Advanced Computational Complexity

George Voutsadakis[1]

[1]Mathematics and Computer Science
Lake Superior State University

LSSU Math 600

1. Notation
   - Standard Conventions
   - Representing Objects as Strings
   - Decision Problems/Languages
   - Big-Oh Notation

Subsection 1

Standard Conventions

Standard Notational Conventions

- Let $\mathbb{Z} = \{0, \pm 1, \pm 2, \ldots\}$ denote the set of integers.
- Let $\mathbb{N}$ denote the set of natural numbers (i.e., nonnegative integers).
- A number denoted by one of the letters i, j, k, ℓ, m, n is always assumed to be an integer.
- If $n \geq 1$, then $[n]$ denotes the set $\{1, \ldots, n\}$.
- For a real number x, we denote by:
  - $\lceil x \rceil$ the smallest $n \in \mathbb{Z}$, such that $n \geq x$;
  - $\lfloor x \rfloor$ the largest $n \in \mathbb{Z}$, such that $n \leq x$.
- If a real number is used in a context requiring an integer, the operator $\lceil\ \rceil$ is implied.

Standard Notational Conventions (Cont'd)

- We denote by $\log x$ the logarithm of x to the base 2.
- We say that a condition $P(n)$ holds for **sufficiently large** n if, there exists some number N, such that $P(n)$ holds, for every $n > N$.
- We use expressions such as $\sum_i f(i)$ (as opposed to, say, $\sum_{i=1}^n f(i)$) when the range of values i takes is obvious from the context.
- If u is a string, u_i denotes the value of the i-th symbol of u.
- If u is a vector, u_i denotes the value of the i-th coordinate of u.

Strings

- Let S be a finite set (of symbols).
- A **string over the alphabet** S is a finite ordered tuple of elements from S.
- We will typically consider strings over the binary alphabet $\{0, 1\}$.
- For any integer $n \geq 0$, we denote by S^n the set of length-n strings over S (S^0 denotes the singleton consisting of the empty tuple).
- We denote by S^* the set of all strings,

$$S^* = \bigcup_{n \geq 0} S^n.$$

Concatenation and Length

- If x and y are strings, then we denote their **concatenation** (the tuple that contains first the elements of x and then the elements of y) by

$$x \circ y \quad \text{or simply} \quad xy.$$

- If x is a string and $k \geq 1$ is a natural number, then

$$x^k$$

  denotes the concatenation of k copies of x.
- For example, 1^k denotes the string consisting of k ones.
- The **length** of a string x is denoted by $|x|$.

Distributions

- If S is a distribution, then we use
$$x \in_R S$$
to say that x is a random variable that is distributed according to S.
- If S is a set, then $x \in_R S$ is used to denote that x is distributed uniformly over the members of S.
- We denote by U_n the uniform distribution over $\{0,1\}^n$.

Dot Product and Inner Product

- For two length-n strings $x, y \in \{0,1\}^n$, we denote their **dot product modulo** 2 by
$$x \odot y.$$
- That is,
$$x \odot y = \sum_i x_i y_i \pmod 2.$$
- The **inner product** of two n-dimensional real or complex vectors $\mathbf{u}, \mathbf{v}$ is denoted by
$$\langle \mathbf{u}, \mathbf{v} \rangle.$$
- For any object x, we use $\llcorner x \lrcorner$ (not to be confused with the floor operator $\lfloor x \rfloor$) to denote the **representation** of x as a string.
- This is detailed in the following slides.

Subsection 2

Representing Objects as Strings

Representation of Input Objects

- Our main computational task is computing a function.
- We focus on functions whose inputs and outputs are finite strings of bits, i.e., members of $\{0,1\}^*$.
- Considering only functions that operate on bit strings is not a real restriction.
- Simple encodings can be used to represent general objects as strings of bits.
- In this way, one can represent, e.g.:
  - Integers;
  - Pairs of integers;
  - Graphs;
  - Vectors;
  - Matrices;

    $\vdots$

Examples of Representation

- We can represent an integer as a string using its binary expansion.
- E.g., 34 is represented as 100010.
- A graph can be represented as its adjacency matrix.
- That is, an n vertex graph G is represented by an $n \times n$ 0/1-valued matrix A, such that

$$A_{i,j} = 1 \quad \text{iff} \quad \text{the edge } \overline{ij} \text{ is present in } G.$$

- We will avoid dealing explicitly low-level representation issues.
- We use $\llcorner x \lrcorner$ to denote some canonical binary representation of x.
- Often $\llcorner\ \lrcorner$ is dropped and x is used for both the object and its representation.

Representing Pairs and Tuples

- We use $\langle x, y\rangle$ to denote the ordered pair consisting of x and y.
- A canonical representation for $\langle x, y\rangle$ can be obtained from the representations of x and y.

  Example: We can first encode $\langle x, y\rangle$ as the string

  $$\llcorner x \lrcorner \# \llcorner y \lrcorner,$$

  over the alphabet $\{0, 1, \#\}$.

