Homework 6

Problem 1. Prove that if P = NP, then NP = coNP.

Solution. Prove by observing that P is closed under complement.

Problem 2. For every 2-SAT instance φ of n variables, define graph G_φ of $2n$ vertices as follows. For each variable x_i in φ, G_φ has two vertices labeled by x_i and $\neg x_i$ respectively. There is a directed edge $\ell_i \to \ell_j$ if $(\neg \ell_i) \vee \ell_j$ or $\ell_j \vee (\neg \ell_i)$ is a clause of φ. For notational convenience, for literal $\ell_i = \neg x_{k_i}$, $\neg \ell_i$ is defined to be x_{k_i}. Prove that φ is unsatisfiable if and only if there exist paths from x_j to $\neg x_j$ and from $\neg x_j$ to x_j in G_φ for some j. Use the above fact to show that 2-SAT $\in$ P.

Solution. Note that for a path from $x_i \twoheadrightarrow x_j$ can be satisfied if and only if we can have a non decreasing assignment to all the variables in the path. The if case is easy to verify, since there must be a decreasing assignment in the two paths $x_i \twoheadrightarrow \neg x_i$ and $\neg x_i \twoheadrightarrow x_i$.

We have the following observation for G_φ: if there is a path from $l_i \twoheadrightarrow l_j$, there is a path from $\neg l_j \twoheadrightarrow \neg l_i$. Now if we assume for any i there is no simultaneous path from x_i to $\neg x_i$ and $\neg x_i$ to x_i, we show the following algorithm can find a satisfying solution for ϕ.

At the first place, every variable is unassigned.

1. For each i, if x_i is not assigned, we check if there is a path from x_i to $\neg x_i$, or vice versa. For the first case, we set $x_i = 0$ and set every reachable term l_i from $\neg x_i$ to 1, and set $\neg l_i = 0$. We perform similar operation for the second case.

2. After testing every n, if there is still unassigned variable x_i, we set $x_i = 1$ and every reachable term l_i from x_i to 1. Repeat until every variable is assigned.

We now prove that there will not be any conflict in the assignment process. Note that when we examine each unassigned variable, all of its predecessor should be unassigned/0, otherwise it would be assigned to 1 in previous steps of the algorithm. By symmetry of G_φ, its successors should be unassigned/1. Now if in step 1 we obtain a conflict, suppose we have a path $\neg x_i \twoheadrightarrow x_i$, and we assign 1 to some successor l_i of x_i that is previously assigned to 0.

This implies there exists some l_j such that we have $l_i \twoheadrightarrow l_j$ and $l_j \twoheadrightarrow \neg l_j$. By symmetry, there is also a path $\neg l_j \twoheadrightarrow \neg x_i$, thus we obtained a path $x_i \twoheadrightarrow l_i \twoheadrightarrow l_j \twoheadrightarrow \neg l_j \twoheadrightarrow \neg x_i$, contradiction to our assumption. We can argue the other cases by symmetry. At step 2, we can similarly construct a path $x_i \twoheadrightarrow \neg x_i$ if there is a conflict, which contradict to our step 2 condition that there should be no path $x_i \twoheadrightarrow \neg x_i$.

We can design our algorithm of 2-SAT as follows: for each x_i, we check if x_i and $\neg x_i$ are reachable from each other by running BFS/DFS.

Problem 3. The Lehmer's theorem states that a natural number n is a prime number if and only if the following two conditions hold:

1. There is number a such that $a^{n-1} \equiv 1 \pmod{n}$.
2. For every prime factor q of $n-1$, $a^{(n-1)/q} \not\equiv 1 \pmod{n}$.

Use this theorem to show that PRIME $\in$ NP $\cap$ coNP. (Hint: To prove PRIME $\in$ NP, you may need to use recursively defined witness.)

Solution. Proving PRIME $\in$ coNP is easy, our witness is some $m \neq 1$ that $m \mid n$, and checking $m \mid n$ is in $O(\log n)$ time.

To prove PRIME $\in$ NP, we would construct the witness $W(n)$ recursively:

$$W(n) = \langle a, q_1, \ldots, q_k, W(q_1), \ldots, W(q_k) \rangle,$$

where q_i are the prime factors of $n-1$, and $W(q_i)$ is the corresponding witness for proving q_i is a prime. We can also assume wlog $a < n$.

Easy to see that $\sum_{i=1}^{k} \log(q_i) \leq \log n$. Thus we can see that $|W(n)| = \sum_{i=1}^{k} |W(q_i)| + O(\log n)$, it can be checked that $W(n) = O(\log^2 n)$.

The verification algorithm run as follows:

- Check $a^{n-1} \equiv 1 \pmod{n}$. Each multiplication of cost time $O(\log^2 n)$, computing a^{n-1} costs $O(\log^3 n)$ time using exponentiation by squaring.
- Check q_i are factors of $n-1$, and there exists t_i such that $\prod q_i^{t_i} = n-1$. Each division costs time $O(\log^2 n)$, and k, t_i is bounded by $\log n$. The total running time is $O(\log^4 n)$.
- Check $a^{(n-1)/q} \not\equiv 1 \pmod{n}$. Costs $O(\log^4 n)$ time.
- Check q_i is a prime using $W(q_i)$. Costs $\sum_i T(\log q_i)$

Thus $T(\log n) = \sum_{i=1}^{k} T(\log q_i) + O(\log^4 n)$. It can be verified that $T(\log n) = O(\log^5 n)$.