# Lecture 19 :

notes are on the webpage

Hw 4 : tomorrow at 11 am

Today: complexity theory ) 7.1 : ignore 7.16 & 7.2

* We saw that certain languages are:

undecidable : even with as much time and memory (↳ space) as you want you can't solve that problem.

Even if a problem is decidable, it may take an enormous amount of time/space to solve.

↳ we can solve the problem but any

algorithm is not usable in practice.

Complexity theory: the study of important problems and how much ressources you need to solve them

↳ time, memory, communication, samples, entanglement

Some examples: $\begin{bmatrix} 1 & 4 \\ 5 & 6 \end{bmatrix} * \begin{bmatrix} -1 & 7 \\ 3 & 11 \end{bmatrix} = ?$

(1) Matrix multiplication:

given 2 $n \times n$ matrices $M_1, M_2$

How much time to multiply them?

↳ $(+, *, \%, -)$

Super important problem. A "bottle neck" in a lot of areas: example ML

it'd be nice to have fast algos for that...

$$M_1 = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \qquad M_2 = \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{bmatrix}$$

Entry (1,1) of $M_1 \times M_2$ :

$$a_{11} \times b_{11} + a_{12} \times b_{21} + a_{13} \times b_{31}$$

If $M_1, M_2$ are $n \times n$ => need $O(n)$ operations to compute $M_{ij}$.

Runtime of obvious algorithm: $n^2 \cdot O(n)$

($n^2$ entries; each entry)

$O(n^3)$

Q: can we do better?

Yes! Strassen $O(n^{2.7})$

$O(n^{2.37\,1339})$ ← Josh Alman

Open Q: $O(n^2)$

(2) Prime: given a number x in binary,

is x prime?

→ central to cryptography

(3) Factoring: given x in binary,

output the prime factorization of x.

132 ⇒ 2 * 2 * 3 * 1

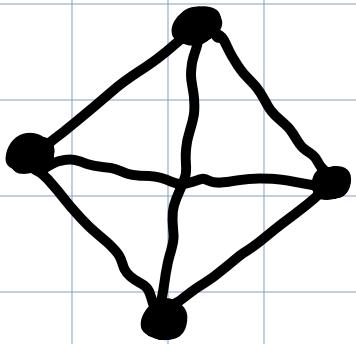
After decades: a fast algorithm was found for PRIME!

For factoring: a fast quantum algorithm,

still unknown for "TM algorithms"

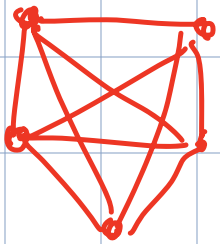
(4) Clique : given a graph $G$ on $n$ vertices, and a number $k$

does $G$ contain a clique of size $k$?



← a clique of size 4

(all vertices are connected to each other!)



# Time Complexity:

A step of a TM is a single transition of the TM on an input.

The time complexity / running time of a TM $M$ is a function $f: \mathbb{N} \to \mathbb{N}$ such that

$$f(n) = \max_{w, |w|=n} \#\text{ of steps taken by } M \text{ on input } w \text{ before halting}$$

$\max_{w,|w|=n}$ → worst-case

To analyze running time: use O

M: on input $w$ ($|w| = n$)

start: $w_1$ ... end: $w_n$

O(n) (1) Scan the tape until we see a 0 or a 1

O(1) (2) If none found -> halt + accept

O(n) (3) If one found -> cross it & continue scanning until you find a matching 0/1

O(1) (4) If none found: reject

O(n) (5) Else cross off that 0/1,
go back to the begining of the tape

(6) go back to line 1

↓ $w_1$ ↓ start $w_n$

lines 1-5 take $O(n) + O(1) + \cdots + O(n)$

$O(n)$ time

lines 1-5 are executed $O(n)$ times

↳ each time cross 2 symbols of $w$

so worst case : execute $\frac{n}{2}$ times

So: $O(n^2)$ running time

---

Defn: Let $t: \mathbb{N} \to \mathbb{N}$ be a function

Time $(t(n)) = \{L \mid L$ is decided by a TM with $O(t(n))$ running time $\}$

(-> M always halts)

(1-tape)

Recall that $t: \mathbb{N} \to \mathbb{N}$ is Polynomial

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

alternatively $t(n) = O(n^k)$ for some $k \geq 0$

$$P = \{L \mid L \text{ is a language decided by a (1-tape) TM running in polynomial time}\}$$

$$= \bigcup_{k=1}^{\infty} \text{Time}(n^k)$$

We think of P as the problems that have a relatively efficient algorithm

Q: Why polynomial time? why not $O(n)$ or $O(n^2)$

$n^{100}$, most poly time algos run in time $O(n)$, $O(n^2)$ or $O(n^3)$

Even if a problem is in P, people still try to make it trully fast

Q2: TMs are super slow. Why don't we define P for a "better" model?

More realistic models like multitape TMs or RAM model

↳ a 1 tape TM can simulate these with only a polynomial blow-up in running time

$n^3$ - 2 tapes $\Rightarrow$ $n^6$ - 1 tape

Exceptions: quantum algorithms

nondeterministic TMs

$k$ states NFA $\Rightarrow$ $2^k$ states DFA

$f(n)$ NTM $\rightarrow$ $2^{f(n)}$ DTM

Q3: why worst case running time?

Just because a problem is hard on some inputs, doesn't mean it's hard on typical inputs.

* Maybe we can approximate the solution

Worst case complexity is a start.

## Some problems in P

$PATH = \{\langle G, s, t\rangle \mid G$ is a directed graph, there's a directed path from $s$ to $t\}$

$n$ vertices

$m$ edges : $m \leq n^2$

$|\langle G\rangle| \leq O(n^2)$

Naive algorithm: look at all possible paths in $G$ => check if one of them is a directed

path from s to t.

↳ # of ways to order n vertices is $n!$ ⇐ not poly(n)

Better: On input $\langle G, s, t \rangle$

~~$O(n+m)$~~ → (1) Perform BFS starting from s

$poly(n,m) = poly(n)$

$O(1)$ → (2) If t is reached : accept

→ (3) Else : reject

Correctness:

(1) Algorithm runs in time poly(n) and

$|\langle G, s, t \rangle| = O(n^2)$. So this is poly time.

(2) If $\langle G, s, t \rangle \in$ Path, then there's a path from s to t, so in BFS we reach t & algorithm accepts

(3) If $\langle G, s, t \rangle \notin$ Path, then there's no path from s to t, so in BFS we can't reach t, so algorithm rejects

So algo decides PATH in poly time $\Rightarrow$ PATH $\in$ C

PRIME = $\{\langle x \rangle \mid x$ is a number in binary, $x$ is prime $\}$

Algo: On input $\langle x \rangle$:

$O(x)$ | For $i = 2$ to $x - 1$

$O(1)$
$O(|\langle x \rangle|)$
$poly(|\langle x \rangle|)$

→ check if $i$ divides $x$

if yes reject $\langle x \rangle$.

Accept $\langle x \rangle$.

input: $\log x$ length

running time is $x$

Running time is $O(x)$

$2^{\log x} = x$

$|\langle x \rangle| = O(\log x)$ ← input size

$O(x)$ is exponential in $O(\log x)$

So this not polytime! It runs in time exponential in the input size!

$\log x = n$, then $O(x) = O(2^n)$

If you get $\langle x \rangle$ ⇒ poly time means

$|\langle x \rangle| = \log(x)$

$\text{poly}\left(\frac{\log x}{\text{input size}}\right)$

poly(k) is bad!

$\overbrace{1111}^{4}$ $\Rightarrow$ $1^k$ to represent $k$