

Triangle Counting and Matrix Multiplication WEH 5415

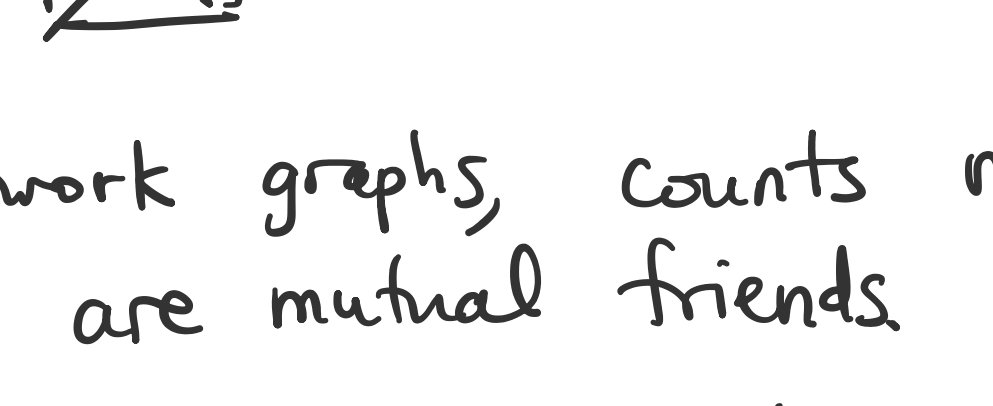
Goal: explore connections between two seemingly unrelated problems

Triangle Counting

Given an undirected graph, find the number of triangles



2 triangles



4 triangles

Ex: In social network graphs, counts number of triples of people who are mutual friends. Measures how well-connected the community is.

Algo 1: naive algorithm. Iterate over all triples.

```

count ← 0
for u = 1...n
  for v = 1...n
    for w = 1...n
      if uv, vw, wu ∈ E // if u,v,w form a triangle
        count += 1
return count / 6 // each triangle counted 6 times
    
```

Assuming checking $uv \in E$ takes $O(1)$ time: $O(n^3)$

Algo 2: edge scan

```

count ← 0
for v = 1..n
  for each pair of edges (v,u), (v,w) incident to v
    if u,v,w form a triangle
      count += 1
return count / 3
    
```

For each vertex v , check $\binom{d_v}{2}$ pairs of edges

Time $\sum_v \binom{d_v}{2} \leq \sum_v d_v^2 \leq d_{\max} \sum_v d_v = d_{\max} \cdot 2m = O(mn)$
 $\sum \text{degrees} = 2m$, "Handshake Lemma"

Faster than $O(n^3)$ when $m \ll n^2$

Tight example: Star, $m \approx n$ so $O(n^2)$

High degrees are expensive. Prioritize low degree?

Algo 3: sort vertices s.t. $d_1 \leq d_2 \leq \dots \leq d_n$

```

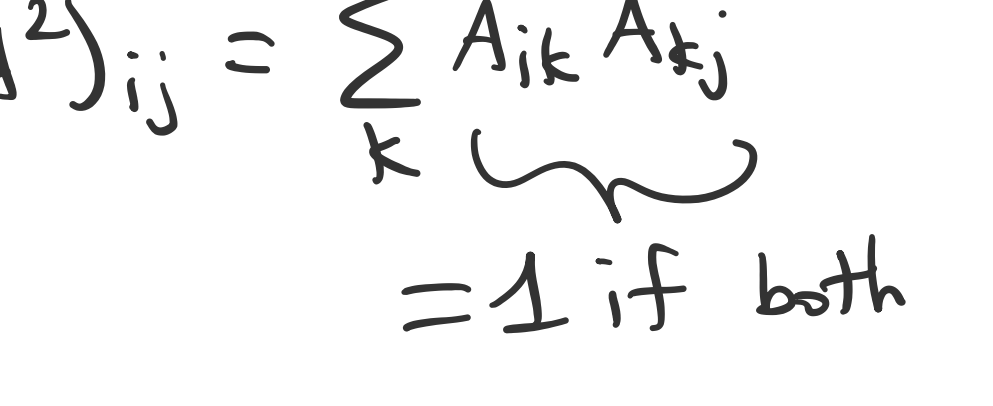
count ← 0
for v = 1..n
  for each pair of edges to later vertices // u > v, w > v
    if u,v,w form a triangle
      count += 1
return count // each triangle now counted once
    
```

On star, only $O(n)$ time!

