Mahya Jamshidian
9525133

Homework 2

Computational Complexity
May 17, 2019

Problem 0

First we prove B $\leq_p$ A $\cup$ B: Let s be a string that is in A. To test whether string s $\in$ A $\cup$ B, first test (in polynomial time) whether s $\in$ B. If it is,Then s$\in A \cup B$ iff s $\in$ A (since they are disjoint sets).

Now let B = Σ^*; for any NP-Complete language like A, A $\cup$ B = $\Sigma^* \rightarrow$ A is in P $\rightarrow$ P = NP.

Problem 1

Given an instance I of 2-SAT, we construct a directed graph G=(V,E) with vertices V labeled by the literals x and $\neg x$ for each variable x appearing in I. This graph has an edge(a,b)in E directed from vertex a to vertex b if the clause $(\neg a \vee b)$ is in I. If $(\neg a \vee b)$ is in I,so is $(b \vee \neg b)$. Thus, if(a,b) $\in$ E,then($\neg$ b,$\neg$ a) $\in$ E also.

We now show that an instance I is a No instance if and only if there is a variable x such that there is a path in G from x to $\neg x$ and one from $\neg x$ to x. If there is a variable x such that such paths exists, this means that $(x \rightarrow \neg x)$ and $(\neg x \rightarrow x)$ which is a logical contradiction. This implies that the instance I is a No instance.

Conversely, suppose I is a No instance. To prove there is a variable x such that there are paths from vertex x to vertex $\neg x$ and from $\neg x$ to x, assume that for no variable x does this condition hold and show that I is a Yes instance, that is, every clause is satisfied, which contradicts the assumption that I is a No instance.

Identify a variable α that has not been assigned a value and let α be one of the two cor-responding literals such that there is no directed path in G from the vertex α to $\neg\alpha$.(By assumption, this must hold for at least one of the two literals associated with x.) Assign value 1 to α and each literal γ reachable from it. (This assigns values to the variables identified by these literals.)

If these assignments can be made without assigning a variable both values 0 and 1, each clause can be satisfied and Iis Yes instance rather than a No one, asassumed. To show that each variable is assigned a single value, we assume the converse and show that the conditions under which values are assigned to variables by this procedure are contradicted. A variable can be assigned contradictory values in two ways: a) on the current step the literals and $\neg\gamma$ are both reachable from α and assigned value 1,

and b) a literal γ is reachable from α on the current step that was assigned value 0 on a previous step. For the first case to happen, there must be a path from α to verticesγ and$\neg\gamma$.

By design of the graph, if there is a path from α to $\neg\gamma$, there is a path from γ to$\neg\alpha$. Since there is a path from α to γ , there must be a path from $\neg\gamma$ to $\neg\alpha$, contradicting the assumption that there are no such paths. In the second case, let a γ be assigned 1 on the current step that was assigned 0 on a previous step.

It follows that $\neg\gamma$ was given value 1 on that step. Because there is a path from α to γ, there is one from $\neg\gamma$ to $\neg\gamma$ and our procedure, which assigned $\neg\gamma$ value 1 on the earlier step, must have assigned α value 1 on that step also. Thus, α had the value 0 before the current step, contradicting the assumption that it was not assigned a value.

To show that $2 - SAT$ is in NL, recall that NL is closed under complements. Thus, it suffices to show that No instances of $2 - SAT$ can be accepted in nondeterministic logarithmicspace. By the

above argument, if I is a No instance, there is a variable x such that there is a path in G from x to$\neg x$ and from $\neg x$ to x. Since the number of vertices in G is at most linear in n, the length of I, an NDTM can propose and then verifyin space O(log(n)) a path in G from x to $\neg x$ and back by checking that the producede edges are edges of G,that x is the first and last vertex on the path, and that x is encountered before the end of the path.

Problem 2

a) $L^{'}$ is in DSPACE(n) :
TM $M^{'}$ for language $L^{'}$ operates as :

- it first checks if the input is in the form $x\$^{|x|^2-|x|}$
- then uses TM M for language L to verify if it is in L.

input size for $M^{'}$ is $m = n^2$
$1 \rightarrow$ takes m space
$2 \rightarrow n^2$ space by defenition of $L = L(M)$
$\longrightarrow M^{'}$ uses 2*m space which is linear w.r.t it input $\longrightarrow L^{'} \in$ DSPACE(n).

