Formal Languages and Automata, 2025 Fall semester

Lec 21. Class P and NP

Eunjung Kim

What we learnt so far

- A simple form of a computational problem as a language, or equivalently, a decision problem.
- Different ways of expressing a language: regular expression, grammar, machine recognizability
- Models of computations with increasing computational power:
  1. finite-state automata
  2. pushdown automata
  3. Turing machines
- Limit of computations / limit of expressibility of language classes:
  1. pumping lemma for regular and context-free languages
  2. diagonal method, Turing- and many-one reduction for decidable and recognizable languages
- Essentially: the capabilities and limit of computing devices, as in, possible versus impossible.

Computational Complexity

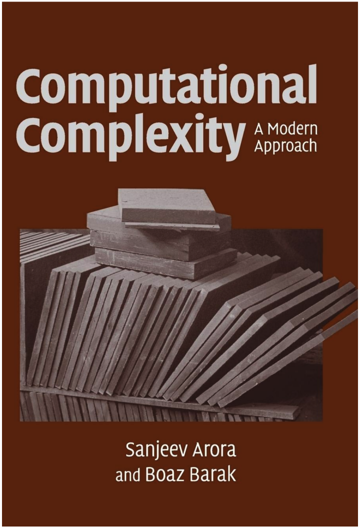
When you know that a decision problem is decidable

- want to design an algorithm which requires a small amount of resources
- analyzing the computational resources of an algorithm: time, memory
- time complexity, space complexity

Different computation model

- (it is believed that) no computational device, if physically implementable, can compute what Turing machine cannot. Be it based on DNA, neurons, quantum entanglement...
- However, various computation models emphasizing different aspects of computations are devised, implemented, and studied.
- e.g. the number of processors, the number of rounds of communications, the number of gates in a circuit, etc.

Computational Complexity



Time complexity

Running time / time complexity

Let M be a deterministic TM that halts on all inputs. The running time or time complexity of M is the function $f : \mathbb{N} \to \mathbb{N}$ such that the maximum number of steps that M takes over all input strings of length n.

- In the above definition, we use worst-case analysis, the most commonly used analysis (but not the only one) for time complexity analysis.
- The exact function f is well-defined, but complex to express.
- We use the asymptotic notation for expressing the function f.

Asymptotic notations

big-O, small-o notations

Let f, g be functions $\mathbb{N} \to \mathcal{R}^+$.

We say that $f(n) = O(g(n))$ if there exists positive integers c and n_0 such that for all integers $n \geq n_0$, we have $f(n) \leq c \cdot g(n)$.

We say that $f(n) = o(g(n))$ if for all positive real numbers $c > 0$. there exists a positive integer c and n_0 such that for all integers $n \geq n_0$, we have $f(n) < c \cdot g(n)$.

Some texts write as $f(n) \in O(g(n))$, respectively $f(n) \in o(g(n))$.

Time complexity: example 1

Consider a single-tape TM which decides the language $A = \{0^k 1^k \mid k \geq 0\}$.

A decider of A

Upon input string $w \in \{0,1\}^*$, M_1 does the following.

1. Scan across the tape and reject if a 0 is found to the right of a 1.
2. Repeat while there is at least one 0 and at least one 1:
   cross off the first remaining 0 and 1.
3. If there is a remaining 0 (while no 1 left), or vice versa, reject.
4. If neither 0 nor 1 is left on the tape, accept.

What is the running time of the above algorithm M_1? (worst-case, asymptotic)

Time complexity: example 2

Consider a single-tape TM which decides the language $A = \{0^k 1^k \mid k \geq 0\}$.

A decider of A

Upon input string $w \in \{0, 1\}^*$, M_2 does the following.

1. Scan across the tape 1 and reject if a 0 is found to the right of a 1.
2. Repeat while there is at least one 0 and at least one 1:

   cross off every other 0, starting from the first remaining 0.
   Do the same for 1.

3. If there is a remaining 0 (while no 1 left), or vice versa, reject.
4. If neither 0 nor 1 is left on the tape, accept.

What is the running time of the above algorithm M_2? (worst-case, asymptotic)

Time complexity: example 3

Consider a two-tape TM which decides the language $A = \{0^k 1^k \mid k \geq 0\}$.

A decider of A

Upon input string $w \in \{0,1\}^*$, M_3 does the following.

1. Scan across the tape 1 and reject if a 0 is found to the right of a 1.
2. Scan across the 0's on tape 1 until the first 1 is detected. Simultaneously, copy the 0s onto tape 2.
3. Scan across the 1's on tape 1 until the end of the input. For each 1 read on tape 1, cross of a 0 on tape 2.
4. If all 0's are crossed off before all 1's are read, reject.
5. Once all 1's are read, reject if there is any 0 left. Accept otherwise.

What is the running time of the above algorithm M_3? (worst-case, asymptotic)

Computability vs computational complexity

- When it comes to decidabiltiy (or computability in general), all TMs are equivalent.
- For time complexity (computational complexity), the choice of machine model matters.

