Complexity Theory

Huan Long

Shanghai Jiao Tong University

Acknowledgements

Part of the slides comes from a similar course given by Prof. Yijia Chen.

http://basics.sjtu.edu.cn/~chen/

Textbook
Introduction to the theory of computation
Michael Sipser, MIT
Third edition, 2012

Outline

Complexity Theory

The Class P

The Class NP

Even when a problem is decidable, it might not be solvable in practice, since the optimal Turing machine which decides this problem could require astronomical time.

Time Complexity

Measuring Complexity

$A = \{0^k 1^k \mid k \geq 0\}$

M_1 on w:

1. Scan across the tape and reject if a 0 is found to the right of a 1.
2. Repeat if both 0s and 1s remain on the tape.
   - Scan across the tape, crossing off a single 0 and a single 1.
3. If 0s still remain after all the 1s have been crossed off, or if 1s still remain after all the 0s have been crossed off, reject. Otherwise if neither 0s or 1s remain on the tape, accept.

Time complexity of M_1

1. Analyze the running time of M_1 on every $x \in \Sigma^*$

$$f_1 : \Sigma^* \to \mathbb{N}.$$

2. Analyze the worst-case running time of M_1 on inputs of length $n \in \mathbb{N}$, $f_2 : \mathbb{N} \to \mathbb{N}$. In particular

$$f_2(n) = \max_{x \in \Sigma^n} f_1(x).$$

3. Analyze the average-case running time of M_1 on inputs of length $n \in \mathbb{N}$, $f_3 : \mathbb{N} \to \mathbb{N}$. In particular

$$f_3(n) = \frac{\sum_{x \in \Sigma^n} f_1(x)}{|\Sigma|^n}$$

Worst-case analysis

Definition

Let M be a deterministic Turing machine that halts on all inputs. The running time or time complexity of M is the function $f : \mathbb{N} \to \mathbb{N}$, where $f(n)$ is the maximum number of steps that M uses on any input of length n.

If $f(n)$ is the running time of M, we say that M runs in time $f(n)$ and that M is an $f(n)$ time Turing machine.

Customarily we use n to represent the length of the input.

Big-$\mathcal{O}$ Notation

Definition

Let $f, g : \mathbb{N} \to \mathbb{R}^+$ be two functions. Say that $f(n) = \mathcal{O}(g(n))$ if positive integers c and n_0 exist such that for every integer $n \geq n_0$

$$f(n) \leq c \cdot g(n).$$

When $f(n) = \mathcal{O}(g(n))$, we say that $g(n)$ is an upper bound for $f(n)$, or more precisely, that $g(n)$ is an asymptotic upper bound for $f(n)$, to emphasize that we are suppressing constant factors.

Examples

1. $5n^3 + 2n^2 + 22n + 6 = \mathcal{O}\left(n^3\right)$.
2. Let $b \geq 2$. Then
$$\log_b n = \frac{\log_2 n}{\log_2 b}$$
Hence, $\log_b n = \mathcal{O}\left(logn\right)$.
3. $3n \log_2 n + 5nlog_2log_2n + 2 = \mathcal{O}\left(n \log n\right)$.
4. $2^{10n^2+7n-6} = 2^{\mathcal{O}\left(n^2\right)}$.

n^c for $c > 0$ is a polynomial bound.

$2^{(n^\delta)}$ for $\delta > 0$ is an exponential bound.

Small o-notation

Definition

Let $f, g : \mathbb{N} \to \mathbb{R}^+$ be two functions. Say that $f(n) = o(g(n))$ if

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0.$$

In other words, $f(n) = o(g(n))$ means that for any real number $c > 0$, a number n_0 exists, where $f(n) < c \cdot g(n)$ for all $n \geq n_0$.

Examples

1. $\sqrt{n} = o(n)$.
2. $n = o(n \log \log n)$.
3. $n \log \log n = o(n \log n)$.
4. $n \log n = o(n^2)$.
5. $n^2 = o(n^3)$.

$$A = \{0^k 1^k \mid k \geq 0\}$$

M_1 on w:

1. Scan across the tape and reject if a 0 is found to the right of a 1.
2. Repeat if both 0s and 1s remain on the tape.
   - Scan across the tape, crossing off a single 0 and a single 1.
3. if 0s still remain after all the 1s have been crossed off, of if 1s still remain after all the 0s have been crossed off, reject.

