AADS - Assignment 4

Aditya Fadhillah (hjg708), Mads Nørregaard (gnz359), Nikolaj Schaltz (bxz911)

January 8, 2025

Contents

1 Exercises from CLRS

1.1 34.2-10

Prove that if NP $\neq$ co-NP, then P $\neq$ NP.

Answer:
To prove that if NP $\neq$ co-NP, then P $\neq$ NP, we use contradiction. We assume that NP $\neq$ co-NP, but P = NP.

Since P is closed under complement, if a language L is in P, then its complement $\overline{L}$ is also in P. Given P = NP, any language in co-NP must also be in P (because its complement is in NP, which equals P). Therefore,

$$\text{co-NP} \subseteq \text{NP} \quad (\text{since P} = \text{NP}).$$

But by definition of co-NP, we also have

$$\text{NP} \subseteq \text{co-NP}.$$

We then combine these inclusions:

$$\text{NP} = \text{co-NP}.$$

This contradicts our original assumption that NP $\neq$ co-NP. Therefore, our assumption P = NP must be false when NP $\neq$ co-NP. Meaning that, if NP $\neq$ co-NP, it follows that P $\neq$ NP.

1.2 34.3-6

A language L is *complete* for a language class C with respect to polynomial-time reductions if $L \in C$ and $L' \leq_p L$ for all $L' \in C$. Show that $\emptyset$ and $\{0,1\}^*$ are the only languages in P that are not complete for P with respect to polynomial-time reductions.

Answer:
We have two languages; $\emptyset$, which is an empty set and always rejects any string, and $\{0,1\}*$, which contains all binary strings and always accepts any string, as long as they are binary. From this, we know that both $\emptyset$ and $\{0,1\}*$ are already in P, since they can be decided by Turing machines that either always rejects or accepts any given strings.

For the empty language $L = \emptyset$, if it has to be P-complete, then $L' \leq_L \emptyset$ for every $L' \in NP$. However, because the language is empty and will always reject strings, no reduction function f exists which can further reduce the language, since no strings $x \in \emptyset$ can be mapped to.

Thus, the empty set $\emptyset$ is not complete for P.

For the language containing all binary strings, $L = \{0,1\}*$, again it has to satisfy $L' \leq_L \{0,1\}*$ for every $L' \in NP$ if it has to be P-complete. However, the language will always accept any strings $x \in \{0,1\}*$, so it does not differentiate between languages in P.

Thus, the set containing all binary strings $\{0,1\}*$ is not complete for P either.

1.3 34.4-3

Professor Jagger proposes to show that SAT $\leq_p$ 3-CNF-SAT by using only the truth-table technique in the proof of Theorem 34.10, and not the other steps. That is, the professor proposes to take the boolean formula ϕ, form a truth table for its variables, derive from the truth table a formula in 3-DNF that is equivalent to $\neg\phi$, and then negate and apply DeMorgan's laws to produce a 3-CNF formula equivalent to ϕ. Show that this strategy does not yield a polynomial-time reduction.

Answer:
We consider a boolean formula ϕ with n variables. The truth table for ϕ has at most 2^n rows since it corresponds with there being 2^n possible assignments to n variables. The construction of the truth table takes

polynomial time. From the truth table for ϕ, we can find all rows that evaluate to false and derive a formula ϕ' equivalent to $\neg\phi$ in 3-DNF form.

The new formula ϕ' has that for each row where ϕ evaluates to false, it corresponds to a term in DNF. Since they are not in 3-DNF, we need to convert the terms to 3-DNF by taking the literals and grouping them together in clauses of 3 to convert to 3-DNF; we can do this using Tseitin transforms since they take polynomial time. Finally, we negate ϕ' to obtain a formula which is equivalent to ϕ.

Since we need to express the new formula in 3-CNF, we negate $\neg\phi$ to obtain a similar formula to ϕ and apply DeMorgan's laws to distribute conjunctions over disjunctions when applicable to obtain a 3-CNF formula. However, this will cause an exponential increase of the size of ϕ, since the no. of clauses will grow with the no. of variables n. The size can be written as $O(2^n)$ since there is an exponential dependency on the number of clauses and number of variables.

To perform a polynomial-time reduction, we require that the size of the output (our new 3-CNF formula) is polynomial in the size of the input (our original formula ϕ); however since we have found that the new 3-CNF formula grows exponentially rather than polynomial, the method cannot constitute a polynomial-time reduction.

Thus, we have shown that the proposed strategy cannot yield a polynomial-time reduction due to the exponential size growth of the 3-CNF formula.

