L_3,

$$\text{Maj}(L_1, L_2, L_3) = \{x \colon x \text{ is in at least two of the } L_i\text{'s}\}.$$

Show that, if $L_1, L_2, L_3 \in \text{NP}$, then the language $\text{Maj}(L_1, L_2, L_3)$ is also in NP.

(c) For two languages L_1, L_2, let $L_1 \oplus L_2 = \{x \in \Sigma^* \mid x \text{ is in exactly one of } L_1, L_2\}$. If $L_1, L_2 \in \text{NP} \cap \text{coNP}$, show that $L_1 \oplus L_2 \in \text{NP} \cap \text{coNP}$ as well.

Solution:

(a) $L_1, L_2 \in \text{NP}$. Hence there exists $V_{L_1}, V_{L_2} \in \text{P}$ and constants c_1, n_0^1 and c_2, n_0^2 such that for any $x_1, x_2 \in \Sigma^*$ with $|x_1| > n_0^1$, $|x_2| > n_0^2$ then $x_1 \in L_1 \iff \exists\ w_1 \in \Sigma^*$ with $|w_1| \le |x_1|^{c_1}$ such that $(x_1, w_1) \in V_{L_1}$ and $x_2 \in L_2 \iff \exists\ w_2 \in \Sigma^*$ with $|w_2| \le |x_2|^{c_2}$ such that $(x_2, w_2) \in V_{L_2}$.

Then take $c = c_1 + c_2 + 1$ and $n_0 = \max\{n_0^1, n_0^2\}$. Then we have for any $x \in \Sigma^*$ with $|x| > n_0$, $x \in L_1 \cap L_2 \iff \exists\ w_1\#w_2 \in \Sigma^*$ with $|w_1\#w_2| \le |x|^c$ such that $(x, w_1) \in V_{L_1}$ and $(x, w_2) \in V_{L_2}$. So construct the language $V_{L_1 \cap L_2} = (x, w_1\#w_2)$. where

$$(x, w_1\#w_2) \in V_{L_1 \cap L_2} \iff (x, w_1) \in V_{L_1}, (x, w_2) \in V_{L_2}$$

Since $V_{L_1}, V_{L_2} \in \text{P}$ there exists polynomial time turing machines M_1, M_2 such that $(x, w) \in V_{L_1} \iff M_1(x, w) = $Accept and $(x, w) \in V_{L_2} \iff M_2(x, w) = $Accept. So the turing machine M for $V_{L_1 \cap L_2}$ on input $(x, w_1\#w_2)$ runs both M_1 on (x, w_1) and M_2 on (x, w_2) in parallel and accepts if both of them accepts.So M is a polynomial time turing machine since both M_1 and M_2 are polynomial time turing machine and $L(M) = V_{L_1 \cap L_2}$. Therefore $V_{L_1 \cap L_2} \in \text{P}$. Therefore $L_1 \cap L_2$ is in NP.

Take $c = \max\{c_1, c_2\}$ and $n_0 = \max n_0^1, n_0^1$. If $x \in L_1 \cup L_2$ then either $x \in L_1$ or $x \in L_2$. Then there exists $w \in \Sigma^*$ with $|w| \le |x|^c$ such that either $(x, w) \in V_{L_1}$ or $(x, w) \in V_{L_2}$. So $(x, w) \in V_{L_1 \cap L_2}$. Hence $x \in L_1 \cap L_2 \iff \exists\ w \in \Sigma^*$ with $|w| \le |x|^c$ such that $(x, w) \in V_{L_1 \cup L_2}$. Let M_1 and M_2 be the polynomial time turing machines for V_{L_1} and V_{L_2} respectively. Then consider the turing machine M which on input (x, w) runs both M_1 and M_2 on (x, w) in parallel and accepts in one of them accepts otherwise rejects. So M is a polynomial time turing machine since both M_1 and M_2 are polynomial time turing machine and $L(M) = V_{L_1 \cup L_2}$. So $V_{L_1 \cup L_2} \in \text{P}$. Therefore $L_1 \cup L_2 \in \text{NP}$.

