Theory of Computation, Homework 5

7.6
i.) union
Without loss of generality, let $M_1$ and $M_2$ be the TMs that decide two languages $L_1$ and $L_2$ in P. Then we construct a TM $M$ that decides the union of $L_1$ and $L_2$ in polynomial time.

$M =$ "on input $w$

1. Run $M_1$ on $w$. Accept if it accepts.
2. Run $M_2$ on $w$. Accept if it accepts.
3. Otherwise, reject."

Since stage1 and stage2 both run in polynomial time, TM $M$ can decide the union of $L_1$ and $L_2$ in polynomial time. Hence, $P$ is closed under union.

ii.) concatenation
Without loss of generality, let $M_1$ and $M_2$ be the TMs that decide two languages $L_1$ and $L_2$ in P. Then we construct a TM $M$ that decides the concatenation of $L_1$ and $L_2$ in polynomial time.

$M =$ "on input $w$

1. For each possible split of $w = w_1w_2$
   a. Run $M_1$ on $w_1$ and run $M_2$ on $w_2$. Accept if both accept.
2. Otherwise, reject."

Assume the $|w| = n$. There are $O(n)$ possible splits of $w$ at most. Besides, the sum of the time complexities of $M_1$ and $M_2$ is also polynomial. Thus, step1 run in polynomial time and TM $M$ can decide the concatenation of $L_1$ and $L_2$ in polynomial time. Hence, $P$ is closed under concatenation.

iii.) complement
Without loss of generality, let $M$ be the TM that decides a language $L$ in P. Then we construct a TM $M'$ that decides the complement of $L$ in polynomial time.

$M' =$ "on input $w$

1. Run $M$ on $w$
2. Accept if it rejects. Otherwise, reject if it accepts."

Since step1 run in polynomial time, TM $M'$ can decide the complement of $L$ in polynomial time. Hence, $P$ is closed under complement.
7.7
i.) union
Without loss of generality, let $N_1$ and $N_2$ be the NTMs that decide two languages $L_1$ and $L_2$ in NP. Then we construct an NTM $N$ that decides the union of $L_1$ and $L_2$ in polynomial time.

$N =$ "on input $w$
1. Run $N_1$ on $w$ nondeterministically. Accept if it accepts.
2. Run $N_2$ on $w$ nondeterministically. Accept if it accepts.
3. Otherwise, reject."

Since stage1 and stage2 both run in polynomial time nondeterministically, NTM $N$ can decide the union of $L_1$ and $L_2$ in polynomial time. Hence, $NP$ is closed under union.

ii.) concatenation
Without loss of generality, let $N_1$ and $N_2$ be the NTMs that decide two languages $L_1$ and $L_2$ in NP. Then we construct an NTM $N$ that decides the concatenation of $L_1$ and $L_2$ in polynomial time.

$N =$ "on input $w$
1. For each possible split of $w = w_1w_2$
   a. Run $N_1$ on $w_1$ nondeterministically and run $N_2$ on $w_2$ nondeterministically. Accept if both accept.
2. Otherwise, reject."

Assume the $|w| = n$. There are $O(n)$ possible splits of $w$ at most. Besides, the sum of the time complexities of $N_1$ and $N_2$ is also polynomial. Thus, step1 run in polynomial time and NTM $N$ can decide the concatenation of $L_1$ and $L_2$ in polynomial time. Hence, $NP$ is closed under concatenation.

7.12
Since $a^b$ is exponential in the length of $b$, we need to break down $a^b$ and apply the modulo operation several times so that the resulting size never goes exponential. First, we treat $b$ in its binary representation. There are at most $\log_b a$ bits. So

$$a^b \mod p = \left( \prod_i a^{b \cdot 2^i} \right) \mod p = \left( \prod_i (a^{b \cdot 2^i} \mod p) \right) \mod p$$

In the worst case, we assume $b_i = 1$ for all $i$. However,

$$a^{b_i \cdot 2^{i+1}} \mod p = (a^{b \cdot 2^i})^2 \mod p = (a^{b \cdot 2^i} \mod p)^2 \mod p.$$ That is, each term can be computed by squaring the remainder of the previous term $\mod p$. Hence, for each modulo operation, the computation is $O(p^2)$ which is in polynomial time. Additionally, for each multiplication of neighboring terms, we also apply the modulo operation in $O(p^2)$. Since there are at most $\log_b p$ terms, $a^b \equiv c \mod p$ is $O(2\log_b p^2)$. Therefore, $\text{MODEXP} \in P$. 

