Problem Set 9

Ekkapot Charoenwanit
Efficient Algorithms

Problem 1. NP-Completeness

1) The Hamiltonian Cycle (Ham-Cycle) Problem

Given a directed graph $G = (V, E)$, a Hamiltonian cycle of G is a simple cycle that contains each vertex in V. The Hamiltonian Cycle (Ham-Cycle) problem is to figure out if there is at least one Hamiltonian cycle in the given graph G. We can show that Ham-Cycle is NP-complete by reducing from the Vertex Cover problem.

Show that if Ham-Cycle $\in P$, then the problem of showing a sequence of vertices that form a Hamiltonian cycle in a given directed graph $G = (V, E)$ is also polynomial-time solvable.

Solution: Suppose that the **black-box** polynomial-time algorithm for solving the decision problem is called HAM-CYCLE, which takes in a graph G and returns either Yes if G contains a Hamiltonian cycle or returns No, otherwise. This black-box algorithm is called in line 7 of Algorithm 1.
The algorithm picks each vertex $v \in V$ and tries deleting all edges incident on v (the set of which is denoted by E_v) but two edges denoted as e_1 and e_2. As the degree of any vertex is at most $n-1$, there are at most $\binom{|V|-1}{2} = \frac{(|V|-1)(|V|-2)}{2} = \mathcal{O}(|V|^2)$ ways to delete all but two edges. Thus, the for loop in line 5 runs for at most $O(|V|^2)$ iterations, which is polynomial in $|V|$, for each vertex $v \in V$. Therefore, the algorithm runs in polynomial time in total.
Note: If no pair of edges (e_1, e_2) exists for a certain vertex v, the graph G contains no Hamiltonian cycle and can immediately return the **empty set** as shown in lines 10 and 11.

Algorithm 1 implements an algorithm for solving the Hamiltonian Cycle problem

```
1: procedure Get-Ham-Cycle($G = (V, E)$)
2:     $HC = \emptyset$
3:     Let $E_v = \{(x, y) | x = v \land y = v\}$
4:     for each vertex $v \in V$ do
5:         for each $(e_1, e_2)$ where $e_1, e_2 \in E_v$ do
6:             $G^{'} = (V, (E - E_v) \cup \{e_1, e_2\})$
7:             if HAM-CYCLE($G^{'}$) then
8:                 $HC = HC \cup \{e_1, e_2\}$
9:                 break
10:        if $E_v = \emptyset$ then
11:            return $\emptyset$
12:    return $HC$
```

2) Given a boolean formula ϕ, show that the problem of determining whether the given boolean formula ϕ is a tautology is in the $co-NP$ class.

Solution: To show that this decision problem is $co-NP$, we must show that we can implement a **verifier** for the **Reject** case in polynomial time.
Given a **rejected** truth assignment as a **certificate**, it is obvious that we can verify that the formula ϕ evaluates to False easily.

3) Prove the following claims:

(3.1) $P \subseteq NP$

Proof: The proof is simple using the following argument. To show that a decision problem q is in NP, we must show that there exists a polynomial-time verifier V that takes in a certificate c and a **Yes** instance i. Since $q \in P$, we can solve q from scratch in polynomial time by running a known polynomial-time algorithm on the instance i. We then use this as the output of the verifier V, completely ignoring the certificate c.

(3.2) $P \subseteq co - NP$

Proof: The proof is similar to the proof for (3.1). To show that a decision problem q is in $co - NP$, we must show that there exists a polynomial-time verifier V that takes in a certificate c and a **No** instance i.
Since $q \in P$, we can solve q from scratch in polynomial time by running a known polynomial-time algorithm on the instance i. We then use this as the output of the verifier V, completely ignoring the certificate c.

Problem 2. Approximation Algorithms

1) Implement the approximation algorithm for the Vertex Cover problem in the lecture using an adjacency list and show that the running time of this implementation is polynomial in the input size $|V|$ and $|E|$.

2) Design a greedy algorithm for finding an optimal vertex cover given an undirected tree. Give an exchange argument you use to argue that your greedy strategy yields an optimal solution? You do not have to give a formal proof.

3) Show that the following weaker form of Theorem III on Slide 36 in Lecture 13 holds:

$|\mathbb{C}| \leq \mathbb{C}^* \max\{S|S \in \mathbb{F}\}$

Problem 3. Bin Packing

Suppose we are given a set of n items, where the size s_i of the i^{th} item satisfies $0 < s_i < 1$. We want to pack these n items into the minimum number of identical bins, each of which has a capacity of 1 unit. It is assumed that each bin can only contain a subset of items whose total size does not exceed its capacity.

Since the Bin Packing problem is NP-Hard, its decision version can be shown to be NP-complete by reducing from the Subset Sum problem.

Therefore, we have come up with an approximation algorithm called **First-Fit** that works by taking each item in turn and putting it into the first bin that has enough capacity to accommodate it.

(1) Design a polynomial-time algorithm for **First-Fit**.

(2) Analyze the running time.

(3) Show that **First-Fit** is a 2-approximation algorithm.

You might want to follow the following steps in order to prove the claim:

(3.1) Show that the minimum number of bins is at least $\lceil S \rceil$.

(3.2) Show that **First-Fit** leaves at most one bin less than half-full.

(3.3) Show that the number of bins **First-Fit** uses never exceeds $\lceil 2S \rceil$.