Soham Chatterjee & **Rishav Gupta**
Email: sohamc@cmi.ac.in & rishavg@cmi.ac.in
Course: Complexity Theory 1

Assignment - 1
Roll: BMC202175 & BMC202145
Date: February 10, 2023

Problem 1

Prove that if a language L is in $DTIME(T(n))$ then it can be solved in time $T^2(n)$ by a one-tape TM.

Solution: Let M be a turing machine with k tapes which decides $L \in DTIME(T(n))$ in $T(n)$ steps. Now let $\mathcal{M}$ be a single tape turing machine.

Encoding M's tapes on $\mathcal{M}$'s tape:

The TM $\mathcal{M}$ encodes k tapes of M on a single tape by using locations $1, k+1, 2k+1, \ldots$ to encode the first tape, l, locations $2, k+2, 2k+2 \ldots$ to encode the second tape, locations $3, k+3, 2k+3 \ldots$ to encode the third tape and so on.

For every symbol a in M 's alphabet, $\mathcal{M}$ will contain both the symbol a and the symbol $\hat{a}$. In the encoding of each tape, exactly one symbol will be of the two types of symbols. $\hat{a}$ indicates that the corresponding head of M is positioned in that location.

$\mathcal{M}$ will not touch the first $n+1$ locations of its tape (where the input is located) but rather start by taking $O\left(n^2\right)$ steps to copy the input bit by bit into the rest of the tape, while encoding it in the above way.

Simulation:

To simulate one step of M, the machine $\mathcal{M}$ makes two sweeps of its work tape: First it sweeps the tape in the left-to-right direction and records to its register the k symbols that are marked by. Then $\mathcal{M}$ uses M's transition function to determine the new state, symbols, and head movements and sweeps the tape back in the right-to-left direction to update the encoding accordingly. Clearly, $\mathcal{M}$ will have the same output as M.

Complexity Analysis:

Since on n-length inputs M never reaches more than location $T(n)$ of any of its tapes, $\mathcal{M}$ will never need to reach more than location $2n + kT(n) \leq (k+2)T(n)$ of its work tape, meaning that for each of the at most $T(n)$ steps of M, $\mathcal{M}$ performs at most $5kT(n)$ steps (sweeping back and forth requires about $4 \cdot k \cdot T(n)$ steps, and some additional steps may be needed for updating head movement and book keeping). SO total time it takes $5kT(n) \times T(n) = 5kT^2(n) = O(T^(n))$ time.

Hence L can be solved in time $T^2(n)$ by a one-tape turing machine. ■

Problem 2

Prove that if L is accepted by a k tape turing machine in $T(n)$ time then it can be also accepted by a two tape turing machine in $T(n) \log T(n)$ time.

Solution: First we will prove a lemma.

Lemma: Define a bidirectional Turing machine to be a Turing Machine whose tapes are infinite in both directions. For every $f : \{0,1\}^* \to \{0,1\}$ and time-constructible $T : \mathbb{N} \to \mathbb{N}$ if f is computable in time $T(n)$ by a bidirectional Turing Machine M using k tapes then it is computable in time $4T(n)$ by a standard Turing Machine $\hat{M}$.

Proof: The proof uses the idea of folding the bidirectional tape into an uni-directional tape . For implementing this idea we use the following construction . Let T be our original bidirectional tape TM with alphabets in $\Sigma \cup \{\_\}$,we construct T' with alphabets in the set $\Sigma \cup \{\_\} \times \Sigma \cup \{\_\}$ .Now we enumerate the start cell of T and T' both with 0 and the cell on right with $1, 2 \ldots$ and the cells on left for T with $-1, -2, \ldots$ .Now the ith cell of T' will store two values one for the ith cell of T and other for the $-ith$ cell

of T that is the ith cell content of T' = (ith cell content of T,$-ith$ cell content of T). Now if wee are at 0 th position and then the head moves left in T, then in T' we will move right and make the change in the second coordinate and keep moving right and modifying the second coordinate in T' if we are moving in left in T while being in a position with index ≤ 0, if we move right in T while being at index< 0 we will move left in T' and modify the second coordinate, if we are at an index>0 in T we will move accordingly in T' modifying the first coordinate . Now by this process we can simulate a bidirectional TM T with an uni-directional TM T' without any blow up in time because for one shift in T we are making exactly one shift in T'.