However

Polynomial equivalence of deterministic TMs

All (reasonable) deterministic Turing machines are polynomially equivalent. That is, one can simulate the other with only a polynomial-factor increase in running time.

The Class P

TIME($T(n)$)

Let $T : \mathbb{N} \rightarrow \mathbb{N}$ be a function. A language L is said to be in T$IME(T(n))$ if there exists a deterministic TM which decides L and runs in time $O(T(n))$ on every input of length n.

- Robust agains the choice of particular TM model, as long as it is deterministic.

The Class P

The class P is defined as $\bigcup_{c \geq 1}$ TIME(n^c). In other words, P is the class of all languages which can be decided in polynomial time on a deterministic single-tape Turing machine.

Time complexity

Running time / time complexity

Let N be a nondeterministic TM that halts in all computation branches over all input strings. The running time or time complexity of N is the function $f : \mathbb{N} \to \mathbb{N}$ such that the maximum number of steps that N takes over all computation branches on any input strings of length n.

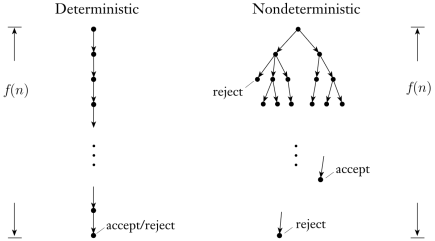


Figure 7.10, Sipser 2012.

The Class NP: Nondeterministic Polynomial-time

NTIME($T(n)$)

Let $T : \mathbb{N} \to \mathbb{N}$ be a function. A language L is in N$TIME(T(n))$ if there exists a nondeterministic TM which decides L and runs in time $O(T(n))$ on every input of length n.

The Class NP

The class NP is defined as $\bigcup_{c \geq 1} \mathrm{NTIME}(n^c)$.

NP stands for "Nondeterministic Polynomial time" (not "Not Polynomial")!

Polynomial-time Verifier

Polynomial-time verifier

A verifier for a decision problem $A \subseteq \{0, 1\}^*$ is an algorithm V such that

$x \in A$ if and only if there exists a string w such that V accepts $\langle x, w \rangle$.

We say that V is a polynomial-time verifier for A if it runs in polynomial time in $|x|$.

The additional string w used for testing whether $x \in A$ is called a proof / certificate / witness for the membership in A of input string x.

Notice that if V is a polynomial-time verifier for A, then there is a witness of length $poly(|x|)$ for every $x \in A$.

Equivalence of P-time verifiability and NP

Equivalence of the two notions

A language A is in NP if and only if it admits a polynomial-time verifier.

($\Rightarrow$) From a nondeterministic TM deciding A, we build a P-time verifier V.

V upon the input (x, w):

1. simulates N on the input string x, treating w as the instruction sequence for choosing the transition (amongst $|Q| \cdot |\Gamma| \cdot |\{L, R, S\}|$ possible choices).
2. If N accepts, V accepts; if N rejects, V rejects.

Equivalence of P-time verifiability and NP

Equivalence of the two notions

A language A is in NP if and only if it admits a polynomial-time verifier.

($\Leftarrow$) From a polynomial-time verifier V for A, we build a nondeterministic TM N deciding A.

N upon the input x:

1. nondeterministically guesses (and writes) a string w of length $poly(|x|)$.
2. runs V on (x, w).
3. If V accepts, N accepts; if V rejects, N rejects.

Example of Polynomial Verifier

Clique

A clique of an undirected graph $G = (V, E)$ is a set $K \subseteq V$ of vertices such that any pair of K are adjacent.

The (decision) problem CLIQUE asks if an input graph G contains a clique of size k.

Input:	a graph $G = (V, E)$, a non-negative integer k.
Question:	does G contain a clique of size k?

As a language,

$$CLIQUE = \{\langle G, k\rangle \mid G \text{ is a graph with a clique of size } k\}.$$

CLIQUE IS IN NP

(via witness) consider the following verifier V: upon the input $\langle\langle G, k\rangle, K\rangle$

1. Test if K is a vertex subset of G.
2. Test if G has an edge between any pair of K.
3. If both pass, accept; otherwise, reject.

(via nondeterministic TM) consider the following verifier nondeterministic N: upon the input $\langle G, k\rangle$

1. Nondeterministically write k vertex names of G; denote them by K.
2. Test if G has an edge between any pair of K.
3. If yes accept; otherwise, reject.

The relationship between P and NP

Loosely speaking,

- The class P is the class of languages for which membership can be decided quickly.
- The class NP is the class of languages for which membership can be verified quickly.

As $P \subseteq NP$ (why?) there are two possibilities. Determining which one is the case, often called P versus NP problem, is one of the biggest open question in TCS and mathematics.

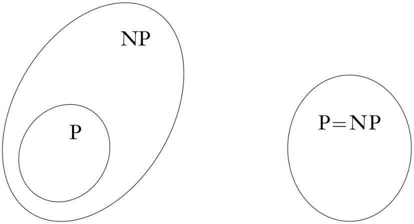


Figure 7.26, Sipser 2012.