Data Structures and Algorithms

Xí Tí Kè

dongmassimo@gmail.com

December 23, 2019

Turing Machines

- Turing Machine
  - C/C++ program, like bubble sort etc.
- Universal Turing Machine
  - A program that can simulate any other programs, depends on how we encode other programs
  - A C/C++ compiler
  - An Python interpreter
  - A CPU or CPU Emulator (PA)
- Non-Deterministic Turing Machine
  - A program that can make guesses. Outputs 1 if and only if there exists a sequence of guesses that make the TM accept.

Uncomputability

Theorem

Turing Machines (programs) are countable.

Proof.

We can use programming languages to encode programs. □

Theorem

Decision problems are uncountable.

Proof.

Decision problems are mappings of the form $f : \mathbb{N} \to \{0, 1\}$. So there are $|2^{\mathbb{N}}|$ decision problems. □

Corollary

There exists a decision problem that can not be solved by any Turing Machine.

Diagonalization

Theorem

The real numbers in the range $[0, 1)$ are uncountable.

Proof.

Suppose on the contrary, the real numbers are

$f(0) = 0.a_{00}a_{01}a_{02}\dots$
$f(1) = 0.a_{10}a_{11}a_{12}\dots$
$f(2) = 0.a_{20}a_{21}a_{22}\dots$
$f(3) = 0.a_{30}a_{31}a_{32}\dots$
$\dots$

Let $r = 0.(1 - a_{00})(1 - a_{11})(1 - a_{22})\dots$.
then $f(n) \neq r$ for all $n \in \mathbb{N}$. $\square$

Diagonalization

We use M_n to denote the Turing Machine encoded by $n \in \mathbb{N}$.
Here we assumed a Universal Turing Machine.

	0	1	2	3	...
0	$M_0(0)$	$M_0(1)$	$M_0(2)$	$M_0(3)$	...
1	$M_1(0)$	$M_1(1)$	$M_1(2)$	$M_1(3)$	...
2	$M_2(0)$	$M_2(1)$	$M_2(2)$	$M_2(3)$	...
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

$M_i(j) \in \{0, 1, ?\}$

Problem (Our first uncomputable problem, UC)

$$\mathit{UC}(x) = \begin{cases} 1 - M_x(x) & \textit{if } M_x(x) \textit{ halts} \\ 0 & \textit{otherwise} \end{cases}$$

Theorem

UC is uncomputable.

Diagonalization

Problem (Our first uncomputable problem, UC)

$$UC(x) = \begin{cases} 1 - M_x(x) & \textit{if } M_x(x) \textit{ halts} \\ 0 & \textit{otherwise} \end{cases}$$

Proof.

Suppose on the contrary, Turing Machine M_n computes UC. Then M_n should halt on every input. Then, according to the definition of UC, $M_n(n) = UC(n) = 1 - M_n(n)$, a contradiction. □

HALT

Problem (Our first uncomputable problem, UC)

$$UC(x) = \begin{cases} 1 - M_x(x) & \textit{if } M_x(x) \textit{ halts} \\ 0 & \textit{otherwise} \end{cases}$$

Problem (HALT)

Given input n and x, decide whether $M_n(x)$ halts.

Theorem

HALT is uncomputable.

Proof.

If HALT is computable, then UC is computable:

- Given x, first use HALT to determine whether $M_x(x)$ halts
- No: output 0
- Yes: Simulate $M_x(x)$ and output $1 - M_x(x)$

□

Problem 34.1-6

Definition (Languages)

A language L is a set of strings. A TM M decides a language $L \subseteq \{0,1\}^*$, if M solves the decision problem

$$f(x) = \begin{cases} 1 & \text{if } x \in L \\ 0 & \text{if } x \notin L \end{cases}$$

We can use a set to denote a problem.

Definition (P)

P is the set of languages decidable by a polynomial-time TM.

Problem (34.1-6)

If $L_1, L_2 \in$ P, then $L_1 L_2 \in$ P. $L_1 L_2 = \{x_1 x_2 : x_1 \in L_1, x_2 \in L_2\}$.

Problem 34.1-6

Problem (34.1-6)

If $L_1, L_2 \in$ P, then $L_1L_2 \in$ P. $L_1L_2 = \{x_1x_2 : x_1 \in L_1, x_2 \in L_2\}$.

- M_1 decides L_1, and M_2 decides L_2.
- Our goal: construct TM M deciding L_1L_2 in polynomial time.
- Given $y \in \{0,1\}^*$, decide if $y = x_1x_2$ s.t. $x_1 \in L_1$, $x_2 \in L_2$.
- Enumerate the length of x_1
- If $|y| = n$, then $n+1$ calls to M_1 and $n+1$ calls to M_2.

Problem 22.5-7

Problem (22.5-7)

A directed graph $G = (V, E)$ is semiconnected if, for all pairs of vertices $u, v \in V$, we have $u \to v$ or $v \to u$.
Determine whether G is semiconnected.

Step one: Compute the component graph $G' = (V', E')$, in G':

- if $u \to v$ and $v \to u$, then $u = v$ (Antisymmetric)
- if $u \to v$ and $v \to w$, then $u \to w$ (Transitivity)
- $u \to v$ or $v \to u$ (Connexity)

This is a total order!

$$u_1 \to u_2 \to \cdots \to u_n$$

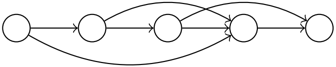
.

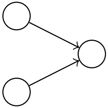
(Hamiltonian path)

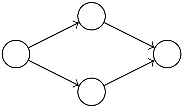
Problem 22.5-7

Problem

Given a direct acyclic graph (DAG) $G' = (V', E')$, check whether it contains a Hamiltonian path.







- G' is DAG, it contains at least one node with in-degree 0;
- if G' contains ≥ 2 nodes with in-degree 0, then reject;
- let the node with in-degree 0 be u;
- then u is the first node in the Hamiltonian path;
- Remove u from G', repeat.

Thank you!