7.16
1.) In the proof of theorem 7.56, the SUBSET-SUM problem instance constructed contains numbers of large magnitude presented in decimal notation. However, compared with binary/decimal encoding, unary encoding will grow the input size exponentially. Therefore, the reduction to generate the corresponding input for UNARY-SSUM will require a number of steps that is exponential in the size of the input, taking more than polynomial time. The proof fails.

2.) SUBSET - SUM is in \( P \) because the following polynomial time algorithm based on dynamic programming solve the problem.

Input: \((S, t)\), where \(S = \{x_1, x_2, \ldots, x_k\}\)
Variables: Integers \(i, j, l[0..k][0..t]\), all are initialized to 0

\[
\begin{align*}
&l[0][0] \leftarrow 1; \\
&\text{for } i = 1 \text{ to } k \text{ do} \\
&\quad \text{for } j = 0 \text{ to } t - x_i \text{ do} \\
&\quad \quad \text{if } l[i-1][j] = 1 \text{ then } l[i][j + x_i] \leftarrow 1; \\
&\quad \text{od} \\
&\text{od} \\
&\text{if } l[k][t] = 1 \text{ then accept else reject;}
\end{align*}
\]

7.17
Assume that \( P = NP \). Let \( B \in P \) but \( B \neq \phi \) and \( B \neq \Sigma^* \) such that there exist a string \( w_{accept} \in B \) and a string \( w_{reject} \notin B \). To show \( B \) in \( P = NP \) is NP-complete, let \( A \) be an arbitrary language in \( P = NP \). Therefore, \( A \) can be decided by a polynomial time decider \( M_A \).

Next, we show a polynomial time reduction \( M \) from \( A \) to \( B \) as follows.

\( M \) = "On input \( w \):
\begin{enumerate}
1. Run \( M_A \) on \( w \).
2. If \( M_A \) accepts then output \( w_{accept} \). If \( M_A \) rejected then output \( w_{reject} \)."
\]

Obviously, there exists a polynomial time reduction from \( A \) to \( B \). Hence \( B \) is NP-complete.
7.20

a.)

SPATH is in P because the following TM M can decide it in polynomial time.

\[ M = "On input \langle G, a, b, k \rangle where G is a graph with n nodes, two which are a and b.
\]
\[ 1. \text{Mark node } a\text{ 0}
\]
\[ 2. \text{For each } i \text{ from 0 to } n:
\]
\[ \quad a. \text{If an edge } (s, t) \text{ is found connecting } s \text{ marked } i \text{ to an unmarked node } i, \text{ mark node } t \text{ with } i + 1.
\]
\[ 3. \text{If } b \text{ is marked with a value of at most } b, \text{ accept. Otherwise, reject."
\]

b.)

LPATH \in NP because a path from a to b can be guessed by an NTM nondeterministically and checked if its length is at least k in polynomial time.

Next, we show UHAMPATH \leq_p LPATH by constructing a TM F that computes the reduction f.

\[ F = "On input \langle G, a, b \rangle of UHAMPATH, where G is a graph with two nodes a and b.
\]
\[ 1. \text{Let } k \text{ be the number of nodes of } G.
\]
\[ 2. \text{Output } \langle G, a, b, k \rangle \text{ as an instance of } LPATH"
\]

If \langle G, a, b \rangle \in UHAMPATH, then G has a Hamiltonian path of length k from a to b. That is, \langle G, a, b, k \rangle \in LPATH. On the other hand, if \langle G, a, b, k \rangle \in LPATH, then G must contain a simple path of length k from a to b. However, since G has only k nodes, the path must be Hamiltonian. There, \langle G, a, b \rangle \in UHAMPATH. Hence, UHAMPATH \leq_p LPATH.

According to the above, LPATH is NP-Complete.
7.21

i.) \textit{DOUBLE} – SAT \in \textit{NP}

On input of a Boolean formula, an NTM could guess two different Boolean assignments nondeterministically and accept if both of the assignments satisfy the formula. Hence, \textit{DOUBLE} – SAT is in \textit{NP}.

ii.) \textit{DOUBLE} – SAT is NP-hard

\textit{SAT} can be reduced to \textit{DOUBLE} – SAT by constructing a TM \( F \) that computes the reduction \( f \) as follows.

\( F = \text{"On input of a Boolean formula } (\phi) \text{ with variables } x_1, x_2, ..., x_k \)

1. Let \( \phi' = \phi \land (x \lor \overline{x}) \) by introducing a new variable \( x \)
2. Output \( (\phi') \) with variables \( x_1, x_2, ..., x_k, x \)

If \( \phi \in \textit{SAT} \), its satisfying assignment plus \( x = \text{true} \) or \( \overline{x} = \text{true} \) can serve as the two satisfying assignments of \( \phi' \). On the other hand, if \( \phi' \in \textit{DOUBLE} – \textit{SAT} \) is satisfiable, \( \phi \) can be satisfied by the assignment where the new variables \( x \) and \( \overline{x} \) are removed. Therefore, \textit{DOUBLE} – \textit{SAT} is NP-hard.

