Problem Set 10

Due date: Electronic submission of this homework is due on **4/21/2023** on canvas. In order to receive any credit, you need to typeset your homework in LaTeX and submit the resulting pdf file. Any submission that cannot be checked for plagiarism will not be graded.

Name: Huy Quang Lai

Resources. Cormen, Thomas H., Leiserson, Charles E., Rivest, Ronald L., Stein, Clifford. *Introduction to Algorithms.* The MIT Press.

On my honor, as an Aggie, I have neither given nor received any unauthorized aid on any portion of the academic work included in this assignment. Furthermore, I have disclosed all resources (people, books, web sites, etc.) that have been used to prepare this homework.

Signature: ____________________________

Read the chapter on Approximation Algorithms in our textbook and study the slides on the Polynomial Hierarchy.

Problem 1. (20 points) Prove by induction that if **P**=**NP**, then **PH**=**P**. [Hint: Study the slides on the Polynomial Hierarchy]

Solution. Assume that $\mathbf{P} = \mathbf{NP}$
By induction we will prove that $\Sigma_i, \Pi_i = \mathbf{P}$ by induction for all $i > 0$.
Base Case:
$i = 1$
$\Pi_1 = \mathbf{coNP}$ in the class of languages L for which $L^c \in \mathbf{NP}$.
Since $\mathbf{P} = \mathbf{NP}$, $L^c \in \mathbf{P}$
Therefore, $\Pi_1 = \mathbf{P} = \mathbf{NP}$.
By assumption, $\Sigma_1 = \mathbf{P} = \mathbf{NP}$
Inductive Case:
Assume that $\Sigma_i, \Pi_i = \mathbf{P}$
A language $L \in \Sigma_{i+1}$ if $\exists L' \in \Pi_i \wedge c > 0$ such that

$$x \in L \Leftrightarrow \exists y, |y| < |x|^c, (x, y) \in L'$$

By the inductive hypothesis, $L' \in \mathbf{P}$. Therefore, there exists a polynomial-time algorithm to compute it. However, $L \in \mathbf{NP}$
Since $\mathbf{P} = \mathbf{NP}, L \in \mathbf{P}$. Therefore, $\Sigma_{i+1} = \mathbf{P}$
A similar proof exists to show that $\Pi_{i+1} = \mathbf{P}$
A language $L \in \Pi i + 1$ if $\exists L' \in \Sigma_i \wedge c > 0$ such that

$$x \in L \Leftrightarrow \exists y, |y| < |x|^c, (x, y) \in L'$$

By the inductive hypothesis, $L' \in \mathbf{P}$. Therefore, there exists a polynomial-time algorithm to compute it. However, $L \in \mathbf{NP}$
Since $\mathbf{P} = \mathbf{NP}, L \in \mathbf{P}$. Therefore, $\Sigma_{i+1} = \mathbf{P}$

Problem 2. (30 points) Let C_n denote the cycle graph on n vertices, where n is a positive integer satisfying $n \geq 3$.
(a) What is the size of a maximum independent set in C_n?
(b) For which n can the approximation algorithm APPROX-VERTEX-COVER (see our textbook) give an optimal result? [Hint: You can use your insights from part (a).]

Solution. Part (a)
An independent set in a graph is a set of vertices in which no two vertices are adjacent. In a cycle graph C_n, any two adjacent vertices form an independent set of size 2. Moreover, any vertex in C_n is adjacent to two other vertices. Therefore, any independent set in C_n can contain at most one vertex from each pair of adjacent vertices.
If n is odd, then C_n has an odd number of vertices, say $n = 2k + 1$, and the maximum independent set is obtained by selecting every other vertex along the cycle. This gives a set of size $k + 1$, because there are $k + 1$ pairs of adjacent vertices in C_n.
If n is even, then C_n has an even number of vertices, say $n = 2k$, and the maximum independent set is obtained by selecting every other vertex along the cycle, starting with either the first or second vertex. This gives a set of size k, because there are k pairs of adjacent vertices in C_n.
Therefore, the size of a maximum independent set in C_n is $\left\lceil \frac{n}{2} \right\rceil$, where $\lceil x \rceil$ denotes the smallest integer greater than or equal to x.
Part (b)
First, note that any vertex cover of C_n must contain at least one vertex from each pair of adjacent vertices, because otherwise the two adjacent vertices are not covered. Therefore, C^* contains at least $\left\lceil \frac{n}{2} \right\rceil$ vertices, which is the number of pairs of adjacent vertices in C_n.
Now consider the vertices selected by the algorithm. Let $v_1, v_2, \ldots, v_k$ be the vertices selected in order, and let E_i be the set of edges incident to v_i that are not yet covered after step i. Note that $|E_1| = \deg(v_1)$, and $|E_i| \leq \deg(v_i)$ for $i > 1$, because each edge incident to v_i is adjacent to a vertex that has already been covered. Therefore, we have

