Problem 7.7

NP is closed under union. For any two NP-languages L_1 and L_2, let M_1 and M_2 be the NTMs that decide them in polynomial time. We construct a NTM M' that decides the union of L_1 and L_2 in polynomial time:

M' = "On input w
1. Run M_1 on w. If it accepts, accept.
2. Run M_2 on w. If it accepts, accept. Otherwise, reject"

M' accepts w if and only if either M_1 and M_2 accepts w. Therefore, M' decides the union of L_1 and L_2. Since both steps take polynomial time, the algorithm runs in polynomial time.

NP is closed under concatenation. For any two NP-languages L_1 and L_2, let M_1 and M_2 be the NTMs that decide them in polynomial time. We construct a NTM M' that decides the concatenation of L_1 and L_2 in polynomial time.

M' = "On input w
1. Nondeterministically cut w into two substrings $w = w_1w_2$
2. Run M_1 on w_1
3. Run M_2 on w_2. If both accept, accept otherwise continue with the next choice of w_1 and w_2"

In both steps, M' uses its non determinism when the machine is being run. M' accepts w if and only if w can be expressed as w_1w_2 such that M_1 accepts w_1 and M_2 accepts w_2. Therefore, M' decides the concatenation of L_1 and L_2. Since step 2 and 3 runs in polynomial time and is repeated for at most $O(n)$ time, the algorithm runs in polynomial time.

Problem 7.10

Note that it takes linear time to decide if there is a path from one vertex to another vertex in a graph. A TM M that decides ALL_{DFA} in polynomial time operates as follows:

M = "On input M where M is a DFA:
1. Construct directed graph G=(Q, E), where E contains (q, p) if p is a next state of q.
2. Test whether a path exists from the start state to one non-accept state in G.
3. If no such path exists, accept; otherwise, reject."

Problem 7.13

Let $MODEXP = \{\langle a, b, c, p\rangle \mid a, b, c, \text{and } p \text{ are positive binary integers such that } a^b = c \pmod p\}$

Show that $MODEXP \in$P

Because b is a binary integer, let $b = b_1b_2...b_{n-1}b_n$. So the decimal representation of b is $\sum_{k=1}^{n} b_{n-k+1}2^{k-1}$. By the hint, we observe that $a^{(10)_2} = a^2$ and $a^{(1000)_2} = ((a^2)^2)^2$.

Construct the following algorithm to decide $MODEXP$:

A = "On input $\langle a, b, c, p\rangle$ where a, b, c, and p are positive binary integers:

1. Let $T = 1$ (and $n = \lceil log_2 b \rceil$).
2. For $i = 1$ to n,
   a. if $b_i = 1$, $T = (a(T^2) \pmod p)$,
   b. if $b_i = 0$, $T = (T^2 \pmod p)$;
3. Return $T \pmod p$.
4. If $T = c \pmod p$, accept; otherwise reject."

Assume that a, b, c and p are at most m bits (so $n \leq m$). It is known that two m-bit numbers cost $O(m^2)$ unit time to do multiplication and division (and hence modular), so each i costs $O(m^2)$ time. The total cost of the for loop is $O(m^2) \times n = O(m^3)$, which is the dominant cost of all steps. The time complexity of the algorithm A is polynomial in the length of its input.

Problem 7.15

P is closed under the star operation.

Assume that $\Sigma = \{a, b\}$. Then, star operation yields any combination of a and b, where checking whether any string that is generated by this star operation belong to this language can be done in polynomial time which is the length of the string. If we use dynamic programming approach, we'll save the result of whether a string belong to the language. In this way, if we want to find if a string with length $n + 1$ belong to this language, we can check whether the substring of length n (excluding the last character) is in the language, then check if the last character is also in the language. Take this idea, given an input string w, we can divide this into two substrings, and check whether these substrings both belong to the language. This procedure can be done in polynomial time.

Problem 7.18

Show that if $P = NP$, then every language $A \in P$, except $A = \emptyset$ and $A = \Sigma^*$, is NP-complete.

