CSS.201.1 Algorithms

Instructor: Umang Bhaskar

TIFR 2024, Aug-Nov

Scribe: Soham Chatterjee

soham.chatterjee@tifr.res.in
Website: sohamch08.github.io

Contents

Chapter 1

P, NP and Reductions

Almost all the algorithms we have studied thus far have been **polynomial time algorithms** i.e. on inputs of size n, their worst-case running time is $O(n^k)$ for some constant k. A natural question to ask is whether all problems can be solved in polynomial time. The answer is no.

There are problems that can be solved but not in polynomial time and there are problems which can not be solved via an algorithm. To discuss problems in general think of computational tasks as language recognition problem. A language is a subset of $\{0,1\}^*$. For example:

$$L_{\text{CONN}} = \{x \in \{0,1\}^* \mid x \text{ represents a connected graph}\}$$

So main problem we want to think about is to decide whether a given string is in the language or not. These problems are also called decision problems.

Definition 1.1: Decision Problems

Given a language $L \subseteq \{0,1\}^*$ and a string $x \in \{0,1\}^*$ decide whether $x \in L$ or not.

An algorithm $\mathcal{A}$ solves this problem if $x \in L \iff \mathcal{A}(x) = 1$. Time complexity of $\mathcal{A}$: $T_{\mathcal{A}}(n)$ is the maximum running time of $\mathcal{A}$ on any string x of length n. Since we can work over any set of alphabets and alphabets can be encoded into binary we will say languages are subset of Σ^* where Σ is the finite set of alphabets.

1.1 Introduction to Complexity Classes

Depending on time, space and some other resources based on how much they are used we divide the computational problems into several sets. We call these sets as complexity classes.

Definition 1.1.1: Polynomial Running Time

A language $L \subseteq \Sigma^*$ has a polynomial-time algorithm if there exists $\mathcal{A}$ that solves L and $T_{\mathcal{A}}(n) = O(n^k)$ for some constant k.

Now we introduce our first complexity class now. This class is called P.

$$\text{P} := \{L \subseteq \Sigma^* \mid \text{ there exists a polynomial time algorithm that decides } L\}$$

Till now all the algorithms we have studied are in P.

Question 1.1

What about $L_{\text{SAT}}, L_{\text{3COL}}, L_{\text{2SAT}}, L_{\text{CONN}}$?

We know $L_{\text{CONN}} \in \text{P}$ since we can run a DFS to check if all the vertex is reachable from a vertex. Also we know $L_{\text{2SAT}} \in \text{P}$. For other languages we don't know if they are in P. But these problems have another noticable nature. Given

a potential solution for the problem one can check if that is indeed a solution of the problem or not in polynomial-time. Let's abstract this notion:

Definition 1.1.2: Short Certificate of Membership

A language have a short certificate of membership if there exists an algorithm $\mathcal{A}$ that runs in polynomial time and

$$\forall\, x \in L,\ \exists\, y \in \text{poly}(|x|), \quad \mathcal{A}(x, y) = 1$$
$$\forall\, x \notin L,\ \forall\, y \in \text{poly}(|x|), \quad \mathcal{A}(x, y) = 0$$

What are the certificates or above-mentioned problems.

- L_{SAT}: Assignment of the variables. Then we can verify if every clause is satisfied
- L_{3COL}: Coloring of the edges. We can verify if all the edges follows the coloring constraint.
- L_{CONN}: A spanning tree. We can verify if every vertex is present there.

Now we introduce another complexity class called NP.

$$\text{NP} := \{L \subseteq \Sigma^* \mid L \text{ has a short certificate of membership}\}$$

For NP we call the algorithm to check for the certificate *verifier*. Another way to think about the class NP is to extend the computer to make "guesses" or exists in multiple state simultaneously. This is known as non-determinism. Then NP is the class of languages decided by a non-deterministic Turing machine in polynomial time. For example a non-deterministic algorithm for 3SAT is

- Make a guess for the assignment for each variable.
- If ϕ is satisfied return yes else return no.

Naturally any problem which is in P has a short certificate.

Theorem 1.1.1

P $\subseteq$ NP.

Another complexity class which come associated with NP is coNP.

$$\text{coNP} := \overline{\text{NP}}$$

i.e. the complement set of NP.

Observation 1.1. $P = \overline{P}$

Theorem 1.1.2

P $\subseteq$ coNP

Apart from these two we don't know any relation between NP and coNP whether they are equal or not.

1.2 Reductions

Question 1.2

What does it mean for a problem to be at least as hard as another?

To relate hardness of one problem to another we introduce the notion of reductions. There are many reductions. We will only focus on polynomial-time many-one karp reduction.

Definition 1.2.1: Many-One Karp Reduction

$L_1, L_2 \subseteq \Sigma^*$ are two languages. L_1 is reducible to L_2 under polynomial time many-one karp reduction if and only if there exists a polynomial time computable function $f : \Sigma^* \to \Sigma^*$ such that $\forall\, x \in \Sigma^*$

$$x \in L_1 \iff f(x) \in L_2$$

and we denote it by $L_1 \leq_m^{\text{poly}} L_2$.

We call a language L to be NP-hard if for every language $L' \in \text{NP}$, $L' \leq_m^{\text{poly}} L$. And L is called NP-complete if $L \in \text{NP}$ and L is NP-hard.

Theorem 1.2.1 Cook's Theorem

3SAT is NP-complete.

Corollary 1.2.2

$\overline{\text{SAT}}$ is coNP-complete.