   Otherwise, if neither 0s nor 1s remain on the tape, accept.

Time analysis

- The first stage scans the tape to verify the input is of the form 0^*1^*, taking n steps. Then the machine repositions the head at the left-hand end of the tape, again using n steps. In total $2n = \mathcal{O}(n)$ steps.
- In stages 2 and 3, the machine repeatedly scans the tape and crosses off a 0 and 1 on each scan. Each scan uses $\mathcal{O}(n)$ steps. Because each scan crosses off two symbols, at most $n/2$ scans can occur. So the total time taken by stage 2 and 3 is $(n/2)\mathcal{O}(n) = \mathcal{O}(n^2)$.
- In stage 4, the machine makes a single scan to decide whether to accept or reject, hence require time $\mathcal{O}(n)$.

The overall running time

$$\mathcal{O}(n) + \mathcal{O}(n^2) + \mathcal{O}(n) = \mathcal{O}(n^2).$$

Time classes

Definition

Let $t : \mathbb{N} \to \mathbb{R}^+$ be a function. Define the time complexity class

$$\mathsf{TIME}(t(n))$$

to be the collection of all languages that are decidable by an $\mathcal{O}\left(t(n)\right)$ time Turing machine.

Example

$\left\{0^k 1^k \mid k \geq 0\right\} \in \mathsf{TIME}(n^2)$.

A better algorithm

M_2 on w:

1. Scan across the tape and reject if a 0 is found to the right of a 1.
2. Repeat as long as some 0s and 1s remain on the tape.
   2.1 Scan across the tape, checking whether the total number of 0s and 1s remaining is even or odd. If it is odd, then reject.
   2.2 Scan again across the tape, crossing off every other 0 starting with the first 0, and then crossing off every other 1 starting with the first 1.
3. If no 0s and no 1s remain on the tape, then accept. Otherwise, reject.

Time analysis

1. Every stage takes $\mathcal{O}(n)$ time.
2. Stage 1 and 3 are executed once, hence total $\mathcal{O}(n)$ time.
3. Stage 2.2 crosses off at least half of the 0s and 1s each time it is executed, hence at most $1 + \log_2 n$ iterations.

Thus the total time of stages 2,3 and 4 is $(1 + \log_2 n)\mathcal{O}(n) = \mathcal{O}(n \log n)$.

The overall running time of M_2 is

$$\mathcal{O}(n) + \mathcal{O}(n \log n) = \mathcal{O}(n \log n).$$

Can we do even better than $\mathcal{O}(n \log n)$?

Theorem

Every language that can be decided in $o(n \log n)$ time on a single-tape Turing machine is regular.

$\{0^k 1^k \mid k \geq 0\}$ in linear time on a 2-tape TM

M_3 on w:

1. Scan across tape 1 and reject if a 0 is found to the right of 1.
2. Scan across the 0s on tape 1 until the first 1. At the same time copy the 0s onto tape 2.
3. Scan across the 1s on tape 1 until the end of the input. For each 1 read on tape 1, cross off a 0 on tape 2. If all 0s are crossed off before all the 1s are read, then reject.
4. If all the 0s have now been crossed off, then accept. If any 0s remain, then reject.
5. If no 0s and no 1s remain on the tape, then accept. Otherwise, reject.

Complexity relationships among models

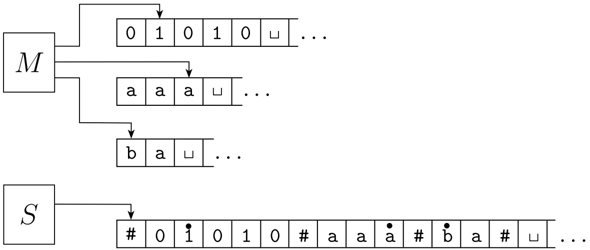
Theorem

Let $t(n)$ be a function with $t(n) \geq n$. The every $t(n)$ time multitape Turing machine has an equivalent $\mathcal{O}\left(t^2(n)\right)$ time single-tape Turing machine.

Proof (1)

We simulate an M with k tapes by a single-tape S.

- S uses $\#$ to separate the contents of the different tapes.
- S keeps track of the locations of the heads by writing a tape symbol with a dot above it to mark the place where the head on that tape would be.