(b) Given $\text{Maj}(L_1, L_2, L_3) = \{x \colon x \text{ is in at least two of the } L_i\text{'s}\}$. Hence $x \in \text{Maj}(L_1, L_2, L_3) \iff x \in L_1 \cap L_2$ or $x \in L_2 \cap L_3$ or $x \in L_3 \cap L_1$. Therefore

$$\text{Maj}(L_1, L_2, L_3) = (L_1 \cap L_2) \cup (L_2 \cap L_3) \cup (L_3 \cap L_1)$$

Since $L_1, L_2, L_3 \in \text{NP}$. Then by results above $L_1 \cap L_2, L_2 \cap L_3, L_3 \cap L_1 \in \text{NP}$ and again by results above we have $(L_1 \cap L_2) \cup (L_2 \cap L_3) \in \text{NP}$ and henceforth $(L_1 \cap L_2) \cup (L_2 \cap L_3) \cup (L_3 \cap L_1) \in \text{NP}$.

(c) Since $L_1, L_2 \in \text{NP} \cap \text{coNP}$ we have $\overline{L_1}, \overline{L_2} \in \text{NP} \cap \text{coNP}$. Now $L_1 \oplus L_2 = \{x \in \Sigma^* \mid x \text{ is in exactly one of } L_1, L_2\}$. Hence $x \in L_1 \oplus L_2 \iff x \in L_1, x \notin L_2$ or $x \in L_2, x \notin L_1$. Hence we have

$$L_1 \oplus L_2 = (L_1 \cap \overline{L_2}) \cup (\overline{L_2} \cap L_2)$$

Since $L_1, \overline{L_1}, L_2, \overline{L_2} \in \text{NP}$ we have $L_1 \cap \overline{L_2} \in \text{NP}$ and $\overline{L_1} \cap L_2 \in \text{NP}$. And therefore $(L_1 \cap \overline{L_2}) \cup (\overline{L_2} \cap L_2) \in \text{NP}$. Hence $L_1 \oplus L_2 \in \text{NP}$.

∎

Problem 5 Not-all-equal-SAT

The ‘not-all-equal’ function $\mathrm{NAE}(x_1,\dots,x_k)$ is the Boolean function that is true when not all the values of $x_1,\dots,x_k$ are equal (that is, it is false only on the inputs $000\cdots0$ and $111\cdots1$).
NAE-SAT is the constraint satisfaction problem where each constraint is the above NAE-function. An instance of NAE-SAT is satisfiable if there is an assignment to the variables that satisfy all the constraints. (Similar to how CNF-SAT had the constants as the ‘OR’ function.)

(a) Consider the following purported reduction from CNF-SAT to NAE-SAT:

> Consider a new varibale z. For each clause of the form, $(x_i \vee x_j \vee x_k)$ in the CNF, add the constraint $\mathrm{NAE}(x_i,x_j,x_k,z)$ to the NAE instance.

Prove that this is a legitimate reduction from CNF-SAT to NAE-SAT

(b) If we begin with an instance of 3CNF-SAT, then we end up with an instance of 4NAE-SAT in the above reduction. Give a polynomial-time many-one reduction from 4NAE-SAT to 3NAE-SAT.

(c) What is 2NAE-SAT problem? Do you know it by a different name? Is it in P?

Solution:

(a) Suppose the NAE-SAT instance obtained from φ is C. We will show

$$\varphi \in \text{CNF-SAT} \iff C \in \text{NAE-SAT}$$

Let φ is a CNF-SAT formula. Let there exists a satisfying assignment y such that $\varphi(y)$ =True. Let $x_i \vee x_j \vee x_k$ is a clause of φ. Since y satisfies this clause at least one of the variables in x_i, x_j, x_k is set to be true in y. Hence setting z to be False in $\mathrm{NAE}(x_i,x_j,x_k,z)$ is also satisfied. Since all of the clauses are satisfied by y, in each clause at least one of the variables is set to be True by y. Hence setting z to be False satisfies that not all assignments are equal for each NAE constraint corresponding to the clauses. Therefore the corresponding formula NAE-SAT$(x_1,\dots,x_k,z)$ is satisfiable.

