**Harvard University**
**Computer Science 121**

**Problem Set 9**

Due November 24, 2015 11:59pm

Problem set by Kevin Zhang

Collaboration Statement: I collaborated with no one, got help from no one else, and used no other resources.

## PART A (Graded by Juan and Geoff)

PROBLEM 1 (5 points, suggested length of 1/3 of a page)

Show that if P = NP then every non-trivial language $L \in$ NP is NP-Complete. (Non-trivial languages are languages other than $\Sigma^*, \emptyset$)

**Solution.**
Assume that P = NP. Let $A$ be a non-trivial language in NP. Choose $x \in A$ and $y \notin A$; this is possible since $A$ is non-trivial. We want to show that $A$ is NP-complete. Let $B$ be an arbitrary language in NP. We can reduce $B$ to $A$ simply by running $B$ on a given input $w$ and determining in polynomial time (since $B$ is in NP = P) whether $B$ accepts $w$; if it does, output $x$ - otherwise, output $y$. This provides a satisfactory mapping function from $B$ to $A$, showing that every language in NP is reducible to $A$; i.e. $A$ is NP-complete.

PROBLEM 2 (10 points, suggested length of 1 page)

A *pattern* is a string over the alphabet $\{0, 1, ?\}$. Say that a pattern $\pi$ *covers* a string $w$ over $\{0, 1\}$ if $w$ can be obtained from $\pi$ by replacing each occurrence of ? in $\pi$ with either 0 or 1. (For example, 0??1 covers the four strings 0001, 0011, 0101, 0111.)

We define the problem PATTERN as the set $\{\Pi : \Pi$ is a set of patterns, each of length $n$, such that there exists a string $s$ of length $n$ over $\{0, 1\}$ where no pattern in $\Pi$ covers $s\}$.

Show that PATTERN is $\mathcal{NP}$-complete. (*Hint*: reduce from $n$-variable 3SAT.)

**Solution.**
First we show that PATTERN is in NP; we do this by constructing a polynomial-time verifier. The verifier takes a certificate that is just a string of $n$ 0's and 1's; verification consists simply of checking that none of the patterns covers the given string, which takes $O(nm)$ time, where there are $m$ patterns in the set. This takes polynomial time, since there are at most $3^n$ possible patterns and so $n$ and $m$ are both finite integers.

Now, we reduce 3SAT to PATTERN; since 3SAT is NP-complete, this reduction would show that PATTERN is NP-complete as well. We want to map an input $c$ to 3SAT to an input $S$ of PATTERN such that 3SAT accepts $c$ if and only if PATTERN accepts $S$. We compute the inverse of

a Boolean formula using De Morgan's Laws, compute the corresponding set of patterns, and show that $c$ is satisfiable if and only if this set of patterns is in PATTERN.

We first use De Morgan's Laws to construct the complement $c'$ of a formula $c$; note that the properties of De Morgan's Laws are such that because $c$ is in CNF form (an AND of many OR clauses), $c'$ will be an OR of many AND clauses. For example, the formula $(a \vee b \vee c) \wedge (\overline{a} \vee b \vee \overline{c})$ can be inverted to obtain $\overline{a}\overline{b}\overline{c} \vee a\overline{b}c$.

Next, we construct a set of patterns from a Boolean formula consisting of many AND clauses OR'd together. Consider the 3SAT problem with $n$ variables. Let those variables be $x_1, x_2, \cdots x_n$. Suppose we have $c'$ as above so that $c'$ consists of AND clauses joined by ORs. We transform $c'$ into a set of patterns $S$ as follows: for each AND clause in $c'$, construct a pattern as follows:

- For each of $i = 1$ through $n$, if $x_i$ is:
  - in the AND clause as $x_i$, the $i$th character in the pattern is a 1
  - in the AND clause as $\overline{x_i}$, the $i$th character in the pattern is a 0
  - not in the AND clause, the $i$th character in the pattern is a ?