HW #1: $O(m^{1.5})$ time.

Algo 4: use Matrix Multiplication.

Represent graph in adjacency matrix form:



Consider A^2 : $(A^2)_{ij} = \sum_k A_{ik} A_{kj}$
 $= 1$ if both $(i,k), (k,j) \in E$
 $= \#$ triangles containing (i,j) (if (i,j) is indeed an edge)

```

count ← 0
compute A^2
for i = 1..n
  for j = 1..n
    count ← count + A_{ij} * (A^2)_{ij}
return count / 3 // if (i,j) ∈ E number of k s.t. (i,k), (k,j) ∈ E
    
```

Time: $O(n^2) + T_{MM}(n)$. Fastest algo known in general!

Matrix Multiplication: Given (square) matrices A, B ,

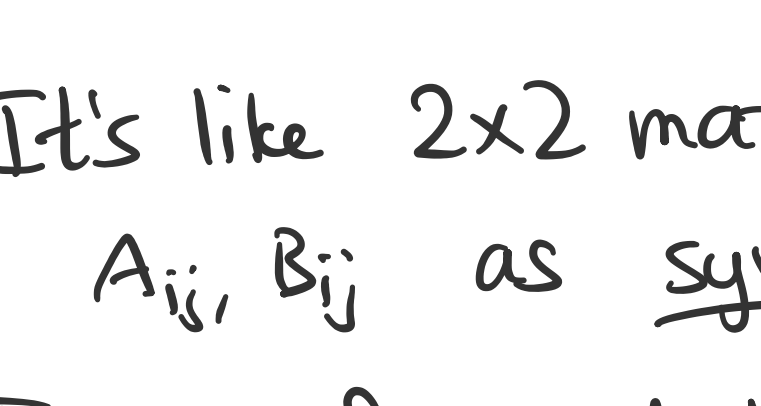
compute $(AB)_{ij} = \sum_k A_{ik} B_{kj} \quad \forall i,j$.

Naive: $O(n)$ time per i,j

$O(n^3)$ time overall.

Can we do better?

Divide and Conquer: Suppose n is even. Break up matrices into 2×2 blocks:



Then $AB = \begin{bmatrix} A_{11}B_{11} + A_{12}B_{21} & A_{11}B_{12} + A_{12}B_{22} \\ A_{21}B_{11} + A_{22}B_{21} & A_{21}B_{12} + A_{22}B_{22} \end{bmatrix}$

It's like 2×2 matrix mult, treating A_{ij}, B_{ij} as symbols.

Time: 8 products of $\frac{n}{2} \times \frac{n}{2}$ matrices

+ $O(n^2)$ time addition.

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

Solve the recurrence:

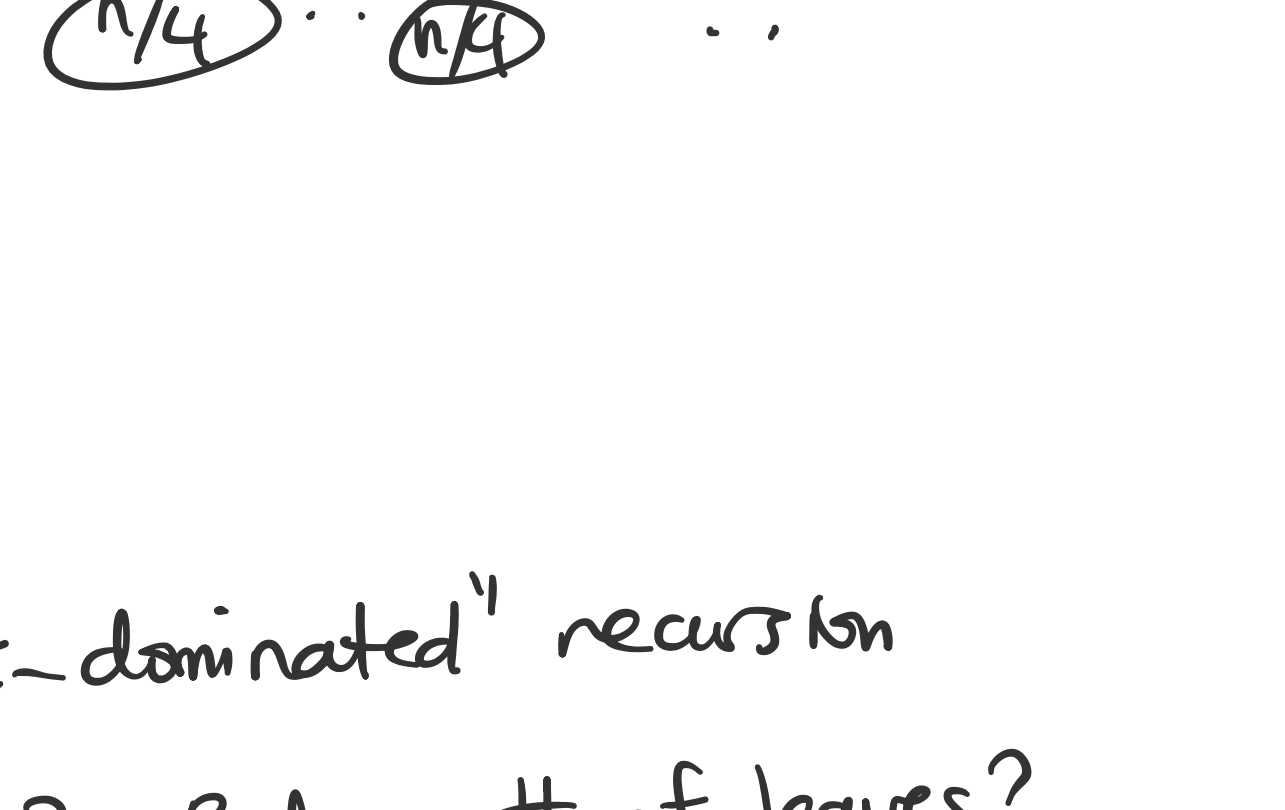
① Unravel the recursion algebraically.

