Time Complexity

investigation of time, memory or other resources required for solving computational problems.

- Time

Objective → basics of time complexity theory

→ way of measuring the time used to solve a problem

→ classify problems according to the amount of time required.

→ possibility that certain decidable problems require enormous amounts of time, how to determine those?

Measuring complexity

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

$\downarrow$ decidable

How much time does it take to decide A?

Terminology

Number of steps that an algo uses may depend on:

1) input type

graph $\longrightarrow$ no of nodes, edges, max degree etc.

For simplicity:

Compute running time as a function of length of the input string

don't consider any other factors

Worst-case analysis:

→ longest running time on all inputs of a particular length.

Average-case analysis

→ ⟨ running times —"— —"— ⟩

Running Time / Time Complexity

Let M be a deterministic Turing Machine that halts on all inputs. The running time or time complexity of M is the function

$$f : \mathbb{N} \rightarrow \mathbb{N}$$

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 TM.

O- and o-notation

→ exact running time is often a complex expression

best bound on worst case

↓

estimate

↓

asymptotic analysis.

↳ understand the running time on large inputs

consider only the highest order term of an expression highest order term = term that $\underline{\text{grows fastest}}$

roughly $\left\{ \begin{array}{l} f(x) \text{ faster than } g(x) \\ \text{if } \displaystyle\lim_{x \to \infty} \frac{f(x)}{g(x)} = \infty \end{array} \right.$

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

$f(n) = O(g(n))$ if positive integers c and n_0 exist such that for every integer $n \geq n_0$

$$f(n) \leq c\, g(n)$$

$f(n) = O(g(n)) \longrightarrow$ g is an asymptotic upper bound for f

constant factors are suppressed

→ examples

$$f(n) = 2^{O(\log n)}$$

$$n = 2^{\log_2 n}$$

$$n^c = 2^{c \log_2 n}$$

$$\therefore \quad 2^{O(\log n)} = O(n^c) \text{ for some } c$$

$$= n^{O(1)}$$

Bounds of the form $n^{c > 0}$

→ polynomial

Bounds of the form $2^{(n^{\delta})}$ $\delta \in \mathbb{R}^{+}$

→ exponential

o - notation

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

$$f(n) = o(g(n))$$

$f(n)$ is never $o(f(n))$

Analysing algorithms

Let's analyze the TM algorithm we gave for the language $A = \{0^k 1^k | k \geq 0\}$. We repeat the algorithm here for convenience.

M_1 = "On input string 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:
3. Scan across the tape, crossing off a single 0 and a single 1.
4. 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 nor 1s remain on the tape, *accept*."

Step 1 : check if input of the form $0^* 1^*$

$\downarrow$

$\underbrace{n}_{\text{length of input}}$ steps

reposition head to left $\leftarrow$ n steps

Step 1 : 2n steps

or O(n) steps

2 & 3 : at most $\frac{n}{2}$ scans

$O(n^2)$ each scan O(n) steps

(4) $\rightarrow$ $O(n)$

$$O(n) + O(n^2) + O(n) = O(n^2)$$

Let $t : \mathbb{N} \rightarrow \mathbb{R}^+$ be a function.
Time complexity class TIME ($t(n)$),
collection of all languages that are decidable by an $O(t(n))$ time turing machine.

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

$$A \in \text{TIME}(n^2)$$

Is there a machine that decides
A asymptotically more quickly ?

$\equiv$

is A in TIME$(t(n))$
for $t(n) = o(n)$?

Yes. $O(n \log n)$ algo exists (see book)
$O(n)$ also exist

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

We can decide the language A in
$O(n)$ time if the TM has
a second tape

single - tape $\rightsquigarrow O(n^2)$

$\rightsquigarrow O(n \log n)$

no single tape machine can do it more quickly

2 - tape $\rightsquigarrow O(n)$

Computability Theory

↓

all reasonable models of computation are equivalent

Complexity theory

↓

which model of computation you choose affects the time complexil

languages decidable in linear time in one

may not necessarily be linear in other

With which model do we measure time?

Time requirements don't differ greatly for typical deterministic models.

