CSC2401 Introduction to Computational Complexity
Student: Jia Sheng, jsheng@dgp.toronto.edu
Student #: 992810115
Instructor: Allan Borodin

Answer to Assignment 1

1 ANSWER TO PROBLEM 1

Proof

I will use *diagonalization* to prove this problem.

First, note that $\exists$ a k-tape Universal Turing Machine U , s.t. if k-tape Turing Machine M_x runs in time $t(n)$ on input y, $|y| = n$, then $U(< x, y >)$ runs in time $t \leq c_x t(n)$, for some const c_x. U simulates M_x as follows: on input $< x, y >$, U first writes the initial state q_0 of M_x on tape 1, then re-layout x and y on the input tape in the form of $y_0 x y_1 x y_2 x ... y_n x$, where $y = y_0 ... y_n$. Because $|x|$ is finite constant, such steps takes $c_1|y|$ time, c_1 is some const. After that, U repeats these steps to simulate M_x:

1. read current state q of M_x from tape 1;
2. read current input y_i;
3. read x
4. simulate M_x, given x, q and y_i
5. write new state of M_x on tape 1;

Since U has the same number of tapes as M_x, step 4 takes exactly the same number of steps as M_x and other steps take finite constant time. So the whole simulation will take $c_2 t(|y|)$ time. Once M_x accepts y, U wipes all the working tapes and moves to an accepting state. Thus U can simulate M_x in $(c_1 + c_2)t(|y|) = c_x t(|y|)$ time.

Then, similar to the proof of Time Hierarchy Theorem on the lecture notes, let U' be a (k+1)-tape Turing Machine. U' is almost the same as U except that it has one more working tape. On input $y = 1^l 0x$, U' simulates $U(< x, y >)$ for $T_2(|y|)$ steps. U' first computes $T_2(|y|)$ and puts $1^{T_2(|y|)}$ on the $(k+1)^{th}$ tape as a counter, then while simulating $U(< x, y >)$, decrease the counter to ensure that it doesn't simulate U more than $T_2(|y|)$ steps. Then U':

- accepts y if the simulation rejects within $\leq T_2(|y|)$ steps, or simulation doesn't complete within $T_2(|y|)$ steps.

- rejects if $U(< x, y >)$ did accept in $\leq T_2(|y|)$ steps.

Then let $L = \mathcal{L}(U')$, then L is not accepted by any k-tape Turing Machine running in time $T_1(n)$. If $L = \mathcal{L}(M_z)$, M_z has k-tape and runs in time $T_1(n)$, i.e., $time_{M_z}(y) \leq T_1(|y|)$, for almost all y. By the first part of the proof, on input $< z, y >$, the k-tape U can run in time $\leq c_z T_1(|y|)$. Because $\lim_{n\to\infty} \frac{T_1(n)}{T_2(n)} = 0$, then for any c_z, $T_2(|y|) > c_z T_1(|y|)$ for sufficiently large l. Therefore, for that l, U' will complete the simulation within $T_2(|y|)$ steps. When U' completes, U' accepts iff M_z rejects. So $L \notin \mathcal{L}(M_z)$.

So far we have got a language L that is decided by a $k+1$-tape TM U' running in time T_2 but not decided by any k-tape TM running in time T_1. ■

2 ANSWER TO PROBLEM 2

Idea

It is obvious that $DTIME(n) \subseteq DTIME(nlogn)$ (every TM running in time $c * n$ surely runs in time $c * nlogn$). Assuming $DTIME(nlogn) - DTIME(n) = \emptyset$, i.e. $DTIME(n) = DTIME(nlogn)$, I want to show $DTIME(nlogn) = DTIME(nlog^2n)$, thus $DTIME(n) = DTIME(nlog^2n)$. But according to Time Hierarchy Theorem, $DTIME(n)$ is properly contained in $DTIME(nlog^2n)$, for $\lim_{n\to\infty} \frac{nlogn}{nlog^2n} = 0$. Thus the assumption is wrong and $DTIME(n)$ is properly contained in $DTIME(nlogn)$.