□

Let L is accepted by the turing machine M which is a k tape turing machine in time $T(n)$ (we will write it as T). Therefore M's one tape is input tape, one tape is output tape and rest are work tapes Let $\mathcal{M}$ be a two tape turing machine. So one of the tape is input tape and the other one is work tape/output tape. Hence we will simulate $k-1$ tapes of M in one tape of $\mathcal{M}$

Encoding M's tapes on $\mathcal{M}$'s tape:

We add a special kind of blank symbol ⊠ to the alphabet of M's parallel tapes with the properties that this symbol is ignored in the simulation. For example, if M's tape contents are 010 then, this can be encoded as $0 \boxtimes 10$ or $0 \boxtimes \boxtimes 1 \boxtimes 0$ and so on or just 010

For convenience, we think of $\mathcal{M}$'s parallel tapes as infinite in both the left and right directions by the Lemma. Thus we index their locations by $0, \pm 1, \pm 2, \ldots$ Normally we keep $\mathcal{M}$'s head on location 0 of these parallel tapes. We will only move it temporarily to perform a shift when, following our general approach, we simulate a left hand movement by sifting the tape to the right and vice versa. At the end of the shift, we return the head to location 0.

We split each of $\mathcal{M}$'s parallel tapes into zones that we denote by R_0, L_0, R_1, $L_1, \ldots$ (We'll only need to go up to $R_{\log T}$, $L_{\log T}$ since the turing machine runs for time T, it can at most access T cells of each tape.). The cell at location 0 is not at any zone. Zone R_0 contains two cells immediately right of location 0 (i.e. location $+1, +$). Zone R_1 contains the four cells $+3, +4, +5, +6$. Generally for every $i \geq 1$, Zone R_1 contains the 2×2^i cells that are right to the R_{i-1} (i.e. locations $2^{i+1}-1, \ldots, 2^{i+2}-2$). Similarly Zone L_0 contains two cells indexed by -1 and -2 and generally Zone L_i contains the cells $-2^{i+2}+2, \ldots, -2^{i+1}+1$

Invariants:

We shall always maintain the following invariants:

- Each of the zones is either empty, full, or half-full with non-⊠-symbols. That is, the number of symbols in zone R_i that are not ⊠ is either 0, 2^i, or 2×2^i and the same holds for L_i . (We treat the ordinary symbol □ the same as any other symbol in Γ, and in particular a zone full of □'s is considered full). We assume that initially all the zones are half-full. We can ensure this by filling half of each zone with ⊠ symbols in the first time we encounter it.
- The total number of non-⊠-symbols in $R_i \cup L_i$ is 2×2^i. That is, either R_i is empty and L_i is full, or R_i is full and L_i is empty, or they are both half-full.
- Location 0 always contains a non-⊠-symbol.

Performing a Shift:

Now when performing the shifts, we do not always have to move the entire tape, but we can restrict ourselves to only using some of the zones.

We illustrate this by showing how $\mathcal{M}$ performs a left shift on the first of its parallel tapes

1. $\mathcal{M}$ finds the smallest i_0 such that R_{i_0} is not empty (Hence $R_0, R_1, \ldots, R_{i_0-1}$ are empty). Note that this is also the smallest i_0 such that L_{i_0} is not full. We call this number i_0 the index of this particular shift.

2. $\mathcal{M}$ puts the leftmost non-$\boxtimes$-symbol of R_{i_0} in position 0. We now denote the new zones as R_i, L_i. Now there are $2 \times 2^{i_0} - 1$ non-$\boxtimes$-symbols remain in R_{i_0} if R_{i_0} is full and otherwise $2^{i_0} - 1$ non-$\boxtimes$-symbols remain in R_{i_0}.

   Now if R_{i_0} was full before we shift the $2 \times 2^{i_0} - 1$ to the $R_0, \ldots, R_{i_0}$ each of the zones becomes half filled. In other words we shift the leftmost $2^{i_0} - 1$ non-$\boxtimes$-symbols to the zones $R_0, \ldots, R_{i_0-1}$ each of which becomes half filled because in total they take the space $\sum_{i=0}^{i_0-1} 2 \times 2^i = 2(2^{i_0} - 1)$. And hence R_{i_0} also becomes half filled because 2^{i_0} non-$\boxtimes$-symbols remain in R_{i_0}

   If R_{i_0} was half-full before we shift the $2^{i_0} - 1$ non-$\boxtimes$-symbols of R_{i_0} into the zones $R_0, \ldots, R_{i_0-1}$ filling up exactly half of the symbols of each R_i zones where $i \in \{0, \ldots, i_0 - 1\}$. Now the zone R_{i_0} is empty.