According to i. and ii., \textit{DOUBLE} – \textit{SAT} is NP-Complete.

7.24

a.)

In a clause of an \( \neq \) assignment, there is one literal set true or two literals set true. Thus, the negation of a the clause will still induce a clause with one literal set true or two literals set true. Since a negated clause is also a clause of an \( \neq \) assignment, the negation of an \( \neq \) assignment is too an \( \neq \) assignment.

b.)

Let \( \phi \) be a \textit{3cnf} formula of an instance of \textit{3SAT} and \( \phi' \) be a \textit{3cnf} formula of an instance of \( \neq \textit{SAT} \). The reduction of replacing a clause \( c_i (y_1 \lor y_2 \lor y_3) \) of \( \phi \) with two clauses \( (y_1 \lor y_2 \lor z_i) \land (\overline{z_i} \lor y_3 \lor b) \) of \( \phi' \) can be computed in polynomial time for sure. Next, we show \( \phi \) is satisfiable iff \( \phi' \) has the corresponding \( \neq \) assignment.

If \( \phi \) is satisfiable, either \( y_1 = \text{true} \) and \( y_2 = \text{true} \), or only \( y_3 = \text{true} \). In the former case, set \( z_i \) false. In the latter case, set \( z_i \) true. In any case, \( b \) is set false. On the other hand, if \( \phi' \) has an \( \neq \) assignment, \( y_1, y_2 \) and \( y_3 \) cannot all be set false since this may cause one of the clauses in the \( \neq \) assignment to be false. Thus, the corresponding clause \( (y_1 \lor y_2 \lor y_3) \) must be true. A satisfying assignment of \( \phi \) is then derived.

c.)

Obviously, \( \neq \textit{SAT} \) is in \textit{NP} because an NTM can guess an \( \neq \) assignment and check if it satisfies the formula nondeterministically in polynomial time. According to the reduction in b, \( \neq \textit{SAT} \) is concluded to be NP-Complete.
7.28

i.) \textit{SET} – \textit{SPLITTING} \in NP

On input of \((S, C)\), where \(S\) is a finite set and \(C = \{C_1, C_2, ..., C_k\}\) is a collection of subsets of \(S\), for some \(k > 0\), an NTM could guess a coloring of the elements nondeterministically and accept if no \(C_i\) has all its elements colored with the same color. Hence, \textit{SET} – \textit{SPLITTING} is in \(NP\).

ii.) \textit{SET} – \textit{SPLITTING} is NP-hard

3\textit{SAT} can be reduced to \textit{SET} – \textit{SPLITTING} by constructing a TM \(F\) that computes the reduction \(f\) as follows.

\(F = \) "On input of a 3CNF Boolean formula \((\phi)\) with \(l\) clauses and variables \(x_1, \bar{x}_1, x_2, \bar{x}_2, ..., x_k, \bar{x}_k\)

\begin{enumerate}
\item Set \(S = \{x_1, \bar{x}_1, x_2, \bar{x}_2, ..., x_k, \bar{x}_k, z\}\)
\item For each clause \(c_i\) in \(\phi\), add a subset \(C_i\) of \(S\) including \(z\) and the literals in \(c_i\) as the elements.
\item add a subset \(C_{x_i}\) for each pair of \(x_i\ and \ \bar{x}_i\"
\end{enumerate}

Clearly, \(C = \{C_1, C_2, ..., C_l, C_{x_1}, C_{x_2}, ..., C_{x_k}\}\).

If \(\phi \in SAT\), its satisfying assignment can serve as a coloring of \((S, C)\) by coloring all the true literals red and all the false literals as well as \(z\) blue such that each subset \(C_i\) has at least one blue element \(z\) and one red literal which has a true assignment. For subset \(C_{x_i}\), the pair of literals \(x_i\ and \ \bar{x}_i\) must be colored exactly red for one and blue for the other. Obviously, \((S, C) \in \textit{SET} – \textit{SPLITTING}\).

On the other hand, if \((S, C) \in \textit{SET} – \textit{SPLITTING}\) has a coloring, \(\phi\) can be satisfied by the assignment where red literals are set true while blue literals are set false. Therefore, \textit{SET} – \textit{SPLITTING} is NP-hard.

According to i. and ii., \textit{SET} – \textit{SPLITTING} is NP-Complete.