Proof

For $L \in DTIME(nlog^2n)$, L is accepted by TM M_1, let $L' = \{1^{\lceil nlog(n) \rceil}0x | x \in L, n = |x|\}$. L' pads a string of $\lceil nlog(n) \rceil$ $1's$ and one 0 in front of x. We can see that $L' \in DTIME(nlogn)$.

We can decide L' as follows: first check the $1's$ before x and if there is $\lceil nlog(n) \rceil$ $1's$, run M_1 on x. The total time is $\lceil nlog(n) \rceil + 1 + c_1(nlog^2n) \leq$

$c_2((\lceil nlog(n)\rceil + 1 + n) * log((\lceil nlog(n)\rceil + 1 + n)))$, c_1, c_2 are consts. So $L' \in DTIME(nlogn)$.

Under the assumption, $DTIME(nlogn) = DTIME(n)$, we have $L' \in DTIME(n)$, L' is accepted by M_2 running in time n. Then we can show that $L \in DTIME(nlogn)$.

We construct a TM M_3, on input $x \in L$, M_3 first pads $\lceil nlog(n)\rceil$ $1's$ and a 0 before x, then simulate M_2, the total time is $\lceil nlog(n)\rceil + 1 + c_3(\lceil nlog(n)\rceil + 1 + n) \leq c_4 nlogn$.

So for every $L \in DTIME(nlog^2n)$, $L \in DTIME(nlogn)$. therefore $DTIME(nlog^2n) \subseteq DTIME(nlogn)$. And it also has $DTIME(nlogn) \subseteq DTIME(nlog^2n)$, we have $DIME(n) = DTIME(nlogn) = DTIME(nlog^2n)$, which is a contradiction.

As to the second question: is $DTIME(n)$ properly contained in $DTIME$ $(n\sqrt{logn})$? The answer is yes and the proof is just the same as the first part. We only need to show that under the assumption $DTIME(n) = DTIME(n\sqrt{logn})$, we have $DTIME(n) = DTIME(n\sqrt{logn}) = DTIME$ $(nlogn)$, and thus a contradiction according to what we have got.

For $L \in DTIME(nlogn)$, we pad L to $L' = \{1^{\lceil n\sqrt{log(n)}\rceil}0x | x \in L, n = |x|\}$. Use the same technique as before, we can show that $L' \in DTIME(n\sqrt{logn})$, because $\lceil n\sqrt{log(n)}\rceil + 1 + c_1(nlogn) \leq c_2((\lceil n\sqrt{log(n)}\rceil + 1 + n) * log((\lceil n\sqrt{log(n)}\rceil + 1 + n)))$.

Then under the assumption, $L' \in DTIME(n)$. That implies $L \in DTIME$ $(n\sqrt{logn})$, since x in L cannot go beyond the bound $n\sqrt{logn}$ in order to make L' linearly time bounded. Then comes again the contradiction $DTIME(n) = DTIME(nlogn)$. ■

3 ANSWER TO PROBLEM 3

Proof

If the graph G_F does not have a " bad node x " , which means there is no node x that there exits a directed path from x to $\overline{x}$ and a directed path from $\overline{x}$ to x, then we can give an assignment $\sigma\{x_1...x_n\} \rightarrow \{True, False\}$ to make F satisfiable. Here is how we accomplish this. (in the proof, *node* stands for x or $\overline{x}$)

First, in G_F, we check all the paths from $\overline{x}$ to x and from x to $\overline{x}$, for some x. If there is a path from $\overline{x}$ to x, $\sigma(x)$ has to be $True$ to make F satisfiable, and all ancestor *nodes* of $\overline{x}$ have to be assigned with $False$, all descendant *nodes* of x have to be $True$ (here ancestor node of one node u means there is a directed path from that node to u; descendant node of one node u means there is a directed path from u to that node).

$$\{AncestorNodes\} \to \overline{x} \to ... \to x \to \{DescendantNodes\}$$

Note that *ancestornodes* and *descendantnodes* cannot share common z, because otherwise there is a path from $x \to ... \to z \to ... \to \overline{x}$ as well as a path from $\overline{x}$ to x, which is contradictory to the condition. We can use the same way to treat the circumstance of path from x to $\overline{x}$. After the treatment, we remove all the related nodes and edges from G_F, for they are all assigned a boolean value, and we are left with $\tilde{G_F}$.

