

CS331: Homework #9

Due on April 18, 2013 at 11:59pm

Professor Zhang 9:00am

Josh Davis

Problem 1

Part A Prove that $n! \in O(n^n)$.

Proof. To prove that $n! \in O(n^n)$, we will use the definition of big-O which states: For two functions, f and g . Say that $f(n) = O(g(n))$ if positive integers c and n_0 exist such that for every integer $n \geq n_0$ that $f(n) \leq cg(n)$.

Thus we need to find a c and n_0 to satisfy the above condition.

Let $f(n) = n!$ and $g(n) = n^n$. Let $c = 1$ and $n_0 = 1$, so for all $n \geq n_0$ we have:

$$\begin{aligned}
 n! &= (n)(n-1)(n-2) \dots (2)(1) \\
 &\leq c(n)(n-1)(n-2) \dots (2)(1) \\
 &= cn^n
 \end{aligned}$$

Since we have satisfied the definition of big-O, we can see that $n! \in O(n^n)$ and thus our proof is complete. \square

Part B Which of the following relations is true and which is false?

1. $n \in O((\lg n)^3)$

False

Linear functions outgrow polylogarithmic asymptotically. No such c or n_0 to satisfy big-O definition.

2. $(\lg n)^3 \in o(n)$

True

$$\lim_{n \rightarrow \infty} \frac{(\lg n)^3}{n} = 0$$

3. $n^{\lg n} \in O(2^{n \lg n})$

True

$$n^{\lg n} \leq c 2^{n \lg n}$$

$$\lg n \lg n \leq c(\lg 2)(n \lg n)$$

$$\lg n \lg n \leq c(n \lg n)$$

$$\lg n \leq cn$$

We can pick $c = 1$ and $n_0 = 1$, thus it is true.

4. $n^4 \in o(100n^4)$

False

$$\lim_{n \rightarrow \infty} \frac{n^4}{100n^4} = \lim_{n \rightarrow \infty} \frac{1}{100} = \frac{1}{100}$$

5. $(\lg n)^n \in O(\sqrt{2^n})$

False

$$n(\lg n)^n \leq c\sqrt{2^n}$$

$$n(\lg \lg n) \leq \frac{c}{2} \lg(2^n)$$

$$n(\lg \lg n) \leq \frac{c}{2} n$$

$$(\lg \lg n) \leq \frac{c}{2}$$

Which is clearly false, no constant is less than any function with n .

Problem 2

Prove that any language in P is **polynomial reducible** to any language in P which is not \emptyset or Σ^* .

Proof. To prove that any language in P is polynomial reducible to any other language in P , we will do a proof by construction to construct a polynomial time mapping function f that maps one language to another.

Let there be two languages A and B that are both in P . Since they are in P we know there are two TMs that run in polynomial time. Let these TMs be M_1 and M_2 .

We will now construct a polynomial reducible function where $w \in A \iff f(w) \in B$. Let this function be computed by a TM N which is as follows:

$N =$ “ On input $\langle M, w \rangle$ where M is a TM and w a string:

1. Construct a new TM, M' :

$M' =$ “ On input w :

- (a) Compute M_1 on input w to find a value in A
- (b) Simulate M_2 on newly computed value outputting what M_2 does to find a value in B

2. Output $\langle M' \rangle$.”

As we can see, since A and B are both in P , we can use the TMs that recognize the languages to reduce A to B .

Note: \emptyset and Σ^* can't be reduced to. If we have a language A that has some w that it accepts and some that it rejects, trying to reduce to \emptyset is impossible because since there exists a $w \in A$, there is no way to map it to \emptyset because the condition $w \in A \iff f(w) \in B$ doesn't hold. The converse is true for Σ^* as well. If $w \notin A$ we can't map it to a value that isn't in Σ^* . Thus we exclude these two languages from our proof. \square

Problem 3

Let $L = \{0^i 1^j : i > j\}$. Show that $L \in TIME(n \lg n)$.

Proof. To show that $L \in TIME(n \lg n)$, we will use a proof by construction to construct a new TM M that only uses $TIME(n \lg n)$ steps to complete.

$M =$ “ On input string w :

1. Start on the left side of the word, scan left, if there is a 1 before a 0, *reject*
2. Repeat as long as there are 0s still on the tape:
 - (a) Scan across the tape, if there are 0s left and no more 1s left, *accept*
 - (b) Scan again, crossing off every other 0 and every other 1
3. Since there are no more 0s, and we haven't accepted, we know $i \leq j$, thus *reject*.”