Now suppose the constructed NAE-SAT instance has a satisfiable assignment. Hence in each clause there is at least one variable set to True and at least one variable set to False. Hence if we flip the assignment of each variable then still this is true that in each clause at least one variable is set to True and at least variable set to False. Hence flipping the assignments for all variable still gives a satisfying assignment. Hence if the variable is z is set to True we flip the assignment otherwise we dont do anything. From the assignment we remove the assignment corresponding to the varible z. This is a satisfying assignment for the CNF-SAT formula φ since for each clause there is at least one variable set to True as z is set to False. Hence with this assignment each clause of φ is satisfied. Therefore φ is satisfiable.

Therefore the given reduction is a legitimate polynomial time many-one reduction from CNF-SAT to NAE-SAT

(b) Let C is a 4NAE-SAT instance. Let $\mathrm{NAE}(w,x,y,z)$ is a constraint of C. Then suppose a is a new variable. Then we remove the constraint $\mathrm{NAE}(w,x,y,z)$ from C and add the following two constraints

$$\mathrm{NAE}(w,x,a) \qquad \mathrm{NAE}(\neg a,y,z)$$

For every constraint of C we introduce a new variable and do this and finally we get a 3NAE-SAT instance C' from C. Now we will show

$$C \text{ is satisfiable} \iff C' \text{ is satisfiable}$$

Since in each clause a new variable is used it is equivalent to show that

$$\mathrm{NAE}(w,x,y,z) \text{ is satisfiable} \iff \mathrm{NAE}(w,x,a) \wedge \mathrm{NAE}(\neg a,y,z) \text{ is satisfiable}$$

Let NAE(w, x, y, z) is satisfiable. Then there is at least one variable set to True and at least one variable set to False. Now there are two possible case. One of th pairs $\{\{w, x\}, \{y, z\}\}$ set to same value. WLOG suppose w, x set to same value. Then y, z are set to different values. Hence in NAE$(\neg a, y, z)$, a can be set to any value but in NAE(w, x, a) since both w, x are set to same value a has to be set the other. This way both NAE(w, x, a) and NAE$(\neg a, y, z)$ are satisfied. If all the pairs in $\{\{w, x\}, \{y, z\}\}$ are set to different values then w, x are set to different values and y, z are set to different values. Hence for any value of a both NAE(w, x, a) and NAE$(\neg a, y, z)$ are satisfied. Therefore NAE$(w, x, a) \wedge$ NAE$(\neg a, y, z)$ is satisfied.

Now let NAE$(w, x, a) \wedge$ NAE$(\neg a, y, z)$ is satisfied. WLOG suppose a is set to be True. Then w, x are set to different values. Therefore in w, x, y, z there is at least one variable set to True and one variable set to False. Therefore NAE(w, x, y, z) is satisfied.

Therefore this gives a polynomial-time many-one reduction from 4NAE-SAT to 3NAE-SAT.

(c) Consider the problem 2Colourable:= $\{\langle G\rangle \mid G$ is a 2 colourable graph$\}$. We will show 2NAE-SAT is reducible to 2Colourable. Let C be a 2NAE-SAT instance with n variables. Now consider the graph $G = (V, E)$ with the vertex set is of size $2n$ with where $V = \{x_i, \neg x_i \mid \forall\ i \in [n]\}$ Now $\forall\ i \in [n]$, $(x_i, \neg x_i) \in E$. For all NAE(x_i, x_j) constraint in C put the edge $(x_i, x_j) \in E$ where x_i, x_j are literals. Then if C is satisfiable then we colour the literals which are set to True to Red colour and the literals which are set to False to Blue colour. For edges of the form $(x_i, \neg x_i)$ if one is True then other is False so these edges satisfies the 2Colourable property. For edges of the form (x_i, x_j) the constraint NAE(x_i, x_j) is satisfied. So one is set to True and other is set to False. Therefore the edge also satisfies the 2Colourable property. Therefore the graph G is indeed 2Colourable.