Proof (2)

On input $w = w_1 \cdots w_n$;

1. First S puts its tape into the format that represents all k tapes of M:

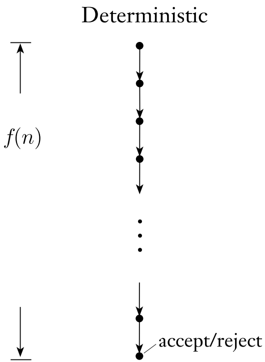
   $$\#\dot{w}_1 w_2 \cdots w_n \# \dot{\sqcup} \# \dot{\sqcup} \# \cdots \#.$$

   Time: $\mathcal{O}(n) = \mathcal{O}(t(n))$.
2. To determine the symbols under the virtual heads, S scans its tape from the first $\#$, which marks the left-hand end, to the $(k+1)$st $\#$, which marks the right-hand end. Time: $\mathcal{O}(t(n))$.
3. Then S makes a second pass to update the tapes according to the way that Ms transition function dictates. If S makes one of the virtual heads to the right onto a $\#$, then S writes $\sqcup$ on this tape cell and shifts the tape contents, from this cell until the rightmost $\#$, one unit to the right.
   Time: $\mathcal{O}(k \cdot t(n)) = \mathcal{O}(t(n))$.
4. Go back to 2.

Nondeterministic machines

Definition

Let N be a nondeterministic Turing machine that is a decider. The running time of N is the function $f : \mathbb{N} \to \mathbb{N}$, where $f(n)$ is the maximum number of steps that N uses on any branch of its computation on any input of length n.


Deterministic
Nondeterministic
$f(n)$
reject
$f(n)$
accept
accept/reject
reject

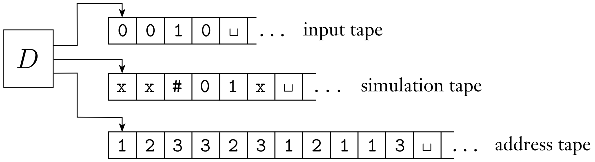
Theorem

Let $t(n)$ be a function with $t(n) \geq n$. The every $t(n)$ time nondeterministic single-tape Turing machine has an equivalent $2^{\mathcal{O}(t(n))}$ time deterministic single-tape Turing machine.

Proof (1)

We simulate a nondeterministic N by a deterministic D.

1. D try all possible branches of N's nondeterministic computation.
2. If D ever finds the accept state on one of these branches, it accepts.



Proof (2)

- On an input of length n, every branch of N's nondeterministic computation tree has a length of at most $t(n)$.
  Every node in the tree can have at most b children, where b is the maximum number of legal choices given by N's transition function. Thus, the total number of leaves in the tree is at most $b^{t(n)}$.
- The total number of the nodes in the tree is less than twice the maximum number of leaves, hence $\mathcal{O}\left(b^{t(n)}\right)$. The time it takes to start from the root and travel down to a node is $\mathcal{O}(t(n))$. Hence the total running time of D is $\mathcal{O}\left(t(n)b^{t(n)}\right) = 2^{\mathcal{O}(t(n))}$.
- D has 3 tapes, thus can be simulated by a single-tape TM in time
$$\left(2^{\mathcal{O}(t(n))}\right)^2 = 2^{\mathcal{O}(t(n))}.$$

The Class P

Definition

P is the class of languages that are decidable in polynomial time on a deterministic single-tape Turing machine. In other words:

$$\mathsf{P} = \bigcup_{k \in \mathbb{N}} \mathsf{TIME}\left(n^k\right).$$

1. P is invariant for all models of computation that are polynomially equivalent to the deterministic single-tape machine.
2. P roughly corresponds to the class of problems that are realistically solvable on a computer.

Examples of problems in P

Reasonable encodings

- We continue to use $\langle \cdot \rangle$ to indicate a reasonable encoding of one or more objects into a string.
- Unary encoding of n as $\underbrace{11 \cdots 11}_{n \text{ times}}$ is exponentially larger than the standard binary encoding of n, hence not reasonable.
- A graphs can be encoded either by listing its nodes and edges, i.e., its adjacency list, or its adjacency matrix, where the (i, j)th entry is 1 if there is an edge from node i to node j and 0 i if not.

The path problem

$\textit{PATH} = \{\langle G, s, t\rangle \mid G$ is a directed graph that has a directed path from s and $t\}$.

Theorem

PATH $\in$ *P*.

