Complexity Assignment 1

Name: Shubh Sharma
Roll Number: BMC202170

1

A Language L is in $DTIME(T(n))$, if it can be solved by a k-tape Deterministic Turing Machine with the language Σ in time $O(T(n))$. For any such turing machine we can construct and equivalent Single Tape Turing Machine with the language $(\Sigma \cup \{\sqcup\} \times \{\sqcup, \uparrow\})^k$ which can be thought of as a Single tape turing machine with $2k$ tracks(numbered 1,2,3 .. and so on) such that the $2i-1th$ track represets the i^{th} tape of the k tape turing machine and in the $2i^{th}$ track all places except for the corresponding pointer location are $\sqcup$ where the location where the pointer is pointing to has $\uparrow$.
One Step of the k tape turing machine can be emulated on the one tape turing machine by, starting to read from the left, until the location of all the pointers is read, then starting from the left again, making changes on all k tracks, one cell at a time, then going back to the left most tape and restarting the process.

If the Machine takes $c = O(T(n))$ steps to execute then it reads atmost c cells in the tape. Hence to emulate on step of the k tape turing machine on a single tape turing Machine it would take $4c$ steps which would be-

- Reading all cells to find where the pointers would be - c step_NFA Universality is PSPACE-complete_s.
- Going back to the beginning - c steps
- Making changes in the tapes - c steps
- Going back to the beginning - c steps

and these are repeated exactly c times hence it takes c^2 steps at most which makes it $O(T(n)^2)$.

2.

Consider one of the tapes of the 2-tape turing machine as the **main tape** and the other one as the **support tap**. Initially keep the support tape empty.
Both tapes have k many tracks, same as the number of tapes in the initial Turing Machine. Both the tapes are two way infinite tapes.
Idea: consider the position of the head to be fixed and imagine sliding the tracks of the tape instead.
The storage in the main tape is divided into multiple sections as follows.

- The cell where the head is pointing to is called the *Home* column.
- To its right and left, the sections are lablelled $R_1, R_2, \ldots$ and $L_1, L_2, \ldots$ respective
- The size of section labelled R_i or L_i is 2^i cells

The **support tape** is used to move a string of length n on the **main tape** by m positions in $O(n+m)$ time in the following way.
The header in the **main tape** goes to the start of the string, then it reads the string, and while doing that it deletes it from the **main tape** and copies it to the **support tape**, then the header moves back to the position of the start of the string, and then moves m positions, then it deletes the work from the **support tape** and copies it onto the **main tape** in the new location.

This process took $3n + m$ steps.
From now on, the support tape will not be mentioned and shifting a string will be considered to take linear time.

In the initial setup, the letter where the header is at, is placed in the *Home* column, while the rest are placed to the left and right such that, half of each section is filled with letters (include blank spaces from the k tape turing machine as characters if there aren't enough to fill)

The invariant to be maintained - number of filled cells in R_i + L_i are 2^i

To simulate a step, if you want to change a letter and move the header to the right(move the tape to the left.)

- scan to the right to find the first empty or half empty section. keep moving the previous sections to fill up half of the sections after them until there is space in R_1 then move the element in the *Home* column to R_1
- Scan to the right to find the first non empty section. If it is full, copy half of it onto the different sections, such that the last element is now in the *Home* column, and if the first non empty section is half full, copy all its contents to the right filling half of each of the following section, putting the required element in the *Home* column.

Moving in the other direction in symmetric.

To prove this takes $\log(T(n))$ steps
Consider the section L_i
Moving 2^{i-1} elements out of L_i takes 2^{i+1} steps atmost, after that L_j are all half full $\forall j < i$. To access L_i we either have to empty all $L_j, \forall j < i$, which takes 2^{j+1} steps for each, givinf a total of $2^{i+1}$$ steps.
Moving elments into L_i takes atmost 2^{i+1} steps, trying to add more elements in L_i would require filling all $L_j, \forall j < i$ which would take $2^j + 1$ steps each, which gives a total of 2^{i+1} steps

Hence L_i can be access atmost twice in 2^{i+2} steps, hence after $c \cdot f(n)$ steps, L_i can be accessed atmost $\frac{c \cdot f(n)}{2^i}$ times.
This process in repeated for every track, hence emulating one step of the k tape turing machien takes atmost $k \log_2(c \cdot f(n))$ steps.
Doing that for $c \cdot (f(n))$ steps gives $kc \cdot f(n) \log(f(n))$

To prove: one side infintie tape turing machine and two way infinte tape can exectute with same time complexity.