3. $\mathcal{M}$ performs symmetric operation to the left of position 0. That is for j starting from $i_0 - 1$ down to 0, $\mathcal{M}$ literally moves the 2×2^j symbols of L_j to fill the cells of L_{j+1}. Finally $\mathcal{M}$ moves the symbol originally in position 0 to L_0

   So if R_{i_0} was full before L_{i_0} was empty and all $L_0, \ldots, L_{i_0-1}$ were full. After the shifting L_j contains all the $2 \times 2j$ symbols of L_{j-1}. So L_{i_0} becomes half filled. And since at each step L_{j-1} was emptied to move all its content to L_j in the next step L_{j-1} becomes half filled. Therefore all $L_0, \ldots, L_{i_0-1}$ are half filled.

   If R_{i_0} was half filled before L_{i_0} was also half filled and all $L_0, \ldots, L_{i_0-1}$ were full. After the shifting L_j contains all the $2 \times 2j$ symbols of L_{j-1}. So L_{i_0} becomes full. And since at each step L_{j-1} was emptied to move all its content to L_j in the next step L_{j-1} becomes half filled. Therefore all $L_0, \ldots, L_{i_0-1}$ are half filled.

4. At the end of the shift all of the zones $R_0, L_0, R_1, L_1, \ldots, R_{i_0-1}, L_{i_0-1}$ are half-full, R_{i_0} has 2^{i_0} fewer non-$\boxtimes$-symbols and L_{i_0} has 2^{i_0} additional non-$\boxtimes$-symbols. Thus our invariants are maintained

Complexity Analysis:

The total cost of performing a shift is proportional to the total size of all zones involved $R_0, L_0, \ldots, R_{i_0}, L_{i_0}$. That is

$$O\left(\sum_{j=0}^{i_0} 2 \times 2^j\right) = O\left(2^{i_0}\right) \text{ operations}$$

After performing a shift with index i the zones $R_0, L_0, \ldots, R_{i-1}, L_{i-1}$ are half-full which means it will take at least $2^i - 1$ left shifts before the zones $L_0, \ldots, L_{i-1}$ becomes empty or at least $2^{i-1} - 1$ right shifts before the zones $R_0, \ldots, R_{i-1}$ becomes empty. Hence once we perform a shift with index i, the next $2^i - 1$ shifts of that particular parallel tape will all have index less than i. This means that for every one of parallel tapes at most a $\frac{1}{2^i}$ fraction of the total number of shifts have index i. Since we perform T shifts and the highest possible index is in the course of simulating T steps of M is

$$O\left((k-1)\sum_{i=1}^{\log T} \frac{T}{2^{i-1}} 2^i\right) = O(T \log T)$$

■

Problem 3

Show that if $P = NP$, then every NP search problem can be solved in polynomial time.

Solution: Let $L \in NP$ be any language. Let N be the turing machine where for input $x \in L$, y is the certificate such that $|y| \leq g(|x|)$ where g is a polynomial. which solves L within $f(n)$ time where f is a polynomial. Now since SAT is a $NP - complete$ problem the language L can be reduced to SAT by a polynomial time Turing machine M_1 which converts an instance $x \in L$ to an instance of ϕ_x of SAT in $p(|x|)$ time following the Cook-Levin Theorem where p is a polynomial.

Now since $P = NP$ given, SAT can be decided by a polynomial time turing machine M_2 which runs in $q(n)$ time where q is a polynomial. Now we will prove that given an instance of SAT, ϕ we can find a satisfying assignment for ϕ. Suppose ϕ is an $n-$variable boolean formula.

Algorithm:

Step 0: Ask the oracle if φ is satisfiable.
Step 1: Then fix $x_1 = 1$. Now putting the value of x_1 in ϕ we have a new boolean formula $\phi'(x_2, \ldots, x_n) = \phi(1, x_2, \ldots, x_n)$. We run M_2 on ϕ', if it accepts then keep $x_1 = 1$. Otherwise keep $x_1 = 0$ since we know there is a satisfying assignment from step 0 and each variable can have only two values 0 or 1.
Step 2: Now in ϕ' fix $x_2 = 1$ and repeat the same process as step 1. Then again keep the value for x_2 and fix x_3 Repeat then process for all variables $x_1, \ldots, x_n$ till every variable is fixed.
Step 3: At the end of the recursive process of step 2 we have a satisfying assignment $\overline{x}$ for the boolean formula phi.

