Lecture 11. Matrix Algorithms and Applications

Introduction to Computer Science, IIIS, Tsinghua
Outline

- Introduction to divide and conquer
  - faster matrix multiplication algorithm
- Algorithms for matrix operations
  - LUP decomposition, matrix inversion, …
- Applications
  - Perfect matching by matrix multiplication
  - 4-node subgraphs
  - dominance product / (max,min)-product
Divide and Conquer

- Break up a problem into two sub-problems
- Solve each sub-problem **recursively**
- Combine solution to sub-problems to form solution to original problem.
Integer Multiplication

- **Addition**: Given two n-digit integers \(a\) and \(b\), compute \(a + b\).
  - \(O(n)\) bit operations.

- **Multiplication**: Given two n-digit integers \(a\) and \(b\), compute \(a \times b\).
  - **Brute force solution**: \(\Theta(n^2)\) bit operations.
Divide-and-Conquer Multiplication

- We can divide the *n-bit numbers* \(a\) and \(b\) into two parts:

\[
a = a_1N + a_2, \quad b = b_1N + b_2. \quad (N=2^{n/2})
\]

- So \(a \times b = (a_1N + a_2)(b_1N + b_2) = a_1b_1N^2 + (a_1b_2 + a_2b_1)N + a_2b_2\).

- Suppose we can compute the multiplication of two \((n/2)\)-bit numbers,
  - We need to use it 4 times.
  - Compute \(a_1b_1, a_1b_2, a_2b_1, a_2b_2\).
Divide-and-Conquer Multiplication

- We can divide the n-bit numbers a and b into two parts:

  ![Diagram of a and b divided into two parts](image)

  where \(a = a_1N + a_2\), \(b = b_1N + b_2\). \((N = 2^{n/2})\)

  so \(a \times b = (a_1N + a_2)(b_1N + b_2) = a_1b_1N^2 + (a_1b_2 + a_2b_1)N + a_2b_2\).

- Suppose we can compute the multiplication of two \((n/2)\)-bit numbers,
  - We need to use it 4 times.
  - Compute \(a_1b_1, a_1b_2, a_2b_1, a_2b_2\)

\[
T(n) = \frac{4T(n/2)}{14} + \frac{\Theta(n)}{23} \quad \Rightarrow \quad T(n) = \Theta(n^2)
\]

How can we improve?
Divide-and-Conquer Multiplication

- We can divide *the n-bit numbers* $a$ and $b$ into two parts:

  $a = a_1 + a_2$
  $b = b_1 + b_2$

- in fact, $a_1b_1 + a_1b_2 + a_2b_1 + a_2b_2 = (a_1 + a_2)(b_1 + b_2)$.
- So if we have already computed $a_1b_1$ and $a_2b_2$,
- $a_1b_2 + a_2b_1 = (a_1 + a_2)(b_1 + b_2) - a_1b_1 - a_2b_2$
Division-and-Conquer Multiplication

- We can divide the n-bit numbers a and b into two parts:

  \[ a = a_1 a_2 \]
  \[ b = b_1 b_2 \]

Compute \( A = a_1 b_1, B = a_2 b_2, C = (a_1 + a_2)(b_1 + b_2) \)

\[
a \cdot b = (a_1 N + a_2)(b_1 N + b_2)
= a_1 b_1 N^2 + (a_1 b_2 + a_2 b_1) N + a_2 b_2
= A \cdot N^2 + (C - A - B) N + B
\]
Theorem. [Karatsuba-Ofman, 1962] Can multiply two n-digit integers in $O(n^{1.585})$ bit operations.

$$A=a_1b_1, B=a_2b_2, C=(a_1+a_2)(b_1+b_2)$$
$$a\cdot b=(a_1N+a_2)(b_1N+b_2)$$
$$=a_1b_1N^2+(a_1b_2+a_2b_1)N+a_2b_2$$
$$=A\cdot N^2+(C-A-B)N+B$$
Karatsuba: Recursion Tree

\[ T(n) = \begin{cases} 
0 & \text{if } n = 1 \\
3T(n/2) + n & \text{otherwise}
\end{cases} \]

\[
T(n) = \sum_{k=0}^{\log_2 n} n \left(\frac{3}{2}\right)^k = \frac{\left(\frac{3}{2}\right)^{1+\log_2 n} - 1}{\frac{3}{2} - 1} = 3n^{\log_2 3} - 2
\]
Integer Multiplication

- [Karatsuba-Ofman, 1962] $O(n^{1.585})$

- [Schönhage, Strassen 1971] $O(n \cdot \log n \cdot \log \log n)$

- [Fürer 2007] $O(n \cdot \log n \cdot 2^{O(\log^* n)})$
  - $\log^* n$ is how many times $\log \log \ldots \log n \leq 1$

- Almost in linear time!
Algebraic Matrix Multiplication

\[ A = (a_{ij}) \times B = (b_{ij}) = C = (c_{ij}) \]

\[ c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj} \]

Can be computed naively in \( O(n^3) \) time.
Strassen’s $n \times n$ algorithm

View each $n \times n$ matrix as a $2 \times 2$ matrix whose elements are $n/2 \times n/2$ matrices.

