A Brief Introduction to Computational Complexity

CS402 (Principles of Programming Languages)

Albin James Maliakal

National Institute of Technology Nagaland

November 12, 2024

Overview

Overview

Computational Complexity?

The study of the computational resources required for solving a given **problem**.

Examples of Problems:

- Given an integer, decide if it is prime.
- Given a set of integers, compute their average.
- Given a set of integers, sort them in ascending order.
- Given a map, find the shortest route from a to b.
- Given a set of constraints, find a solution that satisfies them all.
- Given a graph, decide if...
- ...it is connected.
- ...it admits a 3-coloring.
- ...it contains a clique of size 5.
- ...

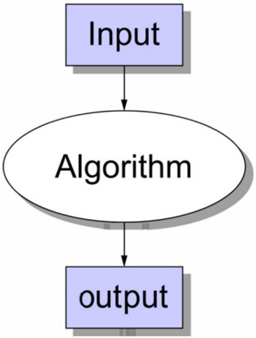
Definition of a Problem

Formally, a problem is a relation between two domains: the input domain and the output domain. Equivalently, it is a function from the input domain to the output domain.

Since computers (so far...) are discrete machines, both domains can be represented as sets of words over the $\{0, 1\}$ alphabet. Thus, a problem is a function from binary words to binary words:

$$\text{Problem } \Pi : \{0, 1\}^* \rightarrow \{0, 1\}^*$$

An algorithm solves Π if it computes $\Pi(x)$ for any $x \in \{0, 1\}^*$ (or in a prescribed subset of $\{0, 1\}^*$).



Overview

Types of Problems

Different types of problems (examples based on coloring of a graph G):

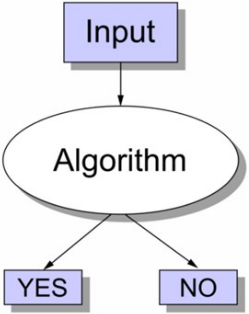
- **Decision:** Does G admit a k-coloring (for a given k)?
- **Search:** Find a k-coloring of G (for a given k).
- **Counting:** How many k-colorings does G admit (for a given k)?
- **Optimization:** What is the smallest k such that G admits a k-coloring?

Decision Problem (Focus)

A special case where $\Pi : \{0,1\}^* \rightarrow \{0,1\}$ (the answer is YES or NO).
$x \in \{0,1\}^*$ is a positive instance if $\Pi(x) = 1$ (resp. negative if 0).

Positive instances define a "formal language" (i.e., a set of words) $L \subseteq \{0,1\}^*$.

Point of View: Solving a decision problem boils down to deciding if a given word is in L.



Overview

What Questions?

Goal of computational complexity:

$\rightarrow$ classify the problems according to the resources needed for solving them.

What Type of Resources?

- **Time** (= number of operations)
- **Space** (= memory)
- **Randomness** (number of random bits)
- **Non-determinism**
- ...

Asymptotic Point of View

- What matters is how these quantities scale with the size of the input, noted n.
- Notations: $O(\cdot)$, $\Omega(\cdot)$, $\Theta(\cdot)$, $o(\cdot)$, $\omega(\cdot)$
- Intuition: $\leq$, $\geq$, $=$, $<$, $>$ as $n \to \infty$, up to constant factors.
- Examples:
  - Constant: $O(1)$
  - Logarithmic: $O(\log n)$
  - Linear: $O(n)$
  - Quasi-linear: $O(n \log n)$
  - Quadratic: $O(n^2)$
  - Polynomial: $O(n^c) = n^{O(1)} = \text{poly}(n)$
  - Exponential: $O(2^n)$ or $O(2^{\text{poly}(n)})$
  - Factorial: $O(n^n)$

Overview

Space and Time

Unless otherwise mentioned, we focus on decision problems.

Generic Classes

- $\mathrm{TIME}(f(n))$: Set of problems solvable in time $O(f(n))$.
- $\mathrm{SPACE}(f(n))$: Set of problems solvable in space $O(f(n))$.

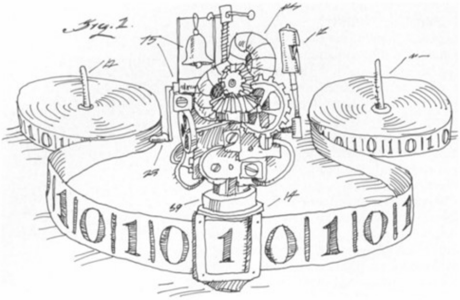
Main Classes of Complexity

$$\begin{aligned} L &:= \mathrm{SPACE}(\log n) && \text{Problems solvable in logarithmic space} \\ P &:= \mathrm{TIME}(\mathrm{poly}(n)) && \text{Problems solvable in polynomial time} \\ \mathrm{PSPACE} &:= \mathrm{SPACE}(\mathrm{poly}(n)) && \text{Problems solvable in polynomial space} \\ \mathrm{EXP} &:= \mathrm{TIME}(2^{\mathrm{poly}(n)}) && \text{Problems solvable in exponential time} \end{aligned}$$

$$L \subseteq P \subseteq \mathrm{PSPACE} \subseteq \mathrm{EXP}$$

Machine Model?

