Machine Model:

Random Access Machine (RAM)

Sequential machine.

Takes unit time to read word from memory,

unit time for basic arithmetic operations (+, −, ×, ÷)

There are many other machine models

Parallel RAM.

Memory Hierarchy (Fast Cache, Multiple levels?)

Massively Parallel Computation

We may study some in HWs, but most of focus on just basic model

(unit-cost RAM).

Want Algorithms that use less resources

- faster (less time) ← most of course
- small space
- other notions of goodness (communication, rounds of parallelism, etc).

But most importantly will emphasize algorithms that

- $\boxed{\text{give new ideas / perspectives into solutions}}$.

Asymptotic Notation (Big-O, Omega, Little-o, etc).

Big-O

• $T(n) \in O(f(n))$ if $\exists c, n_0 \geq 0$ st. $T(n) \leq c \cdot f(n)$ $\forall n \geq n_0$.

↑ sometimes say $T(n) = O(f(n))$, pls do not let this confuse you.

Technically $O(f(n))$ is a set $= \{T(n) : \exists c, n_0$ etc...$\}$.

Loosely, $T(n)$ grows no faster than some constant factor times $f(n)$.

Big-Omega

• $T(n) \in \Omega(f(n))$ if $\exists c, n_0 \geq 0$ st $T(n) \geq c \cdot f(n)$ $\forall n \geq n_0$.

Omega → $T(n)$ grows no slower than. $f(n)$ times constant.

Eg: $3n^2+17 \in O(n^2)$, also $\in O(n^3)$. but ~~not~~ $\notin O(n^{1.9})$

$3n^2+17 \in \Omega(n^2)$ but not $\Omega(n^3)$. though $\in \Omega(n^{1.9})$

Theta: ~~Θ~~

• $T(n) \in \Theta(f(n))$ if $T(n) \in O(f(n))$ and $\in \Omega(f(n))$.

So the growth rates of $T(n)$ and $f(n)$ are similar upto constants.

$\Rightarrow$ ~~$\nexists$~~ $3n^2+17 \in \Theta(n^2)$ but not the others.

Little-o : $T(n) \in o(f(n))$ if $\boxed{\forall c>0}$ $\exists n_0 \geq 0$ st. $T(n) \leq c f(n)$.

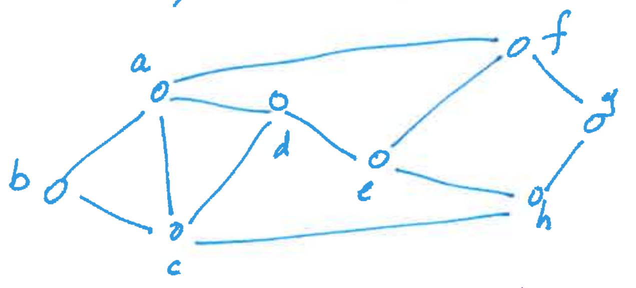
So $T(n)$ grows slower than any constant times $f(n)$.

$3n^2+17 \in o(n^{2.1})$ or even $o(n^2 \log\log n)$ etc.

Algorithmic Question :

(Q) Given graph $G = (V, E)$, how many triangles does it contain?

^ undirected simple



← 2 triangles abc, adc

Suppose $|V| = n$
$|E| = m \leq \binom{n}{2} = O(n^2)$

↑ arises in social networks research, measure of how connected the network is. see "transitivity" & "clustering coeffs".

How fast can we find the answer?

How is graph given? Say adjacency matrix $A_{ij} = \begin{cases} 1 & \text{if } (i,j) \in E \\ 0 & \text{if not} \end{cases}$

↑ $n \times n$ matrix.

Ans 1 : run over all triples u, v, w see if $A_{uv}, A_{vw}, A_{uw} = 1$

time $= O(n^3)$.

Ans 2 : for each edge (u,v) and each other vertex w

check if $A_{uw} = A_{vw} = 1$

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

↑ usually assume that $m \geq n-1$ since most interesting graphs are connected, but being careful today.

(Non-trivial) Ans 3 :

If you can ~~multiply~~ compute $\underset{\text{matrix multiplication}}{A \cdot A}$ in $O(n^c)$ time

$\Rightarrow$ can count # of triangles in $O(n^c)$ time too.

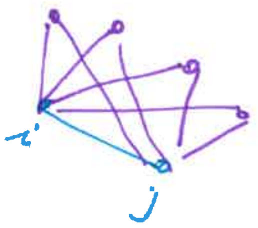
(note: $c \geq 2$ since we need to look at all of A, takes n^2 time).