Apply the $2 \times 2$ algorithm recursively.
Multiplying $2 \times 2$ matrices

\[
\begin{pmatrix}
C_{11} & C_{12} \\
C_{21} & C_{22}
\end{pmatrix} =
\begin{pmatrix}
A_{11} & A_{12} \\
A_{21} & A_{22}
\end{pmatrix} \begin{pmatrix}
B_{11} & B_{12} \\
B_{21} & B_{22}
\end{pmatrix}
\]

\[
C_{11} = A_{11}B_{11} + A_{12}B_{21}
\]
\[
C_{12} = A_{11}B_{12} + A_{12}B_{22}
\]
\[
C_{21} = A_{21}B_{11} + A_{22}B_{21}
\]
\[
C_{22} = A_{21}B_{12} + A_{22}B_{22}
\]

8 multiplications
4 additions

\[
T(n) = 8T(n/2) + \mathcal{O}(n^2)
\]

\[
T(n) = \mathcal{O}(n^{\log_8 8/\log 2}) = \mathcal{O}(n^3)
\]
Strassen’s 2×2 algorithm

\[ C_{11} = A_{11}B_{11} + A_{12}B_{21} \]
\[ C_{12} = A_{11}B_{12} + A_{12}B_{22} \]
\[ C_{21} = A_{21}B_{11} + A_{22}B_{21} \]
\[ C_{22} = A_{21}B_{12} + A_{22}B_{22} \]

\[ M_1 = (A_{11} + A_{22})(B_{11} + B_{22}) \]
\[ M_2 = (A_{21} + A_{22})B_{11} \]
\[ M_3 = A_{11}(B_{12} - B_{22}) \]
\[ M_4 = A_{22}(B_{21} - B_{11}) \]
\[ M_5 = (A_{11} + A_{12})B_{22} \]
\[ M_6 = (A_{21} - A_{11})(B_{11} + B_{12}) \]
\[ M_7 = (A_{12} - A_{22})(B_{21} + B_{22}) \]

\[ C_{11} = M_1 + M_4 - M_5 + M_7 \]
\[ C_{12} = M_3 + M_5 \]
\[ C_{21} = M_2 + M_4 \]
\[ C_{22} = M_1 - M_2 + M_3 + M_6 \]

7 multiplications
18 additions/subtractions
\[
C_{11} = A_{11}B_{11} + A_{12}B_{21} \\
C_{12} = A_{11}B_{12} + A_{12}B_{22} \\
C_{21} = A_{21}B_{11} + A_{22}B_{21} \\
C_{22} = A_{21}B_{12} + A_{22}B_{22}
\]

\[
M_1 = (A_{11} + A_{22})(B_{11} + B_{22}) \\
M_2 = (A_{21} + A_{22})B_{11} \\
M_3 = A_{11}(B_{12} - B_{22}) \\
M_4 = A_{22}(B_{21} - B_{11}) \\
M_5 = (A_{11} + A_{12})B_{22} \\
M_6 = (A_{21} - A_{11})(B_{11} + B_{12}) \\
M_7 = (A_{12} - A_{22})(B_{21} + B_{22})
\]

7 multiplications
18 additions/subtractions

\[
T(n) = \frac{7}{14 \cdot 2^{43}} T\left(\frac{n}{2}\right) + 1 \\ \Theta\left(n^2\right) \quad \Rightarrow \quad T(n) = \Theta(n^{\log_2 7}) = O(n^{2.81})
\]

recursive calls add, subtract
Matrix multiplication algorithms

The $O(n^{2.81})$ bound of Strassen was improved by Coppersmith and Winograd to $O(n^{2.376})$. The current bound is $O(n^{2.3729})$.

(The algorithms are much more complicated...)

We let $2 \leq \omega < 2.376$ be the exponent. Many believe that $\omega = 2 + o(1)$. 
We let $2 \leq \omega < 2.373$ be the exponent of the time complexity. Many believe that $\omega = 2+o(1)$.

So we can denote the time for matrix multiplication by $O(n^\omega)$.
Other matrix operations?

- Matrix inversion
- Determinant (行列式)
- Solve linear system \((Ax=b)\)
- Computing eigenvalue/eigenvectors

(using the fast matrix multiplication algorithm)
Other matrix operations?