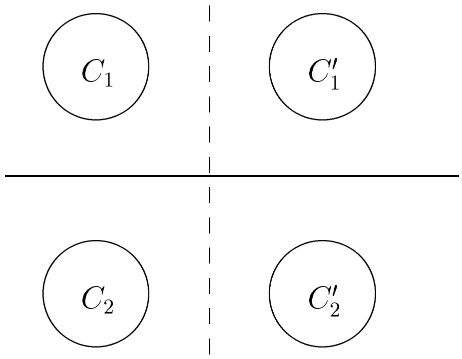
(One remark: $\tilde{G_F}$ does not have paths like $y \to x$ and $y \to \overline{x}$ at the same time, because otherwise, there is a path $y \to \overline{x} \to \overline{y}$, which is removed from previous steps. Neither does it have $x \to y$ and $\overline{x} \to y$ at the same time)

We can divide the graph $\tilde{G_F}$ into several components $\{C_i\}$, where the nodes within a C_i are connected and the nodes of different components are disconnected (two nodes are "connected" means one can walk from one node to the other through edges without worrying about the direction) Then we have:

1. $\forall x$, x and $\overline{x}$ are in different components. That is because there is no path between x and $\overline{x}$ in $\tilde{G_F}$.

2. for a $C = \{V_C, E_C\}$, $V_C = \{node_i\}$, $E_C = \{(node_i, node_j)\}$, there is a "mirror" component $C' = \{V'_C, E'_C\}$, s.t. $V'_C = \{\overline{node_i}\}$ and $E'_C = \{(\overline{node_j}, \overline{node_i})\}$. The reason is that if there is a path from node v to node u, there is a path from $\overline{u}$ to $\overline{v}$. Together with previous fact, we can see such a "mirror" component does exist. Such two components can be grouped into one component-group and

3. different component-groups don't share nodes. That is because different components are disconnected. A common y in C_1 and C_2 will make those components connected.



So now we have a method to partition the $\tilde{G_F}$ into disjoint component-groups, $\{(C_1, C_1'), (C_2, C_2'), ..., (C_n, C_n')\}$. We can assign $True$ to the literals within C_i and $False$ to those in C_i'. There will be no confliction within the components (x and $\overline{x}$ in different components), no confliction within component-groups (all inverted, $True \to True$ in C_i becomes $False \to False$ in C_i') and no confliction between component-groups (disjoint). So it is an satisfiable assignment to $\tilde{G_F}$. Together with what we have done before $\tilde{G_F}$, we really have an assignment to make F satisfiable. ■

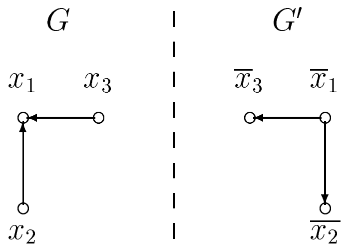
4 ANSWER TO PROBLEM 4

Proof

We already have $\overline{STCON}$ is complete, and thus hard, for NL. That means that $\forall L' \in NL, L' \leq_{logspace} \overline{STCON}$. I am going to show that $\overline{STCON} \leq_{logspace} 2-SAT$. Because $\leq_{logspace}$ is transitive, there is $\forall L' \in NL, L' \leq_{logspace} 2-SAT \Rightarrow 2-SAT$ is hard for NL.

To show $\overline{STCON} \leq_{logspace} 2-SAT$, I need a function f on the strings $< G, s, t >$ and $f(< G, s, t >) = F$, s.t., f is logspace computable and $< G, s, t > \in \overline{STCON} \Leftrightarrow F \in 2-SAT$. Here is the function f.