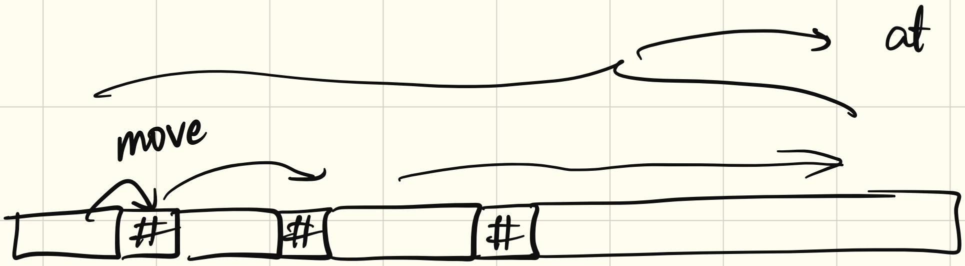
Complexity relationship among models

Theorem 7.8

Every $t(n)$ time multitape turing machine has an equivalent of $O(t^2(n))$ time single tape TM.

Proof Idea $\longrightarrow$ use equivalence of multitape and single tape TM.

simulating each step of multitape TM uses at most $O(t(n))$ steps on the single-tape machine.



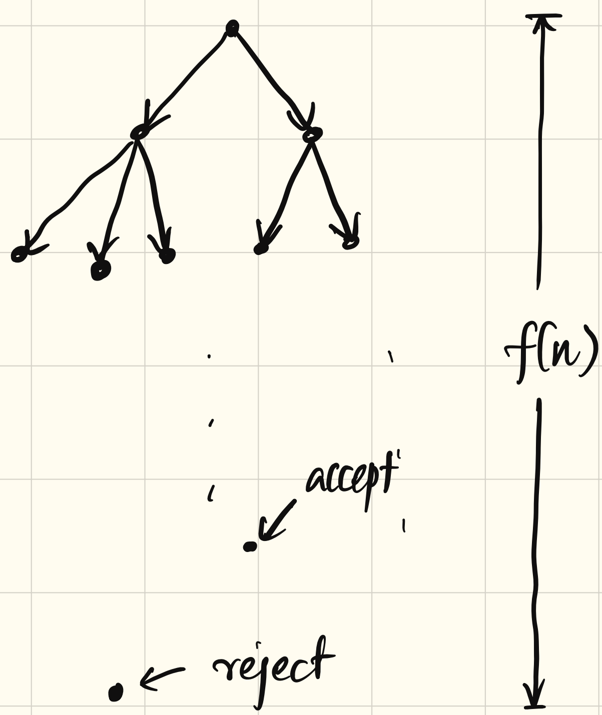
at most $t(n)$ moves per step of multi-tape TM

Non-deterministic TM is a decider if all its computation branches halt on all inputs.

Let N be a non-deterministic 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 of steps that N uses in any branch of its computation on any input of length n,

Deterministic

Non-deterministic



Theorem 7.11

Let $t(n)$ be a function where $t(n) \geq n$.
Then every $t(n)$ time Non-det. single tape TM has an equivalent $2^{O(t(n))}$ time deterministic TM.

Similar proof... $\longrightarrow$ simulate N on det TM M

Every node in the tree can have at most b children, b = max no. of legal choices given by N's δ.

$\therefore$ Total no. of leaves $\leq b^{t(n)}$

Total no. of nodes $<$ 2 $\times$ max no. of leaves

$$= O\left(b^{t(n)}\right)$$

Time taken to travel down a node $= t(n)$

$\therefore$ running time $= O\left(t(n) \cdot b^{t(n)}\right)$

$$= 2^{O(t(n))}$$

Three tapes in simulation

$\therefore$ single tape $= \left(2^{O(t(n))}\right)^2$

$$= 2^{2O(t(n))}$$

$$= 2^{O(t(n))}$$

Class P

Thm 7.8, 7.11

single tape vs multi tape

atmost polynomial difference

deterministic vs non-det

atmost exponential difference

Polynomial Time

polynomial time differences in running time $\rightarrow$ considered to be small

exponential time differences in running time $\rightarrow$ considered to be large

Why?

n^3 vs 2^n

$n = 1000$

$n^3 = 1$ billion

$2^{1000} \rightsquigarrow$ much larger than the number of atoms in the universe

exponential time algorithms $\rightarrow$ rarely useful

$\rightsquigarrow$ typically arise when we solve problems by exhaustively searching through a space of solutions $\rightarrow$ brute force

can be avoided sometimes by a deeper understanding of the problem

All reasonable deterministic models of computation are <u>polynomially equivalent</u>.

any one can simulate other with a poly-↑ in run time

- Why disregarding polynomial diff is not absurd (here).

P is the class of languages that are decidable in polynomial time on a deterministic single-tape TM.

$$P = \bigcup_{k} \text{TIME}(n^k)$$

① P is invariant for all models of computation that are polynomially equivalent

② P roughly corresponds to the class of problems that are realistically solvable on a computer.
→ relevant from a practical standpoint.

