Ye Xue EECS 336 Homework 8

Question 1.

a. Exercise 34.2-1

Let a certificate y be a permutation that maps vertices of G_1 to vertices of G_2. Let R be an algorithm to verify the language. Steps of R:

1. Check if y is a permutation.
2. If the answer is no, then output "No" and stop.
3. Check that (u, v) is an edge of G_1 if and only if $(y(u), y(v))$ is an edge of G_2.
4. If the answer is yes, then output "Yes".
5. If the answer is no, then output "No".

Step 1 takes $O(V)$ time. Step 3 takes $O(E)$ time. Therefore, R runs in $O(V + E)$, which is polynomial time. The size of y is at most V, which is polynomial.

The verification algorithm R runs in polynomial-time for L = GRAPH-ISOMORPHISM, such that:

1. x in L if and only if at least one string y has $R(x, y) = 1$.
2. x is not in L if and only if all strings y have $R(x, y) = 0$.

Therefore GRAPH-ISOMORPHISM $\in$ NP.

Question 2.

a. Exercise 34.2-8

The complement problem of **TAUTOLOGY** is: The formula ϕ' is **complement tautology** if it evaluates to 0 for at least one assignment of 1 and 0 to the input variables.

Let a certificate y be a permutation of assignments of the input variables. Let V be an algorithm to verify the language **complement tautology**. Steps of V:

1. Check if y is a permutation.
2. If the answer is no, then output "No" and stop.
3. Check that assignments in y make ϕ' evaluates to 0.
4. If the answer is yes, then output "Yes".
5. If the answer is no, then output "No".

Both step 1 and step 3 takes $O(n)$ time in size of input variables. Therefore, V runs in $O(n)$, which is polynomial time. The size of y is at most the size of input variables.

The verification algorithm V runs in polynomial-time for L = **complement tautology**, such that:

1. x in L if and only if at least one string y has $V(x, y) = 1$.
2. x is not in L if and only if all strings y have $V(x, y) = 0$.

Therefore, **complement tautology** $\in$ NP. So **TAUTOLOGY** $\in$ co-NP.

Question 3.

a. 34-3(a)

1. Insert any vertex of the graph to a queue Q as starting vertex. Choose color c as starting color.

2. If Q is empty, return "Yes".

3. Pop a vertex s from Q, paint it with c and set c to the other color.

4. Check whether there is an edge which contains s whose both end are painted with the same color.

5. If the answer is yes, return "No".

6. Find all the vertices that are connected to s and insert them into Q.

7. go to step 2.

If the algorithm returns "Yes", it's a 2-coloring of graph. If "No", there is no 2-coloring of this graph.

b. 34-3(b)

The decision problem L could be: Can a graph $G(V, E)$ colored by using k different colors such that all neighboring vertices are colored with different colors.

Assume the **graph-coloring problem** can be solved by algorithm A_{opt} and the decision problem above can be solved by algorithm A_d.

Claim 1: If L is solvable in polynomial time, there exists an algorithm A_{opt} that runs in polynomial time.

Proof: Since L is solvable in polynomial time, it can be solved by a polynomial time algorithm A_d, which takes $O(n^{k'})$ time. Then A_{opt} can be write as follow (Algorithm 1).

Algorithm 1

```
1: procedure A_opt(G, u, v)
2:     for k ← 0 to |V| do
3:         if A_d(G, u, v, k) = true then
4:             break
5:         end if
6:     end for
7:     return k
8: end procedure
```

A_{opt} takes $O(kn^{k'})$ for constant k and k', which is polynomial time.

Claim 2: If **graph-coloring problem** can be solved in polynomial time, then L is solvable in polynomial time.

Proof: **graph-coloring problem** can be solved in polynomial time, so there is a polynomial algorithm A_{opt} that solve this problem, taking $O(n^k)$ time for constant k. Then A_d can be write as follow (Algorithm 2).

Thus, A_d also takes $O(n^k)$ time for constant k, that is, L is solvable in polynomial time. Therefore, **graph-coloring problem** can be solved in polynomial time if and only if L is solvable in polynomial time.

Algorithm 2

```
1: procedure A_d(G, u, v, k)
2:     min_k ← A_opt(G, u, v)
3:     if k ≥ min_k then
4:         return true
5:     else
6:         return false
7:     end if
8: end procedure
```

c. 34-3(c)

To determine if a graph is correctly colored, we check if any edge of the graph whose both end vertices are painted with the same color. This verification process can be done in polynomial time. Thus, the 3-COLOR problem can be reformed as a decision problem, which is a polynomial-time problem.

For a graph $G(V, E)$, we add $k - 3$ connected vertices to G. All other vertices are connected to these vertices and this creates a new graph $G'(V', E')$. Then, G is 3-colorable only if G' can be painted using k colors. This conversion runs in $O(E' - E)$, which is polynomial time.

Therefore, if 3-COLOR is NP-complete, the decision problem from part(b) is NP-complete because it is convertible.

Question 4.

a. Hamiltonian Cycle $\leq$ CNF-SAT

Given a graph G, we shall construct a CNF $R(G)$ such that $R(G)$ is satisfiable iff G has a Hamiltonian cycle.