To emulate a two way infinite turing machine, we can divide the tape into two tracks, which will be equivalent to folding the two way tape at an arbitrary point, keep track of which track the header is in (this can be done by copying the states where one of two copies handes the case where head is on the upper track and the other handles the case where the head is on the lower track), while moving between the letters in the left most elment, we can consider switching the tracks, hence one step in the two way infinite tape corresponst to one step in the one way infintie tape.

Hence if something can be done in a k tape turing machine in $T(n)$, it can be done in a 2 tape turing machine in $T(n) \log(T(n))$ time.

3

We first show that if $P = NP$ the search version on SAT is in P. we cam show this by first making 2 by setting the first variable to `true` and `false` , and check for the decision version of sat in both cases. if any of the cases is true, we pick owe, permanently set the variable that value and continue the process with other variables. If the number of variables in n then we only call the decidability version n times, and we make multiple cases upto n times.

By Cook-Levin theorem, we can convert every problem (Say L) in NP to SAT, also the conversion is a Levin Reuction, hence an assignment to SAT can be mapped back a to the language L in polynomial time, hence

$L \in P$.

4.

FTSOC: $P = SPACE(n)$
Then, there exists a polynomial time solution for every language in $SPACE(n)$. Consider $L \in SPACE(n^2)$, and $w \in L$. There exists a Turing machine such that it accepts w on space $O(|w|^2)$, now we can pad it to get $w' = w10^{|w|^2-|w|-1}$, which makes it $|w|^2$ long. That means, there is a machine which takes in w' and tells if w is accepted by a turing machine in $O(n)$ space. Thus we construced w' in polynomial time and checked if w is in L in polynmial time. which gives us
$SPACE(n^2) \subseteq P \subseteq SPACE(n)$ which violates the Space Hierarchy Theorem.

5. Not to be done

6.

If $P = NP$, then any language $L \in NP$ can be solved in polynomial time. For any language $A \in P$ such that $A \neq \emptyset, A \neq \Sigma^*$, we can find words $w \in A$ and $w' \notin A$. For any $x \in L$, we can reduce it to A as follows.
We compute if $x \in L$ in polynomial time, If it does, we map it to w otherwise, we map it to w' then x is accepted by L if and only if w is accepted by A, and the solution was computed in polynomial time, hence this is a polynomial time reduction. Thus, we can get that A is NP complete.

7.

The first step is to verify if the given input is a valid finite state automata. this can be done in $O(n)$ as we only need to check as per the given description
$ALL_{DFA} \in P$. To show that, consider a turing machine, that takes in a DFA as an input, the DFA can be encoded in the following way-

- The encoding consists of multiple sections separated by $
- The first section tells the number of states, assuming states are now identified using numbers.
- The next section's first letter represents the starting state and the rest represent the accepting state
- The following sections are of the form `State| Letter | State` which represents the transition from the first state to the second one written using the letter.
  We can replace each of the `State | Letter | State` transition to `State | State`, and now the DFA representation becomes the representation of the graph, now we can check for reachability from teh start state to all non accepting states, if any of them is true, then we can say the DFA does not belong to our language. The conversion of the DFA transitions to graph edges is Polynomial time, and we are using reachability (which is polynomial time) polynomial. Hence the claim is proved.

$ALL_{NFA} \in PSPACE$. To show that, consider a turing machine that takes in a description of an NFA similiar to the description of the DFA in the previous section, but if a transition goes from 1 state to k states using a single letter, we accept it as k distinct transitions.
For every NFA with q states we can make an equivalent DFA with 2^q states.
We can then check words with length upto 2^q length non deterministically, along with that we need to keep a

counter that will check if the length of the word exceeds 2^q and one tape that keeps track of the set of states that the word can be in.
If at any point one of the states in the set are the non accpeting states we can conclude that the NFA does not belong to ALL_{NFA} During this process, one counter was used which took n cells and the set of states that the run can go to, which also took upto n states, Hence $ALL_{NFA} \in NSPACE = PSPACE$