Examples

→ proving that an algorithm is polynomial in run time

① Give a polynomial upper bound { O( ) } on the no. of stages it uses when it runs on an input of length n.

② Examine individual stages in the description of the algorithm to be sure that each can be implemented in

poly time
on a reasonable
model.

Encoding $\langle \cdot \rangle$

→ reasonable method of encoding:

→ one that allows for
poly-time encoding
and decoding of
objects into natural
internal representations

Unary notation for encoding
numbers 17 11111111111111111

not reasonable
because it is exponentially
larger than truely reasonable
encoding (base-k $k \geq 2$)

PATH problem: determine whether a directed path exists from nodes s to t in a directed graph G.

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

PATH $\in$ P

① Brute-force algorithm

examine all potential paths in G and determine whether any is a directed path from

Potential path $\rightsquigarrow$ sequence of length atmost m

$\rightsquigarrow$ no. of vertices in G.

m^m potential paths

$$\sum_{i=1}^{m} m^i \approx m^m$$

One way $\longrightarrow$ use BFS

→ successively mark all nodes in G that are reachable from S by directed paths of length 1, then 2, then 3, through m.

Detailed

— Algo and proof —

Book

2) Relatively prime

1 is the largest integer that divides both.

RELPRIME $:= \{ \langle x, y \rangle \mid x$ and y are relatively prime$\}$

Brute-force:

magnitude $\rightsquigarrow$ exponential in the length of representation

search through all possible divisors of both numbers, accept if none are greater than 1

$\rightarrow$ not Poly-time

exponential checks

Solution: Euclidean algorithm

$\gcd(x, y)$ is the largest integers that evenly divides both x and y

x and y are relatively prime $\iff$ $\gcd(x, y) = 1$

RELPRIME $\in$ P

Proof:

E = "On input $\langle x, y \rangle$, where x and y are numbers in binary:

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

$R :=$ " On input $\langle x, y \rangle$, where x and y are natural numbers in binary :

1. Run E on $\langle x, y \rangle$.
2. If the result is 1, accept. O/w reject. "

Stage 2 : $x \leftarrow x \bmod y$

In each subsequent execution of stage 2, x is at least halved

$x < y$

x and exchanged

$y < x$

$\frac{x}{2} \geq y \Longrightarrow x \bmod y < y \leq \frac{x}{2}$

$y > \frac{x}{2} \Rightarrow x \bmod y = x - y < \frac{x}{2}$

Each of the original values of
x and y are reduced by
atleast half every other time

maximum no. of times 2 & 3 executed $= \min\{2\log_2 x,\ 2\log_2 y\}$

$\propto$ to length of representation

$O(n)$

<u>Every context-free language is a member of P.</u>

Thm 4.9 $\rightsquigarrow$ every CFL is decidable

gave an algorithm for each CFL that decides it.

Let L be a CFL generated by a CFG G that is in Chomsky Normal Form.

Claim: if G is a CFG in Chomsky Normal Form, then for any string $w \in L(G)$ of length $n \geq 1$, exactly $2n-1$ steps are required for any derivation of w.

Chomsky Normal Form

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

$$A \rightarrow BC$$

CFG:

4-tuple

(V, Σ, R, S)

terminals

variables

set of rules

$S \in V$ start variable

$L(G) = \{ w \in \Sigma^* \mid S \overset{*}{\Rightarrow} w \}$

$\overset{*}{\Rightarrow}$: derives

u derives v if $u = v$ or $\exists$ a sequence $u_1, \dots, u_k$ $k \geq 0$

$u \Rightarrow u_1 \Rightarrow u_2 \cdots \Rightarrow u_k \Rightarrow v$

$\Rightarrow$: yields

$A \to a$

Where a is any terminal and
A, B and C are any variables,
B and C may not be
the start variable.