Complexity Analysis of Algorithm

Step 0 takes $q(|\phi|)$ time. Then in each $i-$th recursion of step 1 and step 2 takes

$$q\Big(|\phi(1, \ldots, 1, x_{i+1}, \ldots, x_n)|\Big) \leq q(|\phi|) \text{ time}$$

So in total it takes $(n+1)q(|\phi|)$ time to find a satisfying assignment for the given SAT formula.

Now if we input x to N then the satisfying assignment for ϕ_x gives us the full path description for a path which reaches the accept state on input x (if it reaches one) implies it contains the information of the accepting certificate for the input x on N which solves L , due to the construction we used in Cook-Levin Theorem. So the satisfying assignment for the boolean formula ϕ_x gives us the description of certificate from which the certificate can be easily extracted in $O(l)$ where l is the length of satisfying assignment which is $poly(|x|)$. So we found the certificate for the accepting path which is essentially the solution for L on input x in polynomial time.

Therefore every NP search problem can be solved in polynomial time ■

Problem 4

Prove that $P \neq SPACE(O(n))$.

Solution: Lets assume $P = SPACE(O(n))$ then we get that if a problem can be solved in $O(n)$ space then it can be also solved in time n^i for some $i \in \mathbb{N}$. Now lets take a problem L in $SPACE(O(n^2))$ then we make a $SPACE(O(n))$ problem from this problem by padding this problem .

Suppose $L \in SPACE(O(n^2))$ then we define L' to be

$$L' = \{x\#0^k \mid x \in L \ \& \ k = |x|^2\}$$

Now notice that L' is in $SPACE(O(n))$ because checking that $|x|^2 = k$ can be done in $SPACE(k)$ easily and also since L is in $SPACE(O(n^2))$ and $k = |x|^2$ checking if $x \in L$ will take $SPACE(O(|x|^2))$ which is $SPACE(O(k))$ which implies this also takes $SPACE(O(n))$. Hence L' is in $SPACE(O(n))$.

This implies that L' can be solved in some $TIME(O(n^i))$ for some i. Now we claim using L' we can claim that L can also be solved in $TIME(O(n^{2i}))$ because if we take an instance of $x \in L$ of we can

convert it into a instance of L' in $TIME(|x|^2)$ just by padding it by $|x|^2$ many 0's. Then we can solve this instance of L' in $TIME(O(|x|^{2i}))$ which implies L can be solved in $TIME(O(n^{2i}))$ hence we get that $L \in P$. Which implies that $L \in SPACE(O(n))$ (because we assumed that $P = SPACE(O(n))) \implies SPACE(O(n^2)) \subset SPACE(O(n)) \implies SPACE(O(n^2)) = SPACE(O(n))$. Which is not true by Space Hierarchy Theorem. Hence contradiction. Therefore our assumption was wrong and $P \neq SPACE(O(n))$. ■

Problem 5

Show that if $P = NP$ then every language $A \in P$ except $A = \phi$ or $A = \Sigma^*$ is NP complete.

Solution: We need to show if $P = NP$ then every problem in P except ϕ and Σ^* is NP-complete . For this let A be a problem in P where $A \neq \phi$ or $A \neq \Sigma^*$. There exists $w_i \in A$ and $w_o \notin A$. We need to show that A is NP-complete i.e for any problem M in NP $\exists$ a polynomial time reduction from M to A . The reduction is as follows given a problem M in NP since we know $P = NP$ implies there exist a polynomial time Turing machine T which solves M in time $p(n)$ where p is a polynomial. So for reducing an instance of M to an instance of A we will just naively run T on the input x, and if the given input x belongs to M then we will naively search for a string which is in A and return that. If it does not belongs to M then we will search for a string which is not in A and return it.

