ShanghaiTech University

CS240 Algorithm Design and Analysis

Fall 2023

Problem Set 3

Due: 23:59, Nov. 21, 2023

1. Submit your solutions to Gradescope (www.gradescope.com).
2. In "Account Settings" of Gradescope, set your FULL NAME to your Chinese name and enter your STUDENT ID correctly.
3. If you want to submit a handwritten version, scan it clearly. Camscanner is recommended.
4. When submitting your homework, match each of your solution to the corresponding problem number.

Notice: When proving that a problem A is NP-complete, you need to strictly follow the below steps and explicitly state each part.

- Prove that A is in NP.
- Choose an NP-complete problem B. For any instance b of B, construct an instance a of problem A in polynomial time.
- Prove that b has a solution if and only if a has a solution.

Problem 1:

NAE-4-SAT: A clause in an instance of 4-SAT is a disjunction with four literals i.e. $(x_1 \vee x_2 \vee x_3 \vee x_4)$. A satisfied assignment for a collection of clauses is called *not all equal* (NAE) if the literals in each clause are not all equal to each other. In other words, at least one false, and at least one true. Prove NAE-4-SAT is NP-complete, using a reduction from 3-SAT.

Solution:

Lemma. NAE-4-SAT is in NP.

Proof. Give an assignment of variables in an instance of NAE-4-SAT, we can check whether the assignment is satisfied by replacing the variables with boolean values in the assignment, it is polynomial time.

Reduction. We choose 3-SAT as the target NP-complete problem. For any instance $b = \bigwedge_{i=1}^{n} C_i$ of 3-SAT. For each $C_i = (x_i \vee y_i \vee z_i)$, we transform C_i into C_i' by adding a fresh variable s, that is

$$C_i' = (x_i \vee y_i \vee z_i \vee s) \wedge (\bar{x_i} \vee \bar{y_i} \vee \bar{z_i} \vee \bar{s})$$

Then we can get a instance $a = \bigwedge_{i}^{n} C_i'$ of NAE-4-SAT in polynomial time.

Lemma. b is satisfied if and only if a is satisfied.

Proof. ($\Rightarrow$) Suppose b has a satisfied assignment S. Then we can obtain a satisfied assignment S with $s =$ true for a.

($\Leftarrow$) Suppose a has a satisfied assignment W. Consider each $C_i' = (x_i \vee y_i \vee z_i \vee s) \wedge (\bar{x_i} \vee \bar{y_i} \vee \bar{z_i} \vee \bar{s})$ in a. If $s =$ false in W, then W is clearly a satisfied assignment for S. If $s =$ true in W, then $\bar{x_i}$, $\bar{y_i}$ and $\bar{z_i}$ can not be all false. Thus we can invert all boolean values in W to get a satisfied assignment for b.

Problem 2:

NAE-3-SAT: Prove NAE-3-SAT is NP-complete.

Solution:

Lemma. NAE-3-SAT is in NP.

Proof. Given an assignment of variables in an instance of NAE-3-SAT, we can check whether the assignment is satisfied by replacing the variables with boolean values in the assignment, it is polynomial time.

Reduction. We choose NAE-4-SAT as the target NP-complete problem. For any instance $b = \bigwedge_{i=1}^{n} C_i$ of 4-SAT. For each $C_i = (x_i \vee y_i \vee z_i \vee w_i)$, we transform C_i into C_i' by adding a fresh variable s, that is

$$C_i' = (x_i \vee y_i \vee s) \wedge (z_i \vee w_i \vee \bar{s})$$

Then we can get a instance $a = \bigwedge_{i}^{n} C_i'$ of NAE-3-SAT in polynomial time.

Lemma. b is satisfied if and only if a is satisfied.

Proof. ($\Rightarrow$) Suppose b has a satisfied assignment S. Consider $C_i = (x_i \vee y_i \vee z_i \vee w_i)$ in b. We can know that there is at least one variable that is true in C_i. If the true variable is x_i or y_i, then we get a satisfied assignment S with $s =$ false for a. Otherwise we can get a satisfied assignment S with $s =$ true for a.

($\Leftarrow$) Suppose a has a satisfied assignment W. Consider each $C_i' = (x_i \vee y_i \vee s) \wedge (z_i \vee w_i \vee \bar{s})$ in a. If $s =$ false in W, we can know that at least one of x_i and y_i is true. Thus W is a satisfied assignment for b. If $s =$ true in W, we can know that at least one of x_i and y_i is true. Thus W is a satisfied assignment for b.

Problem 3:

4-SAT: Prove 4-SAT is NP-complete.

Solution:

The 4-Satisfiability (4-SAT) problem is a specific case of the Boolean satisfiability problem (SAT), where each clause in the formula contains exactly four literals. To prove that 4-SAT is NP-complete, we must show that it is both in NP and NP-hard.

A problem is in NP if a solution to the problem can be verified in polynomial time. For the 4-SAT problem, given a Boolean formula and an assignment of values to the variables, it is straightforward to verify whether this assignment satisfies the formula. Checking each clause of the formula can be done in constant time since each clause has exactly four literals. Therefore, the entire verification process is polynomial with respect to the size of the formula, and hence 4-SAT is in NP.

To show that 4-SAT is NP-hard, we use a reduction from the 3-SAT problem, which is known to be NP-complete. The reduction transforms any instance of the 3-SAT problem into an instance of the 4-SAT problem in polynomial time.