$$\begin{aligned}
 T(n) &= 8T(n/2) + cn^2 \\
 &= 8(8T(n/4) + c(n/2)^2) + cn^2 \\
 &= 8^2 T(n/4) + cn^2(1+2) \\
 &= 8^2(8T(n/8) + c(n/4)^2) + cn^2(1+2) \\
 &= 8^3 T(n/8) + cn^2(1+2+4) \\
 &\quad \vdots \\
 &= 8^i T(n/2^i) + cn^2(1+2+\dots+2^{i-1})
 \end{aligned}$$

Stop at $2^i = n \Leftrightarrow i = \log_2 n$:

$$\begin{aligned}
 &= 8^{\log_2 n} T(1) + cn^2(1+2+\dots+2^{\log_2 n - 1}) \\
 &= \underbrace{(2^3)^{\log_2 n}}_{= 2^{\log_2 n \cdot 3} = n^3} \cdot 1 + \underbrace{cn^2(1+2+\dots+2^{\log_2 n - 1})}_{= 2^{\log_2 n} - 1 = n - 1} \\
 &= n^3 \\
 &= O(n^3).
 \end{aligned}$$

② Recursion tree.



$$\begin{aligned}
 &1 \times cn^2 = cn^2 \\
 &8 \times c(n/2)^2 = 2cn^2 \\
 &8^2 \times c(n/4)^2 = 4cn^2 \\
 &\vdots \\
 &8^i \times c(1)^2 \\
 &= 8^{\log_2 n} \times c = O(n^3)
 \end{aligned}$$

"root-dominated" recursion

Faster algo? Reduce # of leaves? Reduce # of products of $\frac{n}{2} \times \frac{n}{2}$ matrices.

Stassen found a way to multiply 2×2 symbolic matrices using only 7 multiplies!

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} = \begin{bmatrix} S_2 + S_3 - S_6 - S_7 & S_4 + S_6 \\ S_5 + S_7 & S_1 - S_3 - S_4 - S_5 \end{bmatrix}$$

where $S_1 = (a_{11} + a_{21})(b_{11} + b_{12})$

$S_2 = (a_{12} + a_{22})(b_{21} + b_{22})$

$S_3 = (a_{11} - a_{22})(b_{11} + b_{22})$

$S_4 = a_{11}(b_{12} - b_{22})$

$S_5 = (a_{21} + a_{22})b_{11}$

$S_6 = (a_{11} + a_{12})b_{22}$

$S_7 = a_{22}(b_{21} - b_{11})$

$T(n) = 7T(n/2) + O(n^2)$

↑ more additions now, but doesn't matter

Solves to $7^{\log_2 n} = O(n^{\log_2 7}) \approx O(n^{2.81})$.

Recap: Seemingly unrelated problems can be closely connected

Divide and conquer works in surprising ways