Proof: $A^2_{ij} = \sum_{k=1}^{n} A_{ik} A_{kj}$ = # of vertices k st both i & j have edge to k.

= # 2-hop paths from i to j

So # triangles formed by edge $(i,j) \in E$

$= A^2_{ij}$



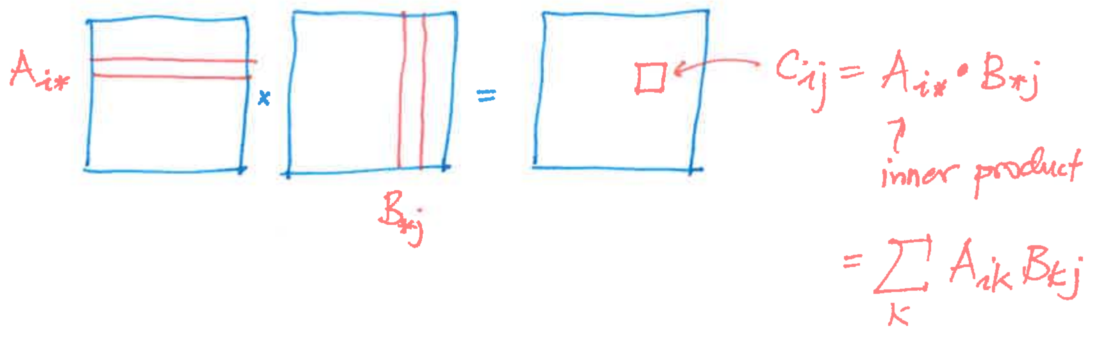
Answer $= \sum_{(i,j) \in E} A^2_{ij}$.

time $= O(n^c)$ for matrix multiply
$+ O(n^2)$ to compute this sum $= O(n^c)$.

Great! So how fast can we compute the product of two matrices

(Actually, just square of A, product of A with itself.
But we don't know ways to make this faster.).

How fast can we compute $A \cdot B = C$ ($n \times n$ matrices).



Ans 1: $O(n^3)$ — takes $O(n)$ for each entry C_{ij} and n^2 entries $\Rightarrow O(n^3)$.

In fact, definition itself gives the algorithm.
(often the case ~~e.g. shortest path~~
but this algorithm may be misleading, may be suboptimal)

Here's a different idea: Divide & Conquer

Suppose n is even:—

write $A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}$ $B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}$ each A_{ij} B_{ij} submatrix

then $AB = \begin{pmatrix} A_{11}B_{11} + A_{12}B_{21}, & A_{11}B_{12} + A_{12}B_{22} \\ A_{21}B_{11} + A_{22}B_{21} & A_{21}B_{12} + A_{22}B_{22} \end{pmatrix}$

(Check!)

So if we can recursively compute the terms. (eight of them)

$A_{11}B_{11}$, $A_{12}B_{21}$ . . . . . . $A_{22}B_{22}$

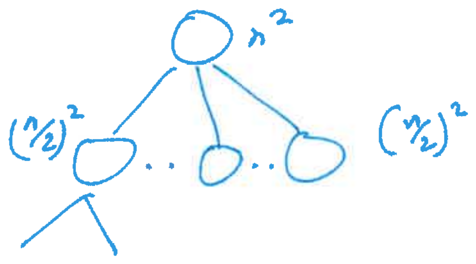
Can sum them up and get answer.

Runtime? ~~Base case~~ $T(n)$ = time to multiply $n \times n$ matrices.

assume n is a power of 2, else can pad A, B to make it so.

. $T(1) = 1$

. $T(n) = 8T(n/2) + O(n^2)$.



n^2

$8 \cdot (n/2)^2$

$8^2 \cdot (n/4)^2$

$\vdots$

$8^{\log_2 n} \cdot 1^2 = n^3$.

Solves to $T(n) = O(n^3)$.

☹

Actually the same as approach #1

we just grouped the products differently.

No surprise got same answer asymptotically.

(Volker) Strassen asked: can we multiply two 2×2 matrices using 7 multiplies instead of 8?

(also don't want to use commutativity of numbers, since $AB \neq BA$ for matrices)

if we can, then can use it recursively as above.

Indeed: If we could do with 7 mults then recursion wins a lot :-

$$T(n) \leq 7T(n/2) + n^2$$
$$= O(7^{\log_2 n}) = O(n^{\log_2 7}) = O(n^{2.81\ldots})$$

Substantially better than $O(n^3)$!!!