Given a 3-SAT formula, we can construct a 4-SAT formula by adding a dummy variable to each clause. For each clause in the 3-SAT formula, such as $(x \vee y \vee z)$, we create a clause in the 4-SAT formula, like $(x \vee y \vee z \vee w)$, where w is a new variable that does not appear elsewhere in the formula. This transformation ensures that if the 3-SAT formula is satisfiable, then the corresponding 4-SAT formula is also satisfiable, and vice versa.

Since every instance of the 3-SAT problem can be transformed into an instance of the 4-SAT problem in polynomial time, and a solution to the 3-SAT instance implies a solution to the corresponding 4-SAT instance, the 4-SAT problem is at least as hard as the 3-SAT problem. Therefore, 4-SAT is NP-hard.

Since 4-SAT is both in NP and NP-hard, it is NP-complete. This proof demonstrates the complexity of 4-SAT and its place in the class of NP-complete problems.

Problem 4:

Feedback Vertex Set problem Let $G = (V, E)$ be a directed graph. A set $F \subseteq V$ is a *feedback vertex set* if every cycle of G contains at least one vertex from F. The *Feedback Vertex Set problem* asks whether G has a feedback vertex set with at most K vertices. Show that Feedback Vertex Set is NP-complete. (Hint: use Vertex Cover problem for reduction.)

Vertex Cover Problem

Given an undirected graph $G' = (V', E')$ and a number K', the Vertex Cover problem asks if there is a vertex cover of size at most K', i.e., a subset of vertices $V'' \subseteq V'$ such that every edge in E' is incident to at least one vertex in V''.

Solution:

1. The problem is in NP.
2. The problem is NP-hard.

A problem is in NP if given a solution, we can verify the correctness of the solution in polynomial time. For the Feedback Vertex Set problem, given a set $F \subseteq V$ of vertices, we can verify whether F is a feedback vertex set of size at most K as follows:

- Check if $|F| \leq K$.
- Remove all vertices in F and their incident edges from G.
- Check if the resulting graph is acyclic. This can be done using depth-first search or other graph traversal algorithms.

If the graph is acyclic after the removal of vertices in F, then F is a valid feedback vertex set. This verification process can be done in polynomial time, thus the Feedback Vertex Set problem is in NP.

To prove NP-hardness, we typically show that an already known NP-complete problem can be polynomial-time reduced to the problem in question. A common approach is to use a reduction from the Vertex Cover problem, which is known to be NP-complete.

- Construct a directed graph $G = (V, E)$ from the undirected graph G' by replacing each undirected edge $\{u, v\}$ in E' with two directed edges (u, v) and (v, u) in E.

- Set $K = K'$.
- If there is a vertex cover of size K' in G', then there is a feedback vertex set of size K in G, and vice versa.

Since every instance of the Vertex Cover problem can be transformed into an instance of the Feedback Vertex Set problem in polynomial time, and a solution to one implies a solution to the other, the Feedback Vertex Set problem is at least as hard as the Vertex Cover problem. Therefore, it is NP-hard.

Problem 5:

STINGY SAT is the following problem: given a set of clauses (each a disjunction of literals) and an integer k, find a satisfying assignment in which at most k variables a true, if such an assignment exists. Prove that STINGY SAT is NP-complete.

Solution:

NP: STINGY SAT can be verified a true assignment in polynomial time, so it's NP.

Reduction: If there is a SAT with k variables, it can be transformed to STINGY SAT, which only has k variables and k of STINGY SAT is k. Given a SAT formula f with k variables we simply chose (f, k) as an instance of Stingy SAT.

$\Rightarrow$ Suppose that f is a yes-instance of SAT. No more than k variables can be true, because there are a total of k variables. So any satisfying assignment of instance f for SAT will be a satisfying assignment of instance (f, k) for Stingy SAT. So (f, k) is a yes-instance of Stingy SAT.

$\Leftarrow$ Suppose that (f, k) is a yes-instance of Stingy SAT. Any satisfying assignment of that instance will be also a satisfying assignment of instance f for SAT. So f is a yes-instance of SAT.

Because SAT is NP-Complete, the STINGY SAT is NP-Complete.

Problem 6:

Given a tree T, a set of terminal vertices, and an integer k, is there a set of at most k edges which, when removed from T, separates every pair of terminal vertices? Prove that this problem is NP-complete.

Solution:

NP: Given a solution (a set of edges), we can check in polynomial time whether it satisfies the condition of the problem. We need to verify that upon removing these edges, all pairs of terminal nodes are segregated. This can be done by performing a depth-first search for each pair of terminal nodes and checking if they are isolated. Hence, this problem is in NP.

Reduction: We can reduce from a known NP-Complete problem, the Vertex Cover problem. Suppose we have an instance of the Vertex Cover problem, with a graph $G = (V, E)$ and an integer k. We can transform it into an instance of the Multicut in Trees problem by converting G into a tree T, where each edge $e \in E$ becomes a path containing two new nodes and a new edge. The set of vertices is all original nodes, plus all new nodes. Terminal nodes are all newly added nodes. k remains the same.

If the Vertex Cover instance has a solution: In this case, there exists a set of at most k vertices such that all edges in G are incident to at least one vertex in the set. Each vertex in this set corresponds to an edge in T, removing these edges will disconnect all pairs of terminal nodes. So, our Multicut in Trees instance also has a solution.

If the Multicut in Trees instance has a solution: In this case, there exists a set of at most k edges, removing these edges will disconnect all pairs of terminal nodes. These edges correspond to a set of vertices in G, the size of the set does not exceed k, and each edge is incident to at least one vertex in the set. So, our Vertex Cover instance also has a solution.

Therefore, we have demonstrated a polynomial-time reduction from the Vertex Cover problem to the Multicut in Trees problem, thus the Multicut in Trees problem is NP-Complete.