Now suppose G is 2Colourable. Then for every variable x_i if the literal x_i is coloured Red then we set x_i variable to be True otherwise False. Then for each constraint in C the corresponding edge satisfies the 2Colourable property. Therefore one of the variable is set to True and other variable is set to False. Hence the constraint is satisfied. Therefore each constraint of C is satisfied. Therefore C is satisfied.

This reduction is polynomial time many-one reduction. Therefore 2NAE-SAT$\leq_m^{\text{poly}}$ 2Colourable. Now we can check if a graph is 2Colourable by taking any vertex and start Breadth-First-Search from that vertex and in each layer we colour the vertex in that layer with the other colour compared to the previous layer. If a vertex appears in both odd layers and even layers then the graph is no 2Colourable otherwise it is 2Colourable. Hence 2Colourable is in P. Therefore 2NAE-SAT is in P.

■

Problem 6 Equations that have integer solutions

Hat-tip: This problem is from Ryan O'Donnel's complexity course

(a) Consider the following language

$$L = \left\{ (a,b) \colon a,b \in \mathbb{Z}, \begin{array}{c} \text{there exists integers } x,y \\ \text{such that } x^2 + ay + b = 0 \end{array} \right\}$$

Here the inputs a, b are provided in binary.

Prove that $L \in \mathrm{NP}$.

(b) Consider the following language

$$L = \left\{ (a,b,c) \colon a,b,c \in \mathbb{Z}, \begin{array}{c} \text{there exists integers } x,y,z \\ \text{such that } x^3 + ay^2 + bz + c = 0 \end{array} \right\}$$

I don't know if this problem is in NP. Why do you think that is so?

Solution:

(a) $x^2 + ay + b = 0$ where $a, b \in \mathbb{Z}$. Now if $a = 0$ then $-b$ has to be a perfect square. Suppose $a \neq 0$. WLOG we can assume $a > 0$ since otherwise we will change sign of y. Then

$$x^2 + ay + b = 0 \iff y = -\frac{x^2 + b}{a} \implies x^2 + b \equiv 0 \bmod a$$

Suppose $\exists\ 0 \leq c < a$ such that $-b \equiv c^2 \bmod a \iff -b = c^2 + ak$ for some integer k. Then take $y = k$ then we have $ay + b = ak + b = ak - c^2 - ak = -c^2$. Hence $x = c$. So if $x^2 + ay + b = 0$ as an integral solution for (x, y) then $\exists c \in \{0, \ldots, |a|\}$ such that $c^2 + b \equiv 0 \bmod |a|$. Since a is given in binary, the binary expression of c takes at most the same number of bits as a needed. Therefore the non-deterministic machine will guess c and then verify if $c^2 + b \equiv 0 \bmod |a|$ and accept if its true.

(b) Let $b \neq 0$. WLOG we can assume $b > 0$ Then

$$x^3 + ay^2 + bz + c = 0 \implies x^3 + ay^3 + c \equiv 0 \bmod b$$

If $x^3 + ay^3 + c \equiv 0 \bmod b$ has a solution then we can take $z = -\frac{x^3+ay^3+c}{b}$. So if there is a solution for (x, y, z) for (a, b, c) then there exists solution for x, y, z where $x, y \in [b]$ and z is polynoimally bounded by x, y. Such x, y takes at most same bit complexity as b. Therefore if $b \neq 0$ then there are solutions which is polynomially bounded by the bit complexity of b.