The set of patterns $S$ consists of all patterns constructed from the clauses in $c'$ as above. For example, the CNF formula $(x_1 \vee x_2 \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee \overline{x_3})$ is transformed first into $\overline{x_1}\overline{x_2}\overline{x_3} \vee x_1\overline{x_2}x_3$; these clauses are then transformed into the set of patterns $\{000, 101\}$.

Finally, we prove that an input $c \in$ 3SAT if and only if the corresponding set of patterns $S \in$ PATTERN. It should be clear that in the above construction, we have provided a way to essentially represent Boolean formulas as sets of patterns, and sets of variable values as strings of 0's and 1's. It should also be clear that if a set of patterns covers a string $s$, the corresponding formula for that set of patterns (e.g. $\overline{x_1}\overline{x_2}\overline{x_3} \vee x_1\overline{x_2}x_3$ for the set of patterns $\{000, 101\}$ above) is satisfied by the variable values represented by $s$. Then, given $c$, we have constructed a set of patterns corresponding to $\overline{c}$; if $c$ is unsatisfiable, then $\overline{c}$ is satisfiable by ANY string, which means that there are no strings that the set of patterns for $\overline{c}$ does not cover. Specifically, this means that if $c$ is unsatisfiable, the set of patterns we have constructed is not in PATTERN. Alternatively, if $c$ is satisfied by a set of values represented by a string $s$, the corresponding set of patterns for $\overline{c}$ does not cover $s$, and so this set is in PATTERN.

Note also that the above constructions take polynomial time; De Morgan's Laws take time proportional to the number of characters to execute, and constructing the set of patterns takes time proportional to the number of clauses multiplied by $n$. This is required to show NP-completeness through reduction. Thus, we have shown that 3SAT is reducible to PATTERN, and we are done.

PROBLEM 3 ((3)+5 points, suggested length of 1 page)

**Note: Part A of this problem is optional extra credit; Part B is required.**
If a divorce is friendly, the spouses often have the problem of trying to divide up the property evenly. If the divorce is hostile, instead of dividing the property two ways, the property is often divided three ways: between the first spouse, the second spouse, and the various lawyers. Define AMICABLE DIVORCE as follows: Given a set of numbers, is it possible to split the set into two subsets such that every element of the original set is in exactly one of the two subsets and the sum of each subset is the same? Define HOSTILE DIVORCE as follows: Given a set of numbers, is it possible to split the set into three subsets such that every element of the original set is in exactly one of the subsets and the sum of each subset is the same?

(A) (CHALLENGE! Extra Credit) Show that AMICABLE DIVORCE is NP-complete.

(B) Assuming that AMICABLE DIVORCE is NP-complete, show that HOSTILE DIVORCE is NP-complete.

**Solution.**

(A)

(B) It is obvious that both AMICABLE DIVORCE and HOSTILE DIVORCE are NP-complete since they have polynomial-time verifiers; the certificates in each case are the disjoint subsets, and verification consists simply of making sure that 1) all numbers from the original set are present exactly once across all sets and 2) the sum of the numbers in each set is equal. This takes time proportional to $n$, where $n$ is the cardinality of the set.

It remains to reduce AMICABLE DIVORCE to HOSTILE DIVORCE. Suppose we have a set of numbers $S$. Do the following:

- Calculate the sum of the set; if it is not even, reject.
- Iterate over the set again; if any of the elements are greater than half the above sum, reject because it is therefore impossible to divide the set into two equal-sum sets.
- Add an element with value equal to exactly half the above sum to the set.

The new set's sum is exactly 1.5 times the old set's; let's call it $S'$. Now, we pass this set to the HOSTILE DIVORCE problem. Note now that any division of the new set into 3 equal-sum parts MUST have one set with only one element, which is the element we added in the above process. Since that element has exactly 1/3 of the total sum of $S'$, it cannot be placed in a set with any other elements. But then each of the sums of the other two sets is exactly 1/3 of the total sum of $S'$ as well, and equal to 1/2 the total sum of $S$, the original set - therefore, these two sets satisfy AMICABLE DIVORCE for $S$.