Strassen showed in 1969 how to do this —
quite non-trivial, several papers try to give intuition for the idea (see webpage).

$S_1 = (A_{11} + A_{21})(B_{11} + B_{12})$

$S_2 = (A_{12} + A_{22})(B_{21} + B_{22})$

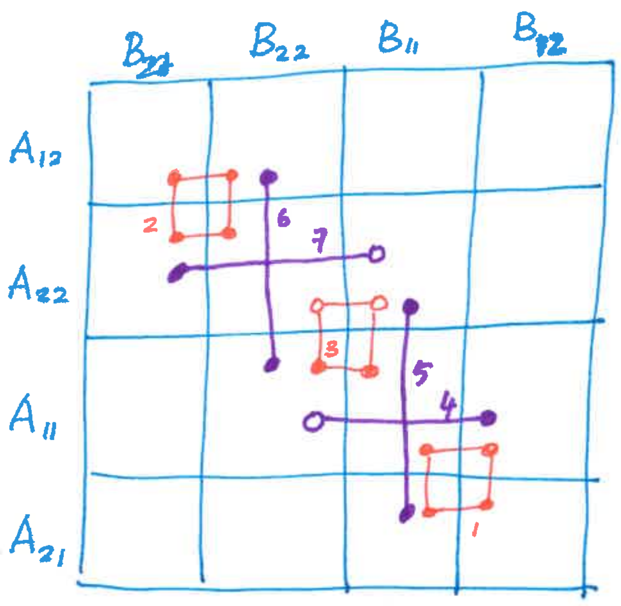
$S_3 = (A_{11} - A_{22})(B_{11} + B_{22})$

$S_4 = A_{11}(B_{12} - B_{22})$

$S_5 = (A_{21} + A_{22}) B_{11}$

$S_6 = (A_{11} + A_{12}) B_{22}$

$S_7 = A_{22}(B_{21} - B_{11})$



then $\begin{pmatrix} S_2 + S_3 - S_6 - S_7 & S_4 + S_6 \\ S_5 + S_7 & S_1 - S_3 - S_4 - S_5 \end{pmatrix}$!

N.b. more additions/subtractions (18). Can be reduced a bit...

Since then, better algorithms, using more ideas. best = $O(n^{2.37\ldots})$

though much more complicated, worse constants, so not practical.

Strassen's algo is very clean, studies suggest can be practical.

Other issues : $\boxed{\text{Can worry about space.}}$

$\exists$ a space-efficient implementation of Strassen too.

Given A, B, set aside space for C. (initially zeros). $\rightarrow 3n^2$ space.

Recursively compute S_1, add to C_{22}

S_2 add to C_{11}

S_3 add to C_{11}, subtract from C_{22} etc.

$$S(n) \leq 3n^2 + S(n/2)$$

space

linear recurrence, so can unroll it

$$= 3n^2 + 3(n/2)^2 + S(n/4) \quad = 3n^2(1 + 1/4 + 1/16 \cdots)$$

$$\leq 3n^2 \cdot 2$$

(in fact $1 + x + x^2 + \cdots = \frac{1}{1-x}$, so $1 + 1/4 + \frac{1}{16} \cdots = \frac{1}{1-\frac{1}{4}} = \frac{4}{3}$.

for $|x|<1$

$$\Rightarrow 3 \cdot n^2 \cdot (4/3) = 4n^2.$$

By the way : Closing gap between $O(n^{2.37})$ and $\Omega(n^2)$ remains open.

Faster Matrix mult would be useful for many other problems too.

(Q) What can we do for sparse graphs triangle counting? $O(m^{1.5})$ can be given. [Exercise*].

Better? Linear time? Lower bounds?

Morals of the Stories:

+ Definitions of the problem may suggest algorithms
  but these may not be the best — to improve, may need to
  rethink the problem.

+ Choose the right representation.
  (If adjacency lists → convert to adj. matrix may help here).

  For triangle counting, matrix representation suggested new algos!

+ If stuck proving upper bounds, see if can prove lower bound.
  And vice versa. (worked for Strassen!)

  May suggest new ideas — "why are we not being able to
  get a stronger lower bound?"

+ For me, insight into Strassen's algo is not the exact formulas, but
  (a) the fact that divide & conquer could get improvement
  (reduction to the 2x2 case)
  (b) computing $X+Y$ does not require computing both. X & Y separately.

Once you have these insights, can find formula yourself with some
elbow grease.