$L^{'} \in \mathbf{NP} \rightarrow L \in \mathbf{NP}$:

since $L^{'}$ is in **NP** there exists some certificate $u^{'}$ such that $M^{'}(x\$|x|^2 - |x|,u^{'}) =$ YES;
since this behaving according to certificate takes polynomial time, and when it outputs YES both format of input and x being in L are verified then it uses some part of $u^{'}$ as certificate for checking wether x in L or not. Now that we knowing verifying the format takes polynomial time and does not act non-deterministicly, then the process of verifying x takes polynomial given $u^{'}$; Thus, $u^{'}$ acts as certificate for x and M and L $\in$ **NP**.

b) Using diagonalization for space hierarchy DSPACE(n) $\subsetneq$ DSPACE(n^2).
therefor there exists language like L which is in DSPACE(n^2) and in **NP** and not in DSPACE(n); now we can construct $L^{'}$ and since both L and $L^{'}$ are in **NP** and $L^{'}$ is in DSAPCE(n) but L is not $\longrightarrow$ DSPACE(n) $\neq$ **NP**.

Problem 3

An NP machine can be easily obtained from M by nondeterministically guessing the answers of oracle B_0 and aborting all computation paths at a suitable polynomial length. Since M has the same answer for all oracles this **NP** machine will not accept x if it is not in L_0. On the other hand, if $x \in \mathbf{L}$ then the path of correct guesses to the queries to B_0 will result in an accepting computation of polynomial length.

Problem 4

We want to find an oracle B such that NP^B
P^B is not empty. For any oracle B, define language L_B as follows: $L_B = \{1^n \mid kl\ B \cap \{0,1\}^n \neq \emptyset\}$
It is immediate that $L_B \in \mathbf{NP^B}$ for any B. (On input 1n, guess $x \in \{0,1\}^n$ and submit it to the oracle; output 1 iff the oracle returns 1).
Claim For any deterministic, polynomial-time oracle machine M, there exists a language B such that M_B does not decide L_B.
Suppose no matter how M queries,B responds NO. Can M possibly say YES? No,because B would always respond NO if its empty, in which case M should say NO. We will construct B such that the real answer is YES.
Let an oracle machine be a Turing machine with access to some fixed oracle,B. Let M_1, M_2, M_3, ...be all possible oracle machines. (We can enumerate them because Turing machines which can query oracles can be represented by finite strings of a finite number of characters, which there are countably many of. The oracle is fixed and separate from this list of machines.) Limit machines to decide L_B in time $2^n/10$ (note that thisis a weaker constraint than P). We want none of these machines to decide whether a string is in L_B within time $2^n/10$. Since there are 2^n strings of length n, and $2^n \geq 2^n/10$, there are necessarily strings of length n that are not queried by M_i. Thus to construct B: On step i, choose a strings of length much greater than the length of any string already in B, that is not queried by M_i.

- If M_i acceptss, do not add it to B.
- If M_i rejectss, add it to B.

Thus for no i does M_i recognize L_B.

Problem 5

Dividing the graph into two sets is the same as 2-coloring the graph such that all the edges are between vertices of different color. If the graph has an odd cycle, then the odd cycle in particular cannot be 2-colored and hence the graph cannot be bipartite. If the graph does not contain an odd cycle, we consider the DFS tree (or forest, if it is not connected) of the graph and color the alternate levels, red and blue. By construction, all the tree edges are between vertices of different color. Suppose an edge not in the tree connects two vertices of the same color. But these vertices must be connected by a path of even length in the tree and this extra edge will create an odd cycle, which is not possible. Hence, the above 2-coloring is a valid one which proves that the graph is bipartite.
We show that $BIPARTITE \in \mathbf{coNL}$ or $\overline{BIPARTITE} \in \mathbf{NL}$, which suffices since $\mathbf{NL} = \mathbf{coNL}$. However, $BIPARTITE$ is precisely the set of all graphs which contain an odd cycle. We guess a starting vertex v, guess an (odd) cycle length l and go for l steps from v, guessing the next vertex in the cycle at each step. If we come back to v (we can remember the starting vertex in logspace), we found a tour of odd length. Since any odd tour must contain an odd (simple) cycle, we accept and declare that the graph is non-bipartite.