1.4 34.4-6

Suppose that someone gives you a polynomial-time algorithm to decide formula satisfiability. Describe how to use this algorithm to find satisfying assignments in polynomial time.

Answer:

This can be done by using a decision algorithm, i.e an algorithm that answers a yes or no question, iteratively to determine the value of all variables. This is to ensure that it remains satisfiable when values are assigned.

We start by using the decision algorithm to ensure that the formula is satisfiable, as if it is not the process will stop as a satisfying assignment is required. We then go through each variable and set them temporarily to true and check that the decision algorithm is still satisfiable. If it is unsatisfiable then we know that the variable must be false. We do this for all variables such that at the end we have a complete assignment where all variables have specific values that satisfy the formula. This is efficient as we only have to look at each variable once to find out if they're true or false, meaning that the runtime is n with n being the amount of variables, making it polynomial time.

1.5 34.5-1

The *subgraph-isomorphism problem* takes two undirected graphs G_1 and G_2, and it asks whether G_1 is isomorphic to a subgraph of G_2. Show that the subgraph-isomorphism problem is NP-complete.

Answer:
If the subgraph-isomorphism problem is NP-complete, then it must both be in NP and NP-Hard. We have to prove that it belongs in both problems.

Proof for NP:
For the subgraph-isomorphism problem to be in NP, it has to have a solution be verifiable in polynomial time (a verification if G_1 is isomorphic to a subgraph of G_2).

We have the certificate: we let G' be a subgraph of G_2, and we know the vertex mappings between G' and G_1, then to verify the solution we have to check if the solution is a bijection, which takes $O(|V_1|)$ comparisons and for every edge (u, v) in G', there is a corresponding edge in G_1, which requires on average $O(|E_1|)$ checks.

This translates to polynomial time and thus, a solution can be verified in polynomial time, which proves that the problem is in NP.

Proof for NP-Hard:
If the subgraph-isomorphism problem S is also NP-Hard, then every NP problem has to be reducible to S in polynomial time. We can do this by reducing another similar NP problem to S, so we know that S is an NP-Hard problem. We choose the clique problem C to reduce to the subgraph-isomorphism problem, as the clique problem is established to be NP-complete (from Theorem 34.11 in CLRS), and if the reduction can be done in polynomial time, then every NP problem can be reduced to S in polynomial time as well.

The clique problem takes a graph G and a size value k, and verifies if G contains a clique of size k. If we let G_1 be a graph with k vertices and $G_2 = G$, where G_1, G_2 correspond to inputs in the subgraph-isomorphism problem. We see that $k \leq n$, where n is the amount of vertices in G, as if $k > n$ then a clique of size k cannot be a subgraph in G.

It takes $O(k^2) = O(n^2)$ (polynomial) time to construct G_1, as $k \leq n$ as the no. of edges in a complete graph of k size, and G has a clique of k size if and only if G_1 is a subgraph of G_2, as G_1 is already a subgraph of G_2 and every graph is isomorphic; then the solution of the subgraph-isomorphism problem can be verified. Thus, G_1 is isomorphic to a subgraph of G_2 and the clique problem can be verified, then the subgraph-isomorphism problem can also be verified in polynomial time and vice versa.

We can then conclude that we can reduce the clique problem to the subgraph-isomorphism problem and verify a solution in polynomial time, which means that the problem is also in NP-Hard.

Thus, the subgraph-isomorphism problem is NP-Complete, as it is both in NP and NP-Hard.

1.6 34.5-3

The *integer linear-programming problem* is like the 0-1 integer-programming problem given in Exercise 34.5-2, except that the values of the vector x may be any integers rather than just 0 or 1. Assuming that the 0-1 integer-programming problem is NP-hard, show that the integer linear-programming problem is NP-complete.

Answer:
0-1 integer programming problem gave us an integer matrix A and an integer vector b where we had to determine whether a vector x exists such that Ax $\leq$ b where $x \in \{0,1\}^n$ which is known to be NP complete. For Integer Linear programming we also have an integer matrix Z and an integer vector b but now the vector x is can take any integer values.