- Turing machines
- Intuition: “standard algorithm” OK
- Universality?



Space versus Time (continued)

Theorems and Hierarchies (Simplified)

Given two functions $f_1(n)$ and $f_2(n)$ (abbreviated as f_1 and f_2 below):

- **Time hierarchy:**
  $f_1 = o(f_2 \log f_2) \Rightarrow \text{TIME}(f_1) \subsetneq \text{TIME}(f_2)$ (Stearns and Hartmanis '65)
- **Space hierarchy:**
  $f_1 = o(f_2) \Rightarrow \text{SPACE}(f_1) \subsetneq \text{SPACE}(f_2)$

Space versus Time?

$$\text{TIME}(f(n)) \subseteq \text{SPACE}\left(\frac{f(n)}{\log(f(n))}\right)$$ (Hopcroft, Paul, Valiant '77)

$\rightarrow$ *Space is strictly better than time!*

Overview

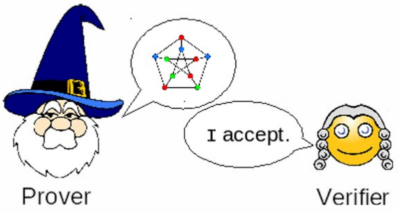
Non-determinism (NP and coNP)

Several equivalent definitions exist, with the simplest being:

- NP: $\exists$ short proof when the answer is YES (positive certificate)
- coNP: $\exists$ short proof when the answer is NO (negative certificate)

Short proof = one that is verifiable in polynomial time.

Example: 3-COLORATION $\in$ NP (certificate = the coloring itself)



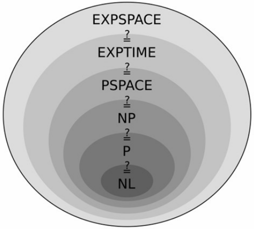
Non-determinism (continued)

More Generally

- NTIME($f(n)$): $\exists$ positive certificates verifiable in time $O(f(n))$.
- NSPACE($f(n)$): $\exists$ positive certificates verifiable in space $O(f(n))$.
- coNTIME and coNSPACE: Same for negative certificates.

Main Classes (Non-deterministic Version)

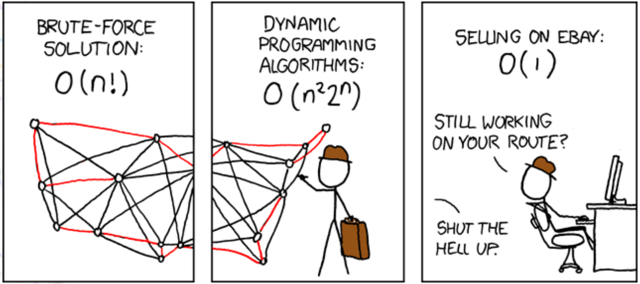
$$NL := \text{NSPACE}(\log n)$$
$$NP := \text{NTIME}(\text{poly}(n))$$
$$NPSPACE := \text{NSPACE}(\text{poly}(n))$$
$$NEXP := \text{NTIME}(2^{\text{poly}(n)})$$



Relations

- $P \subseteq NP \cap coNP$ (algorithm = verifier with an empty certificate)
- NSPACE($f(n)$) $\subseteq$ SPACE($f^2(n)$) (Savitch'70)
- NSPACE($f(n)$) = coNSPACE($f(n)$) (Immerman–Szelepcsényi'87)
- NTIME($f(n)$) $\subseteq$ SPACE($f(n)$) (Certificate enumeration)
- $L \stackrel{?}{=} NL \stackrel{?}{=} P \stackrel{?}{=} NP \stackrel{?}{=}$ PSPACE $\stackrel{?}{=}$ EXP $\stackrel{?}{=}$ NEXP $\stackrel{?}{=}$ EXPSPACE (Inclusions OK)
- Padding arguments: $P = NP \Rightarrow EXP = NEXP$ (and others)
- $P \stackrel{?}{=} NP$ Does "easy to check" imply "easy to compute"?

Overview


BRUTE-FORCE SOLUTION:
O(n!)
DYNAMIC PROGRAMMING ALGORITHMS:
O(n²2ⁿ)
SELLING ON EBAY:
O(1)
STILL WORKING ON YOUR ROUTE?
SHUT THE HELL UP.

Thanks!