Problem Set 4

Instructions

Resources useful to solve the exercises in this problem set are the following:

Sipser's video lectures

- Lecture 12: Time Complexity
- Lecture 13: P and NP, SAT, poly-time reducibility
- Lecture 15: NP-completeness

Books

- (Sipser 2013, § 7)
- (Hopcroft, Motwani, and Ullman 2007, § 10 and § 11.1)

Cases, Rafel, and Lluís Màrquez. 2003. *Llenguatges, Gramàtiques i Autòmats : Curs Bàsic*. 2a ed. Edicions UPC.

Hopcroft, John E., Rajeev Motwani, and Jeffrey D. Ullman. 2007. *Introduction to Automata Theory, Languages, and Computation*. 3rd edition. Pearson Addison Wesley.

Sipser, Michael. 2013. *Introduction to the Theory of Computation*. 3rd edition. Cengage Learning.

All exercises

Exercise 4.1 (Travelling salesperson problem)**.** Suppose that we are given an instance of the **Travelling Salesperson Problem** (`TSP`) with n cities and distances d_{ij}. For each subset S of the cities **excluding city** 1, and for each $j \in S$, define $c[S, j]$ to be the shortest path that starts from city 1, visits all cities in S and ends up in city j.

(a) Give an algorithm that calculates $c[S, j]$ by *dynamic programming*, that is, progressing from smaller to larger sets S and using a recurrent definition of $c[S, j]$. Show that this algorithm solves the `TSP` in time $O(n^2 2^n)$. What are the space requirements of the algorithm?

(b) Suppose we wish to find the shortest (in the sense of sum of weights) path from 1 to n, not necessarily visiting all cities. Argue why this problem can be solved in polynomial time.

Exercise 4.2 (Searching for cliques)**.** A *k-clique* in a graph $G = (V, E)$ is a complete subgraph of order k of G, that is, a subgraph on k vertices having all possible edges between them. We say that G' is a *clique* in G if G' is a k-clique in G for some k. Recall the following well-known NP-complete problem.

`Clique`: Given an undirected graph G and a positive integer k, determine whether G contains a k-clique.

Some problems related to `Clique` are the following.

(a) *Maximal clique.* A clique is *maximal* if it cannot be enlarged, that is, if there is no larger clique containing it. Describe an algorithm as efficient as possible that, given an undirected graph G, finds a maximal clique of G. What's the cost of your algorithm?

(b) *Maximum clique.* A clique is *maximum* if there is no other clique having more vertices (a maximum clique is always maximal but the reverse inclusion is not always true). Describe an algorithm as efficient as possible that, given an undirected graph G, finds a maximum clique of G. What's the cost of your algorithm?

(c) *Planar clique.* Recall that a graph is *planar* if it can be drawn on the plane without edge crossings. Let `PlanarClique` be the following problem: given a planar undirected graph G and a positive integer k, does G have a k-clique? Show that `PlanarClique` has a polynomial-time algorithm.

> **Note**
>
> Planarity testing can be done in linear time (on the number of vertices). You can assume the existence of these algorithms for the exercise.

(d) *Half clique.* Let `HalfClique` be the following problem: given an undirected graph G, does G have a clique with at least $\lceil \frac{|V(G)|}{2} \rceil$ vertices? Prove that `HalfClique` is **NP**-complete.

Exercise 4.3 (Closure properties of **P**, **NP** and **coNP**)**.** The goal of this exercise is to revise some closure properties of **P**, **NP**, and **coNP**.

1. (*closure w.r.t union*) Prove the following implications
   (a) Given A and B in **P**, $A \cup B \in \mathbf{P}$.
   (b) Given A and B in **NP**, $A \cup B \in \mathbf{NP}$.
   (c) Given A and B in **coNP**, $A \cup B \in \mathbf{coNP}$.
2. (*closure w.r.t intersection*) Prove the following implications
   (a) Given A and B in **P**, $A \cap B \in \mathbf{P}$.
   (b) Given A and B in **NP**, $A \cap B \in \mathbf{NP}$.
   (c) Given A and B in **coNP**, $A \cap B \in \mathbf{coNP}$.
3. (*closure w.r.t concatenation*) Prove the following implications
   (a) Given A and B in **P**, $A \cdot B \in \mathbf{P}$.
   (b) Given A and B in **NP**, $A \cdot B \in \mathbf{NP}$.
   (c) Given A and B in **coNP**, $A \cdot B \in \mathbf{coNP}$.

Exercise 4.4 (Polynomial-time reductions)**.** Consider the relation $\leq_m^p$ among languages and justify your answers to the following questions.