$R(G)$ has n^2 boolean variables x_{ij}, $1 \leq i, j \leq n + 1$.

x_{ij} means the "ith position in the Hamiltonian cycle is occupied by node j". Randomly choose a node as the first position.

$x_{12} = x_{21} = x_{34} = x_{45} = x_{53} = x_{62} = 1$;

$\pi(1) = 2, \pi(2) = 1, \pi(3) = 4, \pi(4) = 5, \pi(5) = 3, \pi(6) = 2$.

1. Each node j must appear in the cycle.
   $x_{1j} \vee x_{2j} \vee ... \vee x_{nj}$ for each j
2. Only one node s appears twice and other nodes j appears once.
   $\neg x_{ij} \vee \neg x_{kj}$ for all $i \leq n, j \leq n, k \leq n$ with $i \neq k$
   let $k = \pi(1)$, $\neg x_{1k} \vee \neg x_{(n+1)k}$, $x_{1k} \vee x_{(n+1)k}$, $\neg x_{1k} \vee x_{(n+1)k}$.
3. Every position i in the cycle must be occupied.
   $x_{i1} \vee x_{i2} \vee ... \vee x_{in}$ for each i
4. No two nodes j and k occupy the same position in the cycle.
   $\neg x_{ij} \vee \neg x_{ik}$ for all i, j, k with $j \neq k$.
5. Nonadjacent nodes i and j cannot be adjacent in the cycle.

$\neg x_{kj} \vee \neg x_{k+1,j}$ for all $(i,j) \notin G$ and $k = 1, 2, ..., n-1$

$R(G)$ contains $O(n^3)$ clauses.

$R(G)$ can be computed efficiently.

Suppose $T \models R(G)$.

From the 1st and 2nd types of clauses, for each node j there is a unique position i such that $T \models x_{ij}$.

From the 3rd and 4th types of clauses, for each position i there is a unique node j such that $T \models x_{ij}$.

So there is a permutation π of the nodes such that $\pi(i) = j$ if and only if $T \models x_{ij}$.

The 5th type of clauses furthermore guarantee that $(\pi(1), \pi(2), ..., \pi(n))$ is a Hamiltonian cycle.

Conversely, suppose G has a Hamiltonian cycle $(\pi(1), \pi(2), ..., \pi(n))$, where π is a permutation.

Clearly, the truth assignment $T(x_{ij}) = true$ if and only if $\pi(i) = j$ satisfies all clauses of $R(G)$.

b. Hamiltonian Cycle $\leq$ 01-LP

Given a graph G, we shall construct a 01-LP $R(G)$ such that $R(G)$ is satisfiable iff G has a Hamiltonian cycle.

$R(G)$ has n^2 boolean variables x_{ij}, $1 \leq i, j \leq n+1$.

x_{ij} means the "ith position in the Hamiltonian cycle is occupied by node j". Randomly choose a node as the first position.

$x_{12} = x_{21} = x_{34} = x_{45} = x_{53} = x_{62} = 1$;

$\pi(1) = 2, \pi(2) = 1, \pi(3) = 4, \pi(4) = 5, \pi(5) = 3, \pi(6) = 2$.

1. Each node j must appear in the cycle.

   $x_{1j} + x_{2j} + ... + x_{nj} \geq 1$ for each j

2. Only one node s appears twice and other nodes j appears once.

   $(1 - x_{ij}) + (1 - x_{kj}) \geq 1$ for all $i \leq n, j \leq n, k \leq n$ with $i \neq k$

   let $k = \pi(1)$, $(1 - x_{1k}) + (1 - x_{(n+1)k}) \geq 1$, $x_{1k} + x_{(n+1)k} \geq 1$, $(1 - x_{1k}) + x_{(n+1)k} \geq 1$.

3. Every position i in the cycle must be occupied.

   $x_{i1} + x_{i2} + ... + x_{in} \geq 1$ for each i

4. No two nodes j and k occupy the same position in the cycle.

   $(1 - x_{ij}) + (1 - x_{ik}) \geq 1$ for all i, j, k with $j \neq k$.

5. Nonadjacent nodes i and j cannot be adjacent in the cycle.

   $(1 - x_{kj}) + (1 - x_{k+1,j}) \geq 1$ for all $(i,j) \notin G$ and $k = 1, 2, ..., n-1$

$R(G)$ contains $O(n^3)$ inequalities.

$R(G)$ can be computed efficiently.

Suppose $T \models R(G)$.

From the 1st and 2nd types of inequalities, for each node j there is a unique position i such that $T \models x_{ij}$.

From the 3rd and 4th types of inequalities, for each position i there is a unique node j such that $T \models x_{ij}$.

So there is a permutation π of the nodes such that $\pi(i) = j$ if and only if $T \models x_{ij}$.

The 5th type of inequalities furthermore guarantee that $(\pi(1), \pi(2), ..., \pi(n))$ is a Hamiltonian cycle.

Conversely, suppose G has a Hamiltonian cycle $(\pi(1), \pi(2), ..., \pi(n))$, where π is a permutation.

Clearly, the truth assignment $T(x_{ij}) = true$ if and only if $\pi(i) = j$ satisfies all binary linear inequalities of $R(G)$.