Let $b = 0$. There may be instances of $(a, 0, c)$ for which integral solutions for x, y, z can absurdly larger compared to a, c. In that case we don't know how to give a polynomially bounded certificate in terms of the bit complexity of a, b, c to calculate the integral solutions and verify. That is why this problem may not be in NP

■

Problem 7 Block-respecting TMs

Let $B : \mathbb{N} \to \mathbb{N}$ be "reassonable" function (increasing, time-constructible, yada,yada). A deterministic Turing machine M is said to be B-*block-respecting* if it has the following: property:

> Consider a length n input x. The tape(s) of the Turing machine are split into contiguous blocks of length $B(n)$ each. Each head of the Turing Maxhine crosses a block-boundary only at time-steps that are integer multiples of $B(n)$.
> (That is, within each block of $B(n)$ time steps, each head operates only within a particular block)

Given a deterministic Turing machine M that runs in time $T(n)$, and a "reasonable" function $B : \mathbb{N} \to \mathbb{N}$, construct a deterministic B-block-respecting Truing machine M' for the same language while ensuring the running time of M' is $O(T(n))$.

[Hint: Suppose the machine M uses k tapes, you could perhaps make use of the above picture and use $3k$ tapes.]

Solution: Let the input length is n. We will use B instead of $B(n)$ from now on. Let M uses k tapes. We will create a B-block-respecting turing machine M' which has $3k$ tapes and uses 3 tapes for each tape of M. M' has 3 copies of content of each tape of M. First we divide the content of each tape of M' and M into blocks of B.

Let L_i^j denotes the j^{th} block in certain i^{th} tape of M. Let at some time the head inside the block L_j^i. Then consider the corresponding 3 tapes in M'. The head positions are as follows:

- In the first tape at the end of L_{j-1}^i
- In the second tape the head position is same as the head position in i^{th} tape in M in L_j^i.
- In the third tape at the start of L_{j+1}^i

When the head in L_j^i in i^{th} tape of M is working the head in the second tape of M' in L_j^i works same. Now at some point if the head in i^{th} tape of M goes to L_{j+1}^i but within the time period B then the the head of the second tape remains at the end of L_j^i and the now the head at the third tape at start of L_{j+1}^i starts to work similar to as the head in i^{th} tape of M. And within the same B time period if the head in i^{th} tape of M comes back to L_j^i then the head at the third tape stops at the start of L_{j+1}^i and the head at the end of L_j^i in second tape starts working. Similarly if the head in i^{th} tape of M goes to L_{j-1}^i then the head in second tape stops at the start of L_j^i and the head in first tape at the end of L_{j-1}^i starts working. If within the same B time period comes back to L_j^i in i^{th} tape of M the head at the first tape stops at the end of L_{j-1}^i and the head at the second tape at the start of L_j^i starts working.

The above happens till the B time period. Once the B time period ends then let the head in i^{th} tape of M is in somewhere in L_j^i then the heads of M' in the corresponding 3 tapes are at the end of L_{j-1}^i in first tape, same position as the head in i^{th} tape of M in L_j^i in second tape of M' in L_j^i and at the start of L_{j+1}^i in the third tape. Now head at the second tape goes to the start of L_j^i and then spends the rest of the B time period there. During this time the head at the third tape at the start of L_{j+1}^i comes to the start of the L_{j+1}^i spends the rest of the B time period. Then both the heads of second tape and third tape goes to start of L_{j-1}^i and then copies the content of L_{j-1}^i from first step to the second tape and third tape. This takes another B time. Then heads of all 3 tapes goes to the end of the L_{j-1}^i. Then goes to the start of L_j^i and starts to copy the content of L_j^i of second tape in L_j^i of first tape and third tape and goes to end of L_j^i. This takes another B time. Then all three tapes goes to the start of L_{j+1}^i and then copies the content of L_{j+1}^i from the third tape to the first and second tape. This takes B time. Now all the three tapes comes back to the start of L_{j+1}^i. Then the head of second tape and first tape goes to end of L_j^i. Then the heads in the second tape goes to the same position as head in i^{th} tape of M the head and during that time the head of the first tape goes to the end of L_j^i. And then it goes to start of L_{j-1}^i and then spends B time there. Hence in total this whole process takes at most $10B$ time.