  Then, use the mapping $0 \mapsto 00, 1 \mapsto 11, \# \mapsto 01$ to convert this representation into a string of bits.
- To reduce notational clutter, instead of $\llcorner \langle x, y\rangle \lrcorner$, we use $\langle x, y\rangle$ to also denote the representation of this pair as a binary string.
- Similarly, we use $\langle x, y, z\rangle$ to denote both the ordered triple consisting of x, y, z and its representation $\llcorner \langle x, y, z\rangle \lrcorner$.
- We adopt similar conventions for other representations.

Functions with Nonstring Inputs or Outputs

- The idea of representation allows us to talk about computing functions whose inputs are not strings.
- E.g., functions that take natural numbers as inputs.
- We implicitly identify any function f whose domain and range are not strings with the function

$$g : \{0,1\}^* \to \{0,1\}^*$$

  that:
  - Receives a representation of an object x as input;
  - Outputs the representation of $f(x)$.
- Using the representation of pairs and tuples, we can also talk about computing functions that have more than one input or output.

Subsection 3

Decision Problems/Languages

Languages or Decision Problems

- A special case of functions mapping strings to strings is the case of Boolean functions, whose output is a single bit.
- We identify such a function f with the subset

$$L_f = \{x \in \{0,1\}^* : f(x) = 1\} \subseteq \{0,1\}^*.$$

- We call such sets **languages** or **decision problems**.
- We identify the corresponding computational problems.
  - Compute f:
    Given x, compute $f(x)$.
  - Decide the language L_f:
    Given x, decide whether $x \in L_f$.

Example: Independent Set

- Let G be a graph.
- An **independent set** in G is a subset of the set of vertices, such that no edge in G joins any two of them.
- A computational problem consists, given a graph G, of finding a maximum sized independent set.
- The corresponding language is:

$$\begin{aligned}\textsc{IndSet} \ = \ \{\langle G, k\rangle : \ & \exists S \subseteq V(G) \text{ s.t. } |S| \geq k \text{ and} \\ & \forall u, v \in S, \overline{uv} \notin E(G)\}.\end{aligned}$$

- An algorithm to solve this language will tell us, on input a graph G and a number k, whether there exists an independent set in G of size at least k.

Subsection 4

Big-Oh Notation

Computational Efficiency of Algorithms

- We will typically measure the computational efficiency of an algorithm as the number of basic operations it performs as a function of its input length.
- The efficiency of an algorithm can be captured by a function T from the set $\mathbb{N}$ of natural numbers to itself, such that $T(n)$ is equal to the maximum number of basic operations that the algorithm performs on inputs of length n.
- This function T is sometimes overly dependent on the low-level details of our definition of a basic operation.

  Example: The addition algorithm will take about three times more operations if it uses addition of single digit binary numbers as a basic operation, as opposed to decimal numbers.
- To ignore these low-level details in our measurements, we introduce the big-Oh notation.

Big-Oh Notation

Definition (Big-Oh Notation)

If f, g are two functions from $\mathbb{N}$ to $\mathbb{N}$, then we say:

(1) $f = \mathrm{O}(g)$ if there exists a constant c, such that $f(n) \leq c \cdot g(n)$, for every sufficiently large n;

(2) $f = \Omega(g)$ if $g = \mathrm{O}(f)$;

(3) $f = \Theta(g)$ if $f = \mathrm{O}(g)$ and $g = \mathrm{O}(f)$;

(4) $f = \mathrm{o}(g)$ if, for every $\epsilon > 0$, $f(n) \leq \epsilon \cdot g(n)$, for every sufficiently large n;

(5) $f = \omega(g)$ if $g = \mathrm{o}(f)$.

To emphasize the input parameter, we often write $f(n) = \mathrm{O}(g(n))$ instead of $f = \mathrm{O}(g)$, and use similar notation for $\mathrm{o}, \Omega, \omega$ and Θ.

Examples

1. Let $f(n) = 100n \log n$ and $g(n) = n^2$.

   Then we have the relations

   $$f = \mathrm{O}(g), \quad g = \Omega(f), \quad f = \mathrm{o}(g), \quad g = \omega(f).$$

2. Let $f(n) = 100n^2 + 24n + 2\log n$ and $g(n) = n^2$.

   Then $f = \mathrm{O}(g)$. In this case, we often write $f(n) = \mathrm{O}(n^2)$.

   We also have $g = \mathrm{O}(f)$.

   As a result, $f = \Theta(g)$ and $g = \Theta(f)$.

Examples (Cont'd)

3. Let $f(n) = \min\{n, 10^6\}$ and $g(n) = 1$, for every n.

   Then $f = \mathrm{O}(g)$. We write $f = \mathrm{O}(1)$.

   Similarly, if h is a function that tends to infinity with n (i.e., for every c, it holds $h(n) > c$ for n sufficiently large), then we write $h = \omega(1)$.

4. Let $f(n) = 2^n$ and $g(n) = n^c$, for some $c \in \mathbb{N}$.

   Then $g = \mathrm{o}(f)$. We write $2^n = n^{\omega(1)}$.

   Similarly, we write
   $$h(n) = n^{\mathrm{O}(1)}$$
   to denote that h is bounded from above by some polynomial.

   I.e., there exist a number $c > 0$, such that, for sufficiently large n, $h(n) \leq n^c$.

   Another notation for this is $h(n) = \mathrm{poly}(n)$.