$$|C| = k \leq \sum_{i=1}^{k} |E_i| \leq \sum_{i=1}^{k} \deg(v_1)$$

where the last inequality follows from $|E_i| \leq \deg(v_i)$.
On the other hand, since C^* is a vertex cover, we have

$$\sum_{v \in C^*} \deg(v) \geq n$$

because every edge is incident to at least one vertex in C^*. Therefore, if $\sum_{v \in C^*} \deg(v) = n$, then C^* is a minimum vertex cover and the algorithm will output C^* as well.

If n is odd, then C_n has $\lfloor n/2 \rfloor$ pairs of adjacent vertices, and the maximum degree of any vertex is 2. Therefore, the sum of degrees in C_n is n, and C^* is a minimum vertex cover. In this case, the algorithm will output a minimum vertex cover and the result will be optimal.
If n is even, then C_n has $\frac{n}{2}$ pairs of adjacent vertices, and the maximum degree of any vertex is 2. Therefore, the sum of degrees in C_n is n, and C^* is a minimum vertex cover if and only if C^* contains exactly one vertex from each pair of adjacent vertices. If C^* contains more than one vertex from any pair of adjacent vertices, then we can obtain a smaller vertex cover by removing one of the vertices. In this case, the algorithm will output a vertex cover of size $\left\lceil \frac{n}{2} \right\rceil$, which is at most one larger than the minimum vertex cover. Therefore, the algorithm will give an optimal result if and only if C^* contains exactly one vertex from each pair of adjacent vertices.

Problem 3. (30 points) (a) Give an efficient greedy algorithm that finds an optimal vertex cover for a tree in linear time. (b) Explain why your algorithm works and justify your claim about the run-time.

Solution. Part (a)

1. Start at the root node of the tree.
2. Traverse the tree using depth-first search (DFS) and maintain two sets of nodes: S (the vertex cover) and T (the set of nodes not yet covered).
3. When visiting a node v, add it to S and remove v and its neighbors from T.
4. Continue DFS until all nodes have been visited.

Part (b)
This algorithm works because a tree is a special case of a graph where each edge connects exactly two nodes. Hence, for any edge in the tree, one of its endpoints must be included in the vertex cover. Therefore, we can use a greedy approach and include each node as we traverse the tree.
Moreover, the algorithm's runtime is linear, $O(n)$, where n is the number of nodes in the tree. This is because we visit each node once, and each node takes constant time to process (adding to S and removing from T). Additionally, the set operations can be performed in constant time using hash tables or other efficient data structures. Therefore, the overall runtime of the algorithm is $O(n)$.

Problem 4. (20 points) From the lecture (see also the chapter on NP-completeness in our textbook for details), we know that the vertex-cover problem and the NP-complete clique problem are complementary in the sense that an optimal vertex cover is the complement of a maximum-size clique in the complement graph. Does this relationship imply that there is a polynomial-time approximation algorithm with a constant approximation ratio for the clique problem? Justify your answer.

Solution. No, the relationship between the vertex cover problem and the clique problem does not imply the existence of a polynomial-time approximation algorithm with a constant approximation ratio for the clique problem.
Although an optimal vertex cover is the complement of a maximum-size clique in the complement graph, this relationship only provides a way to obtain a lower bound for the clique problem from an upper bound for the vertex cover problem. It does not give any direct insight into the approximability of the clique problem.
The clique problem is known to be NP-complete, and there is no known polynomial-time approximation algorithm with a constant approximation ratio for it, unless $\mathbf{P} = \mathbf{NP}$. In fact, it is known that there is no polynomial-time approximation algorithm for the clique problem with an approximation ratio better than $O\left(n^{\frac{1}{2}-\epsilon}\right)$, for any $\epsilon > 0$, unless $\mathbf{P} = \mathbf{NP}$.
Therefore, the relationship between the vertex cover problem and the clique problem does not imply the existence of a polynomial-time approximation algorithm with a constant approximation ratio for the clique problem.