Problem 6

If the graph contains a cycle,there must exist at least one vertex u and at least one starting edge (u, v) such that if we start the traversal through (u, v), we will come back to u through an edge different that (u, v). Hence, we enumerate all the vertices and all the edges incident on them, and start a traversal through each one of them. If we come back to the starting vertex through an edge different than the one we started on, we declare that the graph contains a cycle. Since we can enumerate all vertices and edges in logspace and also remember the starting vertex and edge using logarithmic space, this algorithm shows $CYCLE \in L$.

Problem 7

a) to compute A_{ij}, we maintain a counter t, read A_{it}, B_{tj}, compute $A_{it} \wedge B_{tj}$ and take an OR with $\bigvee_{k=1}^{t-1}$ which we can keep stored. Since maintaining the counter takes logarithmic space and all boolean operations take constant space, we can compute AB in logspace.
b) To multiplykmatrices, we generate the result entry by entry, by running a countertand generating the it entry in the product of the first $k1$ and the tj entry in thel ast matrix. Inductively, we need to maintain k counters which can be done in $O(klogn)$ space. Finally, note that using repeated squaring, we can compute A^p using O(log(p)) matrices, which are different powers of A. To generate each of these matrices, we just need A and a single counter. Hence the total space needed isO(log(p)log(n)).
c) We argue this by induction. The case k= 1 is easy since Aij= 1 iff there is an edge between i and j. Assuming it to be true for $k1$, note that $A_{ij}^k = \bigvee_{t=1}^n A_{it}^{k1} \wedge A_{tj}$, which is 1 iff for some t, $A_{it}^{k1} =$ 1 and A_{tj}= 1. However, this means there is a path of length $k1$ from i to t and an edge from t to j, which gives a path of length k from i to j.
d) Since $PATH$ is **NL−complete** and we can check if there is a path (which can be of lengthat most n) between any two vertices in a graph by computing A^n in O($log^2 n$) space,**NL** $\subseteq SPACE(log^2 n)$.

Problem 8

We construct a **NL**-machine for Shortest Path as follows: on inputG, k, s, t, first compute r_{k1} (the number of vertices reachable from s in at most $k1$ steps). Then:
consider the variable d for each vertex w in V :
—-starting from s we move forward for k-1 length to p:
—-if p is w : some path from s to w has length k : d++
———if w is t : NO : since we found a path from s to t with length less than k
———if w is one step away from t : there is a path of length k from s to t : f = TRUE
if d ¡ r_k : NO since it contracts with definition of r_k
if f == TRUE : YES we found all the paths from s to nodes with length less than k and t is not one of them while there is a path of length k from s to t
else NO there was no such path of length k from s to t

Problem 9

a) Let M be the machine that decides A in time n^6. Now, consider the machine $M^{'}$ for $pad(A, n^2)$ that on input x, check if x is of the format $pad(w, |w|^2)$ for some string $w \in \Sigma^*$. If not, NO. Otherwise, simulate M on w. The running time of $M^{'}$ is $\mathrm{O}(|x|^3)$ $+\mathrm{O}(|w|^6) = \mathrm{O}(|x|^3)$.
b) Suppose that $\mathbf{P} = \mathbf{NP}$. Then, consider any language $\mathbf{L} \in \mathbf{NEXP}$, and let c be a positive integer such that $L \in NTIME(2^{n^c})$. Then,it is easy to see that $pad(L, 2^{n^c}) \in \mathbf{NP}$. By assumption, $\mathbf{P} = \mathbf{NP}$, so $pad(L, 2^{n^c}) \in \mathbf{P}$ and therefore $L \in TIME(2^{O(n^c)}) \subseteq \mathbf{EXP}$. It follows that **EXP**=**NEXP**.