- Matrix inversion
- Determinant (行列式)
- Solve linear system \((Ax=b)\)
- Computing eigenvalue/eigenvectors

(using the fast matrix multiplication algorithm)
LUP decomposition

- For an $n \times n$ matrix $A$, find $L, U, P$, such that $PA = LU$ ($A = PLU$)
  - $L$ is a unit lower-triangular matrix
  - $U$ is an upper-triangular matrix
  - $P$ is a permutation matrix (*what’s the inverse of $P$?*)

**Permutation matrix:** every row and every column has 1 element of 1
LU decomposition

- For an \( n \times n \) matrix \( A \), find \( L, U \), such that \( A = LU \)
  - \( L \) is a unit lower-triangular matrix
  - \( U \) is an upper-triangular matrix

- Just the Gaussian elimination

\[
\begin{bmatrix}
a \\
\end{bmatrix}
\begin{bmatrix}
w^T \\
\end{bmatrix}
\begin{bmatrix}
A' \\
\end{bmatrix}
= \begin{bmatrix}
1 \\
\end{bmatrix}
\begin{bmatrix}
v / a \\
\end{bmatrix}
\times \begin{bmatrix}
a \\
w^T \\
\end{bmatrix}
\]
LU decomposition

For an \( n \times n \) matrix \( A \), find \( L, U \), such that \( A = LU \)
- \( L \) is a unit lower-triangular matrix
- \( U \) is an upper-triangular matrix

Just the Gaussian elimination

\[
egin{align*}
\begin{bmatrix} a & w^T \\ v & A' \end{bmatrix} &= \begin{bmatrix} L & \vdots \\ \vdots & \vdots \end{bmatrix} \begin{bmatrix} v/a \\ \vdots \end{bmatrix} \\
&= \begin{bmatrix} a & w^T \\ A' - vw^T/a \end{bmatrix}
\end{align*}
\]

Recursively perform on \( A' - vw^T/a \)
LU decomposition

- For an $n \times n$ matrix $A$, find $L, U$, such that $A = LU$
  - $L$ is a unit lower-triangular matrix
  - $U$ is an upper-triangular matrix

- Just the Gaussian elimination

```
\begin{align*}
a & \quad w^T \\
v & \quad A' \\
\end{align*}
\begin{align*}
\begin{bmatrix}
I \\
v/a
\end{bmatrix} & \times \\
\begin{bmatrix}
A' - vw^T/a
\end{bmatrix}
\end{align*}
```

Recursively perform on $A' - vw^T/a$

---

What’s the problem with an arbitrary matrix?
LU decomposition

- For an \( n \times n \) matrix \( A \), find \( L, U \), such that \( A = LU \)
  - \( L \) is a unit lower-triangular matrix
  - \( U \) is an upper-triangular matrix

- Just the Gaussian elimination

- What if the first element \( a_{11} = 0 \)?
  - Change the order of the rows by multiplying a permutation matrix \( Q \)
LUP decomposition – Gaussian Elimination

- **Elementary matrix:**

\[
T_{i,j} = \begin{bmatrix}
1 & & & \\
& \ddots & & \\
& & 0 & 1 \\
& & 1 & 0 \\
& & & \ddots \\
& & & & 1
\end{bmatrix}
\]

\[
L_{i,j}(m) = \begin{bmatrix}
1 & & & \\
& \ddots & & \\
& & 1 & \\
& & m & 1 \\
& & & \ddots \\
& & & & 1
\end{bmatrix}
\]

\[
D_{i}(m) = \begin{bmatrix}
1 & & & \\
& \ddots & & \\
& & 1 & \\
& & 1 & m \\
& & & \ddots \\
& & & & 1
\end{bmatrix}
\]

Left multiply: change rows
Right multiply: change columns

- **Gaussian elimination:**
  - a sequence of elementary matrix \(E_1, E_2, \ldots, E_k\) s.t. \(A = E_1 \ldots E_k U\)
  - How to deal with the permutation matrix \(T_{i,j}\)?
LUP decomposition

If we change the order of the 2nd line and the i-th line

\[
\begin{bmatrix}
1 & 1 \\
a_{21} & 1 \\
a_{31} & 1 \\
\vdots & \ddots \\
a_{i1} & \ddots \\
a_{n1} & \ddots & 1
\end{bmatrix}
\begin{bmatrix}
1 \\
0 \\
1 \\
\vdots \\
1 \\
0
\end{bmatrix}
\times
\begin{bmatrix}
A' - vw^T / a
\end{bmatrix}
\]
LUP decomposition

If we change the order of the 2nd line and the i-th line

Right multiply: change columns
LUP decomposition

If we change the order of the 2nd line and the i-