Testing relative prime

$$\textit{RELPRIMIE} = \{\langle x, y \rangle \mid x \text{ and } y \text{ are relatively prime}\}.$$

Theorem

RELPRIMIE $\in$ *P*.

The Euclidean Algorithm

Recall the greatest common divisor $\text{gcd}(x, y)$ is the largest integer that divides both x and y.

E on $\langle x, y \rangle$:

1. Repeat until $y = 0$:
2. Assign $x \leftarrow x \pmod{y}$.
3. Exchange x and y.
4. Output x.

R on $\langle x, y \rangle$:

1. Run E on $\langle x, y \rangle$.
2. If the result is 1, then accept. Otherwise, reject.

Time analysis

We show that E runs in polynomial time

1. Every execution of stage 2 with $y \leq x$ cuts the value x at least by half.
2. Thus, the maximum number of times that stage 2 and 3 are executed is the lesser of $2 \log_2 x$ and $2 \log_2 y$.

Testing context-freeness

Theorem

Every context-free language is a member of P.

Recall (1)

Definition

A context-free grammar is in Chomsky normal form if every rule is of the form

$$A \to BC \text{ and } A \to a$$

where a is any terminal and A, B and C are any variables, except that B and C may be not the start variable. In addition, we permit the rule $S \to \epsilon$, where S is the start variable.

Theorem

Any context-free language is generated by a context-free grammar in Chomsky normal form.

Theorem

Let G be CFG in Chomsky normal form, and G generates w with $w \neq \epsilon$. Then any derivation of w has $2|w| - 1$ steps.

Recall (2)

S on $\langle G, w \rangle$

1. Convert G to an equivalent grammar in Chomsky normal form.
2. List all derivations with $2|w| - 1$ steps; except if $|w| = 0$, then instead check whether there is a rule $S \rightarrow \epsilon$.
3. If any of these derivations generates w, then accept; otherwise reject.

The running time of S is $2^{\mathcal{O}(n)}$.

Dynamic programming

Let w be an input string and $n := |w|$.
For every $i \leq j \leq n$ we will compute

table(i, j) = the collection of variables that can generate the substring $w_i w_{i+1} \dots w_j$.

Dynamic Programming (cont'd)

D on $w = w_1 \cdots w_n$:

1. For $w = \epsilon$, if $S \to \epsilon$ is a rule, then accept; else reject.
2. For $i = 1$ to n:
3. For each variable A:
4. Test whether $A \to b$ is a rule, where $b = w_i$.
5. If so, place A in table(i, i).
6. For $\ell = 2$ to n:
7. For $i = 1$ to $n - \ell + 1$:
8. Let $j = i + \ell - 1$
9. For $k = i$ to $j - 1$:
10. For each rule $A \to BC$:
11. If $B \in$table(i, k) and $C \in$table$(k + 1, j)$, then put A in table(i, j).
12. If $S \in$table$(1, n)$, then accept; else reject.

The Class NP

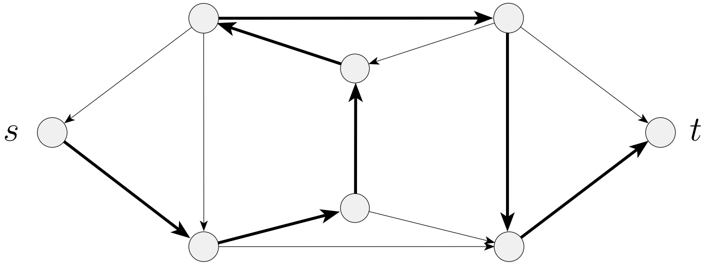
Hamiltonian path

Definition

A Hamiltonian path in a directed graph G is a directed path that goes through each node exactly once.

$$\textit{HAMPATH} = \{\langle G, s, t\rangle \mid G \text{ is a directed graph With a Hamiltonian path from } s \text{ and } t\}$$

Hamiltonian path (cont'd)



Polynomial verifiability

Even though we don't know how to determine fast whether a graph contains Hamiltonian path, if such a path were discovered somehow (perhaps using the exponential time algorithm), we could easily convince someone else of its existence simply by presenting it.

In other words, verifying the existence of a Hamiltonian path may be much easier than determining its existence.