Now searching for a string which is in A or not naively will take constant with respect to the input given to check in M or not because on any size of input it will naively start searching and verifying from strings of length 1 till we get a satisfying or non-satisfying assignment of A . We know this process of searching will halt because A is neither ϕ nor Σ^*.Hence we reduced an instance of M into an instance of A using the above reduction such that a string $x \in M \iff f(x) \in A$. And if $|x| = n$ then this reduction took at most $p(n) + c$ many steps where c is max(steps to find a satisfying assignment of A, steps to find a non-satisfying assignment of A) since the TM T will take at most $p(n)$ time and the assignment finder will take constant time with respect to $|x|$. Hence this reduction is a polynomial time reduction.

This implies A is NP-hard and we know $A \in NP$. Therefore A is $NP - complete$. ■

Problem 6

Define $ALL_{DFA} = \{<A> : A \text{ is a } DFA \text{ and } L(A) = \Sigma^*\}$. What is the complexity of this language? What do you think about ALL_{NFA}?

Solution:

- We claim that ALL_{DFA} is in P. Let $D = (Q, \Gamma, \delta, q_0, F)$ be any DFA where Q is the set of states, Γ is the set of alphabets, δ is the transition function, q_0 is the starting state and F is the set of final states. Now let $R = L(D)$. Now we create a Turing Machine M which converts D to a DFA, D' for which $L(D') = R^c$ which can be done by changing the final states from F to $Q \setminus F$ which can be done in polynomial time. Now if $R = \Sigma^*$ then $L(D') = R^c = \phi$ which implies the initial state and the final states are not connected in the graph of the DFA, D'. So we run DFS algorithm in the graph of D' from the vertex q_0. If in the runtime of the DFS algorithm at any point any vertex of $Q \setminus F$ is visited then we reject D otherwise if at the end of the DFS none of the vertices of $Q \setminus F$ are visited then we accept D. Thus D is accepted $\iff \forall\ q \in Q \setminus F$ q_0 and q are not connected in $D' \iff L(D') = \phi$. Therefore $L(M) = ALL_{DFA}$.

  The DFS algorithm takes polynomial runtime on the graph of D'. And the conversion of D to D' takes also polynomial time. Hence M takes polynomial time to decide ALL_{DFA}. Therefore $ALL_{DFA} \in P$

- Case for ALL_{NFA} : We claim that ALL_{NFA} is in $PSPACE$.
  Proof- Given an NFA with q states we claim that if the NFA rejects any string it must reject a string of length at most 2^q. Because if it rejects some word w of length greater than 2^q then ,we know that a NFA with q states can be simulated by a DFA D with 2^q many states implies that on input w ,D will reach a rejecting state r now the path from starting state s to r has length $> 2^q$ which

is the number of nodes in the DFA hence it must have got cycled somewhere if we keep removing these cycles we get a path from s to r which has a length smaller than 2^q implies that the D must reject some string of length at most 2^q if it rejects some string.

• Now using the above fact we will develop a $NPSPACE$ machine , T for the complement of the given problem , which will show the above problem to be in $CONPSPACE$ and by Savitch's we know that $NPSPACE = PSAPACE \implies CONPSPACE = PSPACE$ which will imply that ALL_{NFA} is in $PSPACE$.

• **For proving this we will do the following on input N on machine T where N is an NFA-**

Step 1- Mark the start state s of N.

Step 2- Nondeterminstically select an input symbol and now unmark $N's$ start state then mark all states which can be reached by reading that letter to simulate reading of that letter.

Step 3- Now nondeterminstically select another input symbol and then change the position of markers to all states which can be reached from marked states in the previous iteration by reading this guessed letter .

Step 4- Now we will repeat **'Step 3'** for 2^q many steps recursively.

Step 5-We will accept N if at any stage none of the markers lie on an accept state of N , otherwise we will reject N.

• **Explanation-** Now we are accepting the machines which are rejecting a word of length at most 2^q since we know that if the language of N is not Σ^* it must reject a word of length at most 2^q ,this implies we are accepting the machines which are accepting the $NFA's$ which are rejecting some string i.e. , its language is not Σ^*. Hence we are accepting complement of ALL_{NFA}.

• **Analysis-** Now at any iteration we just need to store marks of the previous iteration and then generate marks of the current iteration then forget marks of previous iteration . So if the number of states in the NFA ,N is q then the space needed is $O(q)$.Implies this machine T uses linear space on the size of graph implies complement of ALL_{NFA} is in $NPSPACE = PSPACE$ which implies ALL_{NFA} is in $PSPACE$.

■