Let's analyze the running time of M . First observe that every stage except for the last run in $TIME(n)$. Stage 2 halves the number of characters in the input every time it repeats thus it runs in $TIME(n) * TIME(\lg n)$. The total running time is then $TIME(n) + TIME(n \lg n)$. This gives us the final running time of $TIME(n \lg n)$. \square

Problem 4

Prove that Graph isomorphism is in NP . That is

$$GI = \{\langle G, H \rangle : G, H \text{ are isomorphic}\} \in NP$$

Two graphs $G = \langle V_G, E_G \rangle$ and $H = \langle V_H, E_H \rangle$ are isomorphic iff there is a bijection $f : V_G \rightarrow V_H$ such that $\langle v, v' \rangle \in E_G$ if and only if $\langle f(v), f(v') \rangle \in E_H$.

Proof. To show that graph isomorphism is in NP , we will construct a verifier that can verify the problem in polynomial time. This proves that a language is in NP because of **Theorem 7.20**.

The verifier V that verifies that a graph is isomorphic is as follows:

$V =$ “ On input $\langle \langle G, H \rangle, c \rangle$:

1. Check that c is a valid bijection from nodes in $G \rightarrow H$
2. For every $(v_1, v_2) \in c$:
 - (a) Check that $v_1 \in G$
 - (b) Check that $v_2 \in H$
3. If everything passes, *accept*, otherwise *reject*.”

We can see that given our input, $\langle G, H \rangle$ where $n = |G|$ and $m = |H|$, we can see that the verifier's time complexity is running in $O(m + n)$ time, or $2n = O(n)$ because the number of nodes in the graphs must be equal.

Thus we have proven that Graph Isomorphism is in NP . □

Problem 5

Prove that Double Satisfaction Problem, defined as

$$SAT2 = \{\langle \phi \rangle : \phi \text{ is a 3CNF-formula with at least two solutions}\}$$

is NP -complete.

Proof. To prove that $SAT2$ is NP -complete, we will first show that it is in NP and then reduce $3SAT$ to it. Similar to what is done in **Corollary 7.42** in the textbook.

To show that $SAT2$ is in NP , we can just create a TM that guesses two assignments to all the variables and then tries them. This TM would be nondeterministic but could run in polynomial time. Thus $SAT2 \in NP$.

Given a $\phi \in 3SAT$, we know that ϕ is a 3cnf-formula that is satisfiable once. Now we can just make a dummy variable say j and add it along with another variable x so that $\phi' = \phi \wedge (j \vee \bar{j} \vee x)$. Thus we can see that for any assignment of j , it is satisfied twice because it was already satisfied once.

This construction is very easily polynomial time. Now we need to show that if $\phi \notin 3SAT$, using our reduction, $\phi' \notin SAT2$.

Given a $\phi \notin 3SAT$, we know that there is no such assignment that satisfies it. Thus when using our reduction, $\phi' = \phi \wedge (j \vee \bar{j} \vee x)$. This will never result in having two satisfactions for it because the previous part is not satisfied.

Thus we have shown that $SAT2$ is NP -complete and our proof is finished. \square

Problem 6

Prove that the class P is closed under union and complementation.

Part One Prove that P is closed under union.

Proof. To prove that P is closed under union, we assume that there are two languages L_1 and L_2 with M_1 and M_2 as TMs that decide them.

Assume that M_1 runs in polynomial time, $O(n^x)$ as well as M_2 , $O(n^y)$.

We will construct a new TM M that runs both M_1 and M_2 and show that it still runs in polynomial time.

$M =$ “ On input w :

1. Run M_1 on w .
2. Run M_2 on w .
3. If one of the TMs accepted, accept, else reject.”

The runtime of M can be determined as $O(n^x) + O(n^y)$. Asymptotically, this is equal to the following: $O(n^z)$ where $z = \max(x, y)$.

Thus we can see that P is closed under union. □

Part Two Prove that P is closed under complementation.

Proof. To prove that P is closed under complementation, we assume that there is a language L with a TM M that decides it.

Assume that M runs in polynomial time, $O(n^x)$.

We will construct a new TM N that runs M and we will show that it still runs in polynomial time.

$N =$ “ On input w :

1. Run M on w .
2. If M accepted, reject, else accept.”

The construction of N is such that it decides the complement of the language that M decides. This TM also runs in $O(n^x)$, which means it still runs in polynomial time.

Thus we can see that P is closed under complementation. □