Testing composite

Definition

A natural number is composite if it is the product of two integers > 1.

$$\text{COMPOSITES} = \{x \mid x = pq \text{ for integers } p, q > 1\}.$$

Verifiers

Definition

A verifier for a language A is an algorithm V, where

$$A = \{w \mid V \text{ accepts } \langle w, c \rangle \text{ for some string } c\}.$$

We measure the time of a verifier only in terms of the length of w, so a polynomial time verifier runs in polynomial time in the length of w. A language A is polynomial verifiable if it has a polynomial time verifier.

The string c in the above definition is a certificate, or proof, of membership in A. For polynomial verifiers, the certificate has polynomial length (in the length of w).

Certificates

For *HAMPATH*, a certificate for $\langle G, s, t \rangle \in$ HAMPATH is a Hamiltonian path from s to t.

For *COMPOSITES*, a certificate for x is one of its divisors.

The class NP

Definition

NP is the class of languages that have polynomial time verifiers.

Nondeterministic polynomial Turing machines

Theorem

A language is in NP if and only if it is decided by some nondeterministic polynomial time Turing machines.

Proof (1)

Assume that the verifier V is a TM that runs in time n^k.
N on w with $n = |w|$

1. Nondeterministically select string c of length at most n^k.
2. Run V on $\langle w, c \rangle$.
3. If V accepts, then accept; otherwise, reject.

Proof (2)

Assume that A id decided by a polynomial time NTM N.
V on $\langle w, c \rangle$

1. Simulate N on input w, treating each symbol of c as a description of the nondeterministic choice to make at each step.
2. If this branch of N's computation accepts, then accept; otherwise, reject.

Nondeterministic time complexity classes

Definition

NTIME(t(n)) $= \{L \mid L$ is a language decided by an $\mathcal{O}(t(n))$ time nondeterministic Turing machine $\}$.

Corollary

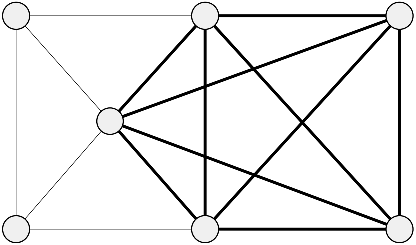
$$NP = \bigcup_{k \in \mathbb{N}} NTIME\left(n^k\right).$$

Examples of problems in NP

The clique problem

Definition

A clique in an undirected graph is a subgraph, wherein every two nodes are connected by an edge. A k-clique is a clique that contains k nodes.



A graph with a 5-clique.

The clique problem (cont'd)

CLIQUE=$\{\langle G, k\rangle \mid G$ is an undirected graph with a k clique$\}$.

Theorem

CLIQUE is in NP.

Proof (1)

V on $\langle\langle G, k\rangle, c\rangle$:

1. Test whether c is a subgraph with k nodes in G.
2. Test whether G contains all edges connecting nodes in c.
3. If both pass, then accept; otherwise reject.

Proof (2)

N on $\langle G, k \rangle$:

1. Nondeterministically select a subset c of k nodes in G,
2. Test whether G contains all edges connecting nodes in c.
3. If yes, then accept; otherwise reject.

The subset-sum problem

$$\textit{SUBSET-SUM} = \{\langle S, t\rangle \mid S = \{x_1, \cdots x_k\} \text{ and for some } \{y_1, \cdots, y_\ell\} \subseteq S, \text{ we have } \sum_{i\in[\ell]} y_i = t\}.$$

Theorem

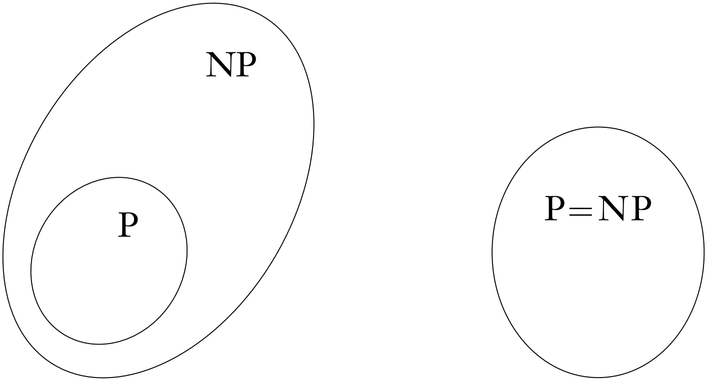
SUBSET-SUM is in NP.

The P versus NP question

P = the class of languages for which membership can be decided quickly.

NP = the class of language for which membership can be verified quickly.

Two possibilities



S. Smale. P versus NP, a gift to mathematics from computer science.