Conversely, if $S'$ is not in HOSTILE DIVORCE, there exists no way to divide $S'$ into 3 equal-sum sets; there is therefore no way to divide $S$ into two equal-sum sets either, since if there was, those two sets in addition to a set containing an element exactly equal to the sum of each of the other two sets would satisfy HOSTILE DIVORCE for $S'$, a contradiction. The above construction takes time proportional to $n$, where $n$ is the size of the original set - the process simply iterates twice over the set. Hence, AMICABLE DIVORCE is reducible to HOSTILE DIVORCE in polynomial time, so HOSTILE DIVORCE is NP-complete if AMICABLE DIVORCE is.

## PART B (Graded by Varun and Zhengyu)

PROBLEM 4 (3+8+3 points, suggested length of 1 page)

A language is NP-*hard* if all problems in NP $\leq_p$-reduce to it. (Recall that NP-completeness also requires that the language is in $NP$.) Let:

$$A = \{\langle M \rangle \ : \ M \text{ is a DFA and } M \text{ accepts some string.}\}$$

$$B = \{\langle M_1, M_2, ..., M_k \rangle \ : \text{ Each } M_i \text{ is a DFA and all of the } M_i \text{ accept some common string.}\}$$

(A) Show that $A \in$ P.

(B) Show that $B$ is NP-hard by giving a reduction from some NP-complete problem to $B$.

(C) We can convert an instance of $B$ into an instance of $A$ by applying the product construction (See p.45-46 of *Sipser*) $k-1$ times in succession. Does this show that P = NP? Why or why not?

**Solution.**

(A) We construct a TM $T$ that decides $A$ in polynomial time. $T$ takes an input string $s$ that encodes a; DFA call the length of the string $n$. Since there are at most $n$ characters, there are at most $n$ transitions in the input TM. Our TM will simply start from the start state of the input DFA and search through the DFA along transitions until there are no more transitions available. To be exact, $T$ starts by marking the start state; afterwards, it continually searches for unvisited transitions away from any marked states and marks the states at the other ends of those transitions, until there are no more available transitions away from marked states. If $T$ encounters any accept states in this process, it immediately terminates and rejects the encoded DFA. Otherwise, if the process terminates without doing so, it accepts the encoded DFA. Since there are only $n$ transitions at most, the above process must terminate in time proportional to $n$ (since it can only take up to $n$ steps to traverse all possible transitions). This process clearly terminates, and so it must decide $A$; because it takes time proportional to $n$, $A \in$ P.
(B) We reduce 3SAT as follows: given an expression $c$ and a set of values for the variables $S$, we construct a set of DFAs that accept a string representing $S$ iff $c$ accepts $S$. Specifically, we construct a TM that acts as follows: given a Boolean expression in 3CNF form with $n$ variables (call them $x_1$ through $x_n$), we construct a DFA with alphabet $\{0, 1\}$ for each of the clauses in the expression as follows:

- Check to make sure the input string has length exactly $n$; otherwise, reject.
- Construct $n$ states, along with an extra dead state and an accept state that transitions to itself over $\Sigma$.
- Let the 1st state from the $n$ states be the start state.
- Create transitions from the $i$th numbered state as follows:
  - If $x_i$ is in the clause, create a transition on 1 to the accept state and a transition on 0 to the dead state.
  - If $\overline{x_i}$ is in the clause, create a transition on 0 to the accept state and a transition on 1 on the dead state.
  - If neither of the above are applicable, create a transition on $\Sigma$ to the $i+1$th numbered state (i.e. either 0 or 1 creates a transition to the $i+1$th numbered state); if there is no $i+1$th numbered state (i.e. $i = n$), transition to the accept state.