$S \to \varepsilon$ ✓ S = start variable

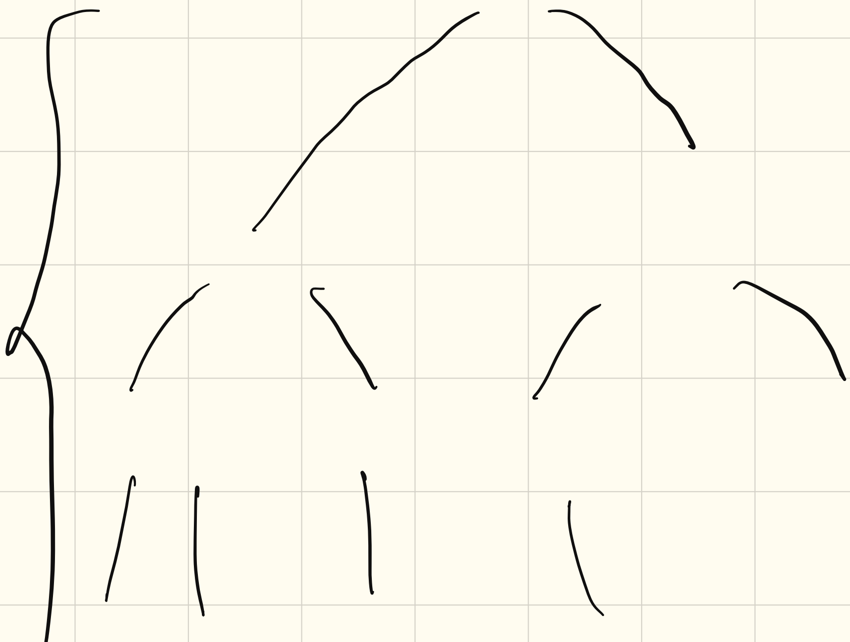
$w \in L(G)$ of length n

Σ^{2n-1} checks

Use DP
Consider the subproblems of determining
whether each variable in G
generates each substring in w

Algorithm enters the solution to this subproblem in an $n \times n$ table. For $i \leq j$, the $(i,j)^{th}$ entry of the table contains the collection of variables that generate the substring $w_i\, w_{i+1} \cdots w_j$.

Taught in class.



String

The class NP

Certain other problems

↓

Attempts to avoid brute force haven't been successful.

Why?

We don't know

We don't yet have a deeper understanding of the problem?

Undiscovered yet?

Do not exist intrinsically difficult

* Complexity of many problems are linked. A polynomial time algorithm for one problem → can solve an entire class of problems.

HAMPATH = { <G, s, t> | G is a directed graph with a hamiltonian path from s to t }

Modify PATH soln ⟶ add a check to verify that the path is hamiltonian

Solvable in polynomial time?

OK

Verifiable in polynomial time? Yes

* Polynomial verifiability

A <u>verifier</u> for a language A is an algorithm V, where

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

Time of verifier → in length of w,
polynomial time verifier runs in polynomial time in the length of w.
A language A is polynomially verifiable if it has a polynomial time verifier.

$c \equiv$ certificate, proof of membership in A.

↓

has to have a length polynomial in w

↓

$\because$ verifier runs in polynomial time (in w)

Example:

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

COMPOSITES → certificate = one divisor

check in poly-time that the input is in the language when it is given the certificate

NP → class of languages that have polynomial time verifiers.

↓

contains many problems of practical importance.

Non-deterministic polynomial time

HAMPATH using NDTM:

N_1 = " On input $\langle G, s, t \rangle$, where G is a directed graph with nodes s and t:

non-det polynomial ← 1. Write a list of m numbers $p_1, p_2, \ldots, p_m$, where m is the number of nodes in G. Each number in the list is non-deterministically selected to be b/w 1 and m.

polynomial
2. Check for repetitions → reject
3. Check whether $s = p_1$ and $t = p_m$. If either fail, reject.

polynomial
4. For each i b/w 1 and $m-1$, check whether (p_i, p_{i+1})

is an edge of G. If any are not, reject → all test pass ⇓ accept

---

"What is the difference b/w parallel computing and non-determinism"?

A language is in NP ⟺ it is decided by some non-deterministic polynomial time TM.

Proof idea: convert a polynomial time verifier to an equivalent polynomial time NTM and vice versa

NTM simulates verifier → guess certificate

verifier simulates NTM → use accepting branch as the certificate.

PROOF : $\Longrightarrow$

Let $A \in NP$

Let V be the poly-time verifier for A that exists by the def^n of NP.

Assume that V is a TM that runs in time n^k and construct N as follows

N = "On input w of length n :

1. Non-deterministically select string c of length at most n^k.
2. Run V on input $\langle w, c \rangle$
3. If V accepts, accept; o/w reject"

$\Longleftarrow$

Let A be decided by a NTM N and construct a poly-time verifier as follows:

V = " On input $\langle w, c \rangle$

1. Simulate N on input w, treating each symbol of c as a description of the non-deterministic choice to make at each step.

2. If this branch of N's computation accepts; accept o/w reject.

$\text{NTIME}(t(n)) = \{ L \mid L$ is a language decided by an $O(t(n))$ time non-deterministic TM $\}$

$$NP = \bigcup_k \text{NTIME}(n^k)$$

<u>Examples</u>

→ clique and k-clique