We start by showing that ILP is in NP. The solution to an ILP is an integer vector x, to verify if x satisfies the $Ax \leq b$ we calculate the product of Ax and check if each component is $\leq$ to the corresponding b. As this can be done in polynomial time ILP is in NP. Now we prove that ILP is NP hard by reducing the 0-1 integer programming problem (0-1 IPP)to ILP. We can see that the 0-1 IPP is a specific case of ILP where the components of x are restricted to be 0 or 1. We will add the restriction to the ILP instance by adding the constraints $0 \leq x_i \leq 1$ for every variable x_i and require that it is an integer. This constraint makes it so x_i can only be 0 or 1 due to integer constraints. Using these constraints we can transform any instance of 0-1 IPP into an ILP which can be done in polynomial time. Since we had that 0-1 IPP is NP hard and we can reduce it to an ILP in polynomial time, ILP must also be an NP hard. Since we showed that ILP is in NP ILP is NP complete.

1.7 34.5-6

Show that the hamiltonian-path problem is NP-complete.

Answer:

HAM-PATH $\in$ NP:

First, we show that HAM-PATH $\in$ NP. Given $\langle G, u, v\rangle$, a suitable certificate is an ordering $w_1, \ldots, w_{|V|}$ of G's vertices with $w_1 = u$, $w_{|V|} = v$, and each vertex appearing exactly once. In polynomial time, we verify that $(w_i, w_{i+1}) \in E$ for all $1 \leq i < |V|$. Therefore, HAM-PATH $\in$ NP.

Any language in NP reduced to HAM-PATH:

Next, we have to show that every problem to NP class is polynomial time reducible to HAM-PATH to show it is NP-complete. We show HAM-PATH is NP-hard by reducing from the known NP-complete problem HAM-CYCLE taken from Theorem 34.13. This means that we have to show HAM-CYCLE $\leq_P$ HAM-PATH.

Given an instance $G = (V, E)$ of HAM-CYCLE, construct $G' = (V', E')$ by adding two vertices s and t and connecting them to every vertex in V:

$$\begin{aligned} V' &= V \cup \{s, t\}, \\ E' &= E \cup \{(s, w) : w \in V\} \cup \{(t, w) : w \in V\} \end{aligned}$$

So if G has a HAM-CYCLE $(v_1, v_2, \ldots, v_{|V|}, v_1)$, then we can form a HAM-PATH in G' that starts as s, that goes through $(v_1, v_2, \ldots, v_{|V|}, v_1)$, and ends at t. The path in G' uses all vertices of G exactly once and connects s to t. Therefore, if G has a HAM-CYCLE, G' has a HAM-PATH from s to t.

Inversely, if G' has a HAM-PATH $(s = x_1, x_2, \ldots, x_{|V|+1} = t)$, then $x_2, x_3, \ldots, x_{|V|}$ are precisely all the vertices of V. Since s and t are connected to every vertex in G, we can follow these vertices and then return to x_2, forming a HAM-CYCLE in G. This cycle includes every vertex in V exactly once.

This means that, since HAM-CYCLE reduces to HAM-PATH in polynomial time, and HAM-CYCLE is NP-complete, HAM-PATH is NP-hard. Combined with HAM-PATH $\in$ NP, we conclude that HAM-PATH is NP-complete.

1.8 34.5-7

The longest-simple-cycle problem is the problem of determining a simple cycle (no repeated vertices) of maximum length in a graph. Formulate a related decision problem, and show that the decision problem is NP-complete.

Answer:

Decision problem: Given an undirected graph $G = (V, E)$ and an integer $k \leq |V|$, is there a simple cycle in G containing at least k distinct vertices?

LSC $\in$ NP:

To show that the problem is in NP, we use a cycle of length at least k as a certificate. This certificate consisting of a cycle $(v_1, v_2, \ldots, v_m)$ where $m \geq k$. In polynomial time, we can verify that all v_i are distinct, each edge (v_1, v_{i+1}) is in E for $1 \leq i < m$. And the edge v_m, v_1 is in E. Because these checks are done in $O(|V|^2)$ time, the problem is in NP.

LSC is NP-hard (reduce HAM-CYCLE):

We reduce from the Hamiltonian-cycle problem, which is known to be NP-complete. Given an instance $G = (V, E)$ for HAM-CYCLE, we ask the LSC decision problem on the same graph G with $k = |V|$. We get that:

- If G has a Hamiltonian cycle, then there is a cycle using all $|V|$ vertices. Thus, the answer to the LSC problem (with $k = |V|$) is "yes."
- If G does not have a Hamiltonian cycle, then there is no cycle containing all $|V|$ vertices, so the LSC problem (with $k = |V|$) must answer "no."

This polynomial-time reduction shows that LSC is at least as hard as HAM-CYCLE. Since HAM-CYCLE is NP-complete, LSC is NP-hard.

Since LSC is both in NP and NP-hard, it follows that LSC is NP-complete.