This turing machine simulates each B time period of M in $11B$ time. Hence this turing machine takes $\frac{T(n)}{B} \times 11B = O(T(n))$ time. ■

Problem 8 Improving the time-hierarchy theorem

For this problem, you may assume that any 'reasonable-looking' function is time-constructible. And whenever you see functions like $n^2 \log^{3/4}(n)$, assume that there is an implicit ceiling to make sure this is an integer etc. (Basically don't worry about technicalities!) In class we proved that the deterministic time hierarchy theorem that stated the following:
Suppose $t_1, t_2 : \mathbb{N} \to \mathbb{N}$ are non-decreasing time-constructible functions with $t_1(n), t_2(n) \geq n$. If we have $t_1(n) \log t_1(n) = o(t_2(n))$, then we have $\mathsf{DTIME}(t_1) \subsetneq \mathsf{DTIME}(t_2)$.

(a) Let $t_1, t_2, f : \mathbb{N} \to \mathbb{N}$ be time-constructible non-decreasing functions that satisfy $t_1(n), t_2(n), f(n) \geq n$. Show that $\mathsf{DTIME}(t_1(n)) = \mathsf{DTIME}(t_2(n))$ implies

$$\mathsf{DTIME}(t_1(f(n))) = \mathsf{DTIME}(t_2(f(n)))$$

[Hint: Padding]

(b) Show that $\mathsf{DTIME}\left(n^2\right) \subsetneq \mathsf{DTIME}\left(n^2 \log^{3/4}(n)\right)$.

[Hint: You may have to use the above part multiple times]

(c) Extend this to show that for any rational number a, ε satisfying $a > 1$ and $0 < \varepsilon < 1$, we have

$$\mathsf{DTIME}(n^a) \subsetneq \mathsf{DTIME}(n^a (\log n)^{\varepsilon})$$

Solution:

(a) Let $L \in \mathsf{DTIME}(t_1(f(n)))$. Then there exists a deterministic Turing machine M which runs in $O(t_1(f(n)))$ time decides L. Now consider the language L_{pad} where

$$L_{pad} := \left\{\tilde{x} = \langle x, 1^{f(|x|)}\rangle \mid x \in L\right\}$$

Consider the turing machine M^{pad}

M^{pad} : Input $\tilde{x}$
Check if $\tilde{x}$ of the form $\langle x, 1^m\rangle$ where $m = f(|x|)$. Reject otherwise
Run M on the input x.
Accepts if M accepts otherwise reject.

To check the input of the form $\langle x, 1^{f(|x|)-|x|}\rangle$ it takes $O(f(|x|))$. Time Then it runs M which takes $O(t_1(f(|x|))$ time. Therefore $L_{pad} \in \mathsf{DTIME}(t_1(n))$. Since $\mathsf{DTIME}(t_1(n)) = \mathsf{DTIME}(t_2(n))$ there exists a turing machine M' which runs in $O(t_2(n))$ time and decides L_{pad}. So consider the turing machine M''

M'' : Input x
Build $\tilde{x} = \langle x, 1^{f(|x|)}\rangle$
Run M' on $\tilde{x}$.
Accept if M' accepts otherwise reject

Certainly M'' runs in $O(t_2(f(n)))$ time. Therefore $L'' \in \mathsf{DTIME}(t_2(f(n)))$. Hence

$$\mathsf{DTIME}(t_1(f(n))) \subseteq \mathsf{DTIME}(t_2(f(n)))$$

With similarly argument by switching t_1 and t_2 we can show $\mathsf{DTIME}(t_1(f(n))) \supseteq \mathsf{DTIME}(t_2(f(n)))$. Therefore we have $\mathsf{DTIME}(t_1(f(n))) = \mathsf{DTIME}(t_2(f(n)))$

(b) Suppose $\mathsf{DTIME}\left(n^2\right) = \mathsf{DTIME}\left(n^2 \log^{\frac{3}{4}}(n)\right)$. Let $f(n) = n \log^{\frac{3}{8}} n$. Then by using the previous part we have that

$$\mathsf{DTIME}\left(f^2(n)\right) = \mathsf{DTIME}\left(f^2(n) \log^{\frac{3}{4}}(f(n))\right)$$