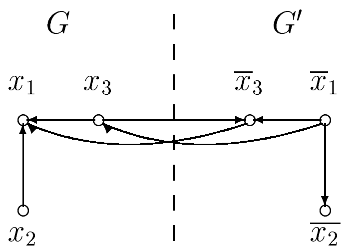
For directed graph $G = (V, E)$, $V = \{x_i\}$ and $E = \{(x_i, x_j)\}$, it has a "mirror" directed graph $G' = (V', E')$, $V' = \{\overline{x}_i\}$ and $E' = \{(\overline{x}_j, \overline{x}_i)\}$. For example as following figure, $V = \{x_1, x_2, x_3\}$, $V' = \{\overline{x}_1, \overline{x}_2, \overline{x}_3\}$, $E = \{(x_2, x_1), (x_3, x_1)\}$, $E' = \{(\overline{x}_1, \overline{x}_3), (\overline{x}_1, \overline{x}_2)\}$



F is constructed based on $G_F = G \cup G'$. F contains all the nodes in G_F as literals, and if there is an edge (x_i, x_j) in G_F, F will have a clause $\overline{x_i} \vee x_j$. For the figure above, $F = (\overline{x}_3 \vee x_1) \wedge (\overline{x}_2 \vee x_1)$.

Note that such transformation requires only $log(N)$ space, $N = |V|$, because it checks all the edges one by one, each time it requires spaces to store the two nodes. For $|V| = N$, two nodes take $2log(N)$ bits space. This is the first part of our f.

Now for the string $< G, s, t >$ we are concerning, f adds three additional edges to G_F: $(t, \overline{t})$, $(\overline{t}, s)$, and $(\overline{s}, t)$ (note: $(\overline{t}, s)$ implies $(\overline{s}, t)$) and respectively, F has two more clauses: $\overline{t} \vee \overline{t}$ and $t \vee s$. For our example, if $s = x_1, t = x_3$, G_F looks like this. $F = (\overline{x}_3 \vee x_1) \wedge (\overline{x}_2 \vee x_1) \wedge (\overline{x}_3 \vee \overline{x}_3) \wedge (x_3 \vee x_1)$. This additionally requires $2log(N)$ spaces.

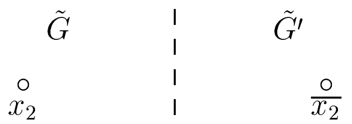


Here is our f, we can see that it takes log space and we will see that it is what we want for the reducibility. We have to show that $< G, s, t > \in \overline{STCON} \Leftrightarrow F \in 2 - SAT$. In other words, "there is no directed path from s to t in G" $\Leftrightarrow$ "F is satisfiable".

"$\Leftarrow$" It equals to prove "if there is a path from s to t, then F is unsatisfiable". That is obvious, because we have already have a path $t \to \overline{t}$ (added by f) in G_F and if there is a path from s to t, then there will be a path $\overline{t} \to s \to ... \to t$ (the first part is added by f). Thus t is a "bad point" in G_F and use the result of Problem 3, F is unsatisfiable.

"$\Rightarrow$" First we note that before f adds three edges ($t \to \overline{t}$, $\overline{t} \to s$ and $\overline{s} \to t$) to G_F, there are no paths from G to G'. That means there are no paths from x to $\overline{x}$ and vice versa. After we added these three edges, there may be some paths from x to $\overline{x}$ and vice versa, but they must pass through one of the three edges (they are the only "bridges" between the islands).

Similar as we do in Problem 3, we set $t = False$ (thus $\overline{t} = True$) and $s = True$. Then we set all ancestor nodes of t to $False$ and all descendant nodes of s to $True$. After that, we remove all the nodes and related edges from G_F. In our example, $x_3 = False$, $x_1 = True$, there left only x_2 .



Let us see what we have now: we have truncated graph $\tilde{G}$ and $\tilde{G'}$, where $\tilde{G}$ only has nodes $\{x_i\}$ and $\tilde{G'}$ only has $\{\overline{x}_i\}$, and more importantly, there are no paths between $\tilde{G}$ and $\tilde{G'}$ (the only three "bridges" have been removed). So we can set the remaining literals $x_i = True$. There will not be confliction and thus F is satisfied.

So far we have completed the proof of $\overline{STCON} \leq_{logspace} 2 - SAT$. Therefore $2 - SAT$ is hard for NL. ■