Each of these DFAs accept iff the input string matches the clause it was constructed from; e.g. if $n = 4$ and the clause was $x_1 \vee x_2 \vee \overline{x_4}$, the input string must match 11?0, where ? represents either 0 or 1. This construction takes polynomial time in the length of the Boolean expression; it takes constant time to construct each DFA for the clauses. Hence, we have reduced 3SAT to the problem of determining whether a string is accepted by a set of DFAs, and therefore the problem of determining whether a set of DFAs accept a common string is NP-hard.
(C) No; the reasoning behind proving that $A \in P$ relies on the fact that it takes polynomial time in the number of transitions to search through an input DFA for an accept state, but this is no longer true when you combine several DFAs together. In particular, if there are $n$ DFAs in a set, the number of states in the product construction is on the order of $k^n$, where $k$ is the average number of states in each of the DFAs (the exact value isn't necessary, only the observation that the number of states is exponential in the number of DFAs). Checking each of these states alone takes exponential time, making it impossible for the result of the product construction to be in $A$. Note: the reason why $P \neq NP$ here boils down to the use of different measurements for $n$; in $A$, we used the length of the input string as $n$, but in $B$ we used the number of DFAs.

PROBLEM 5 (6 points, suggested length of 1/2 page)

Let DoubleSAT = $\{\phi : \phi$ has at least two satisfying assignments$\}$. Show that DoubleSAT is NP-complete.

**Solution.**
We reduce SAT to DoubleSAT as follows: on an input expression $c = \phi(x_1, x_2, \cdots, x_n)$, add a variable $y$ and create the expression $\phi' = \phi(x_1, x_2, \cdots, x_n) \wedge (y \vee \overline{y})$. If $\phi$ had some satisfactory assignment for $x_i$, this assignment in conjunction with either $y = 1$ or $y = 0$ satisfies $\phi'$. Alternatively, if $\phi$ had no satisfactory assignment, then there is no way to satisfy $\phi'$ either (because $\phi$ is not satisfiable, so even if we can satisfy $(y \vee \overline{y})$, $\phi \wedge (y \vee \overline{y})$ is not satisfiable). Hence, $\phi'$ has either at least 2 satisfactory assignments or none, depending on whether the input expression is satisfiable or not. But this means that we can pass the constructed expression to DoubleSAT, and $\phi \in$ SAT iff $\phi' \in$ DoubleSAT.

PROBLEM 6 (8 points, suggested length of 1/2 page)

A strong nondeterministic TM is one that has three possible halt states: "yes", "no", or "maybe". We say such a machine decides $L$ in polynomial time iff all computations run in polynomial time, and if the following holds: if $x \in L$, then all computations end up with "yes" or "maybe," and at least one ends up "yes". If $x \notin L$, then all computations end up in "no" or "maybe", but at least one ends up "no". Prove that $L$ is decided by a strong nondeterministic TM in polynomial time if and only if $L \in \mathcal{NP} \cap$ co-$\mathcal{NP}$.

**Solution.**
Suppose $L$ is decided by a strong nondeterministic TM $T$ in polynomial time $n^k$. Modify $T$ such that whenever it would have halted in "maybe", it instead halts on "reject". This results in a non-deterministic TM that decides $L$, so $L$ is in NP. Similarly, we can also modify it by changing all halting "accept" states to halting "reject" states and vice versa, and changing "maybe" halt states to "reject" halt states. This instead non-deterministically decides $\overline{L}$, so $L$ is in co-NP as well.

Now suppose that $L$ is in NP and co-NP. This means that both $L$ and $\overline{L}$ have certificates that can be used to decide inputs in polynomial time. Specifically, there are two languages $A$ and $B$ such that $L = \{x : \exists c, (x, c) \in A\}$ and $\overline{L} = \{x : \exists c, (x, c) \in B\}$. We construct a strong NTM as follows: given an input string $s$, non-deterministically guess certificates $x$ for $A$ and $y$ for $B$. If $(s, x) \in A$, halt on "accept"; if $(s, y) \in B$, halt on "reject". Otherwise, halt on "maybe". If $s \in L$, then there is a certificate that is accepted by $A$ when paired with $s$, so at least one path will halt on "accept"; similarly, if $s \notin L$, then there is a certificate that is accepted by $B$ when paired with $s$, so at least one path will halt on "reject". Moreover, if $s \in L$, no path can halt on "reject" or that would imply that there exists a certificate for which $s$ is accepted by $B$ (and therefore $s \in \overline{L}$ as well, a contradiction). Similarly, if $s \notin L$, no path can halt on "accept". It should be fairly obvious that this strong NTM works as intended.