Let A be any language in NP and let B be another language not equal $\emptyset$ or Σ^*. Then there exist strings $x \in B$ and $y \notin B$. To reduce an instance w of A to that of B, we just check in polynomial time if $w \in A$. If yes, we output x and y when $w \notin A$. That is, $f(w) = x$ if $w \in A$, and $f(w) = y$ if $w \notin A$. So $w \in A$ iff $f(w) \in B$.

The languages $\emptyset$ and Σ^* cannot be NP-complete, because to reduce a language A to a language B, we need to map instances in A to instances in B and those outside of A to outside B. However, for $B = \emptyset$, there are no instances in B (and none outside B for $B = \Sigma^*$) which means there cannot be such a reduction from any language $A \neq \emptyset$, Σ^*.

Problem 7.21

a. $SPATH \in P$ because breadth-first search (BFS) can be used to compute the shortest paths from a to any other points. If the shortest path from a to b is equal to or less than k, then $(G, a, b, k) \in SPATH$; otherwise, then $(G, a, b, k) \notin PATH$. Since BFS takes linear time, $SPATH \in P$.

b. $LPATH \in NP$ because there exists a polynomial time verification algorithm which takes (G, a, b, k, s), where s is a list of k distinct vertices of G, starting at a and ending with b. The verification algorithm will check if there exists an edge of G between any two consecutive points in s. This checking takes $O(k)$ time where $k < n$, and n is the number of vertices in G. To show that $LPATH$ is NP-Hard, we reduce the $HAMPATH$ problem to $LPATH$. For any instance (G, s, t) of $HAMPATH$ there exists a Hamiltonian path from s to t in G iff there exists a simple path from s to t of length at least $n - 1$, where n is the number of vertices in G. So the reduction takes (G, s, t) and produces (G, s, t, k), where $k = n - 1$. This reduction takes linear time.

Problem 7.22

DOUBLE-SAT $\in NP$ because there exists a polynomial time verification algorithm which takes (ϕ, s) where $s = \{s_1, s_2\}$, two distinct assignments of Boolean formula ϕ, and check if both s_1 and s_2 are models of ϕ, i.e., s_1 and s_2 are satisfying assignments of ϕ.

DOUBLE- SAT is NP-hard because we can reduce SAT to DOUBLE-SAT: for any instance ϕ of SAT, let z be a Boolean variable not appearing in ϕ, and $\phi' = \phi$ and $(y \vee \neg y)$. Then ϕ is satisfiable iff ϕ' has at least two satisfying assignments. That is, if ϕ has one satisfying assignment, say s, then let $s_1 = s \cup \{y = true\}$ and $s_2 = s \cup \{y = false\}$, then both s_1 and s_2 are satisfying assignments of ϕ'. On the other hand, if ϕ' has two satisfying assignments, these two assignments are also satisfying assignments of ϕ by ignoring y.

Problem 7.24

a. $CNF2 \in P$ because there exists a polynomial time algorithm to solve $CNF2$. For any Boolean formula ϕ we construct $G = (V, E)$, where V is the set of all possible literals using the variables in ϕ and E consists of the following edges: For any binary clause $(a \vee b)$ of ϕ, we create two edges: $\neg a$ to b and $\neg b$ to a. This graph represents the implication relation between the literals. Then ϕ is satisfiable iff there exists no cycles in G such that both x and $\neg x$ are in the cycle for some variable x. That is, if ϕ has a satisfying assignment, then every binary clause $(a \vee b)$ is true, so the associated two implications $(\neg a \to b)$ and $(\neg b \to a)$ must be true. That means x and $\neg x$ cannot be in the same cycle of G, because the implication relation is transitive. If they are in the same cycle, we would have both $(\neg x \to x)$ and $(x \to \neg x)$ being true, impossible in logic.

On the other hand, if G does not have such a cycle, we may assign a true value to every literal in V: If there exists a path from $\neg x$ to x, assign x true. If there exists a path from x to $\neg x$, assign x false. For each edge (a, b), if a is assigned true, then assign b to be true; if b is assigned false, then assign a to be false. We will obtain a satisfying assignment of ϕ this way.

b. The implication of $CNF3$ is in form a then $b \vee c$. If we set a to be True, then b or c must be True, but we cannot determine which one to be True. Therefore, $CNF3 \notin P$. Which means, CNF3 is NP-complete.