(a) *Is* $\leq_m^p$ *reflexive?* That is, does it hold that $A \leq_m^p A$ for any language A?
(b) *Is* $\leq_m^p$ *symmetric?* That is, does it hold that if $A \leq_m^p B$, then $B \leq_m^p A$ for any languages A, B?
(c) *Is* $\leq_m^p$ *antisymmetric?* That is, does it hold that $A \leq_m^p B$ and $B \leq_m^p A$ imply $A = B$ for any languages A, B?
(d) *Is* $\leq_m^p$ *transitive?* That is, does it hold that $A \leq_m^p B$ and $B \leq_m^p C$ imply $A \leq_m^p C$ for any languages A, B, and C?

i Polynomial many-one reductions (or Karp reductions)

Recall that given two languages A, B over the same alphabet Σ, we say that A polynomial-time reduces to B (denoted as $A \leq_m^p B$ or $A \leq_p B$) if there exists a total function $f : \Sigma^* \to \Sigma^*$ computable in polynomial time s.t. for every $w \in \Sigma^*$, $w \in A$ if and only if $f(w) \in B$.
A useful property is the closure of the classes **P**, **NP** i **coNP** under polynomial-time reductions, that is, given $A \leq_m B$,

- if $B \in \mathbf{P}$, then $A \in \mathbf{P}$,
- if $B \in \mathbf{NP}$, then $A \in \mathbf{NP}$, and
- if $B \in \mathbf{coNP}$, then $A \in \mathbf{coNP}$.

The m in $\leq_m$ stands for the fact that f is *many-one*, that is, not necessarily injective.

Exercise 4.5 (Closure under the Kleene star)**.**

(a) Show that **P** is closed under the Kleene star.

> Tip
>
> Use dynamic programming. Given a set $A \in \mathbf{P}$ over Σ and an input $x = x_1 \dots x_n$ for $x_i \in \Sigma$, build a table indicating for each $i < j$ whether the substring $x_i \dots x_j$ is in A^*.

(b) Show that **NP** is closed under the Kleene star.

Exercise 4.6 (Search *vs* decision)**.** Show the following consequences of the hypothesis $\mathbf{P} = \mathbf{NP}$.

(a) There is a polynomial-time algorithm that produces a satisfying assignment when given a satisfying Boolean formula.

(b) Integers can be factored in polynomial time.

(c) There is a polynomial-time algorithm that takes an undirected graph as input and finds a largest clique (see exercise 7.2) contained in that graph.

> **i** Note
>
> The algorithms you are asked to provide compute a function, but **NP** contains languages, not functions. The $\mathbf{P} = \mathbf{NP}$ assumption implies that deciding satisfiability, compositeness, and the existence of cliques of a given size is all solvable in polynomial time. But even though the assumption does not show how solutions are found, you must show that you can find them anyway.

Exercise 4.7 (Berman's theorem)**.** A language is called *unary* if every string in it is of the form 1^n for some $n \geq 0$.

Prove Berman's theorem (1978): if a unary language is **NP**-complete, then $\mathbf{P} = \mathbf{NP}$.

> Tip
>
> Use the fact that `SAT` is **NP**-complete and, by hypothesis, reducible to a unary language via some function f. To decide `SAT`, given an input Boolean formula ϕ, consider the tree whose root (level 0) is ϕ and whose formulas at level k are obtained from the ones at level $k-1$, where the k-th variable takes the 2 possible truth values. Show how f allows us to explore this tree in polynomial time.

Exercise 4.8 (**NP** and HALT)**.** Show that HALT is **NP**-hard. Is it **NP**-complete?

> **Note**
>
> Recall that
>
> $$\texttt{HALT} = \{\langle x, y\rangle \mid M_x(y)\downarrow\} \ ,$$
>
> where M_x is the Turing machine with Gödel number x and the $\downarrow$ means that the machine terminates.

Exercise 4.9 (`UniqueSAT`)**.** The class **DP** (for *difference polynomial time*) is defined as the set of languages L for which there are two languages $L_1 \in \mathbf{NP}$, $L_2 \in \mathbf{coNP}$ such that $L = L_1 \cap L_2$. Notice that since $\overline{L_2} \in \mathbf{NP}$, L is the difference of two **NP** sets (but do not confuse **DP** with $\mathbf{NP} \cap \mathbf{coNP}$, which may seem superficially similar).

Consider the problem `UniqueSAT` on determining whether a Boolean formula has a unique satisfying assignment (or model) and show the following facts.

(a) `UniqueSAT` $\in \mathbf{DP}$.
(b) If `UniqueSAT` is in **NP**, then $\mathbf{NP} = \mathbf{coNP}$.

Exercise 4.10 (Are they in **P**?)**.** Prove that the following languages on undirected graphs are in **NP**. Which ones of them are in **P**?

(a) *Two coloring.* $\texttt{2COL} = \{G \mid \text{graph } G \text{ has a coloring with two colors}\}$, where a k-coloring of G is an assignment of a number in $\{1, \ldots, k\}$ to each vertex of G such that no adjacent vertices get the same number.

(b) *Three coloring.* $\texttt{3COL} = \{G \mid \text{graph } G \text{ has a coloring with three colors}\}$.

(c) *Hamiltonian path.* $\texttt{HP} = \{\ G \mid \text{graph } G \text{ has a Hamiltonian path}\}$, where a path is said to be *Hamiltonian* if it visits every vertex exactly once.

(d) *Connectivity.* $\texttt{CONNECTED} = \{G \mid \text{graph } G \text{ is a connected graph}\}$.