Now

$$f^2(n) = \left(n \log^{\frac{3}{8}} n\right)^2 = n^2 \log^{\frac{3}{4}} n$$

And

$$\log(f(n)) = \log\left(n \log^{\frac{3}{8}} n\right) = \log n + \log\left(\frac{3}{8}\log n\right) = \log n + \log\frac{3}{8} + \log\log n = \Theta(\log n)$$

Hence $f^2(n)\log^{\frac{3}{4}}(f(n)) = \Theta\left(n^2 \log^{\frac{3}{4}} n \log^{\frac{3}{4}} n\right) = \Theta(n^2 \log^{\frac{3}{2}} n)$. Therefore we have

$$\mathsf{DTIME}(n^2) = \mathsf{DTIME}(n^2 \log^{\frac{3}{4}} n) = \mathsf{DTIME}(n^2 \log^{\frac{3}{2}} n)$$

But $\mathsf{DTIME}(n^2) \subsetneq \mathsf{DTIME}(n^2 \log^{\frac{3}{2}} n)$ by time hierarchy theorem. Hence contradiction $\lightning$ Therefore $\mathsf{DTIME}\left(n^2\right) \subsetneq \mathsf{DTIME}\left(n^2 \log^{\frac{3}{4}}(n)\right)$.

(c) Let $N = \left\lceil \log\frac{1}{\epsilon}\right\rceil + 1$. Then $2^N \cdot \epsilon > 1$. Now suppose $\mathsf{DTIME}(n^a) = \mathsf{DTIME}(n^a \log^{\epsilon} n)$. Let $f_1(n) = n\log^{\frac{\epsilon}{a}} n$. Then we have $\mathsf{DTIME}(f_1^a(n)) = \mathsf{DTIME}(f^a(n)\log^{\epsilon}(f_1(n)))$. Now $f_1^a(n) = n^a \log^{\epsilon} n$ and

$$\log\left(n \log^{\frac{\epsilon}{a}} n\right) = \log n + \log\frac{\epsilon}{a} + \log\log n = \Theta(\log n)$$

Therefore $\mathsf{DTIME}(f_1^a(n)\log^{\epsilon}(f_1(n))) = \mathsf{DTIME}(n^a \log^{\epsilon} n \log^{\epsilon} n) = \mathsf{DTIME}(n^a \log^{2\epsilon} n)$. Hence

$$\mathsf{DTIME}(n^a) = \mathsf{DTIME}(n^a \log^{2\epsilon} n)$$

Now choose $f_2(n) = n\log^{\frac{2\epsilon}{a}} n$. Then by similar calculation we have

$$\mathsf{DTIME}(n^a) = \mathsf{DTIME}(n^a \log^{2\epsilon} n) = \mathsf{DTIME}(n^a \log^{4\epsilon} n)$$

Continuing like this at k^{th} step we have

$$\mathsf{DTIME}(n^a) = \mathsf{DTIME}(n^a \log^{2^{k-1}\epsilon} n) = \mathsf{DTIME}(n^a \log^{2^k\epsilon} n) \implies \mathsf{DTIME}(n^a) = \mathsf{DTIME}(n^a \log^{2^k\epsilon} n)$$

the function $f_k(n) = n\log^{\frac{2^k\epsilon}{a}} n$. Then we get

$$\mathsf{DTIME}(n^a) = \mathsf{DTIME}(n^a \log^{2^k\epsilon} n) = \mathsf{DTIME}(n^a \log^{2^{k+1}\epsilon} n)$$

Continuing like this after N steps we get

$$\mathsf{DTIME}(n^a) = \mathsf{DTIME}(n^a \log^{2^N\epsilon} n)$$

But now $2^N\epsilon > 1$. Hence $n^a \log n = o(n^a \log^{2^N\epsilon} n)$. Therefore this violets the time hierarchy theorem. Hence contradiction $\lightning$ Therefore

$$\mathsf{DTIME}\left(n^a\right) \subsetneq \mathsf{DTIME}\left(n^a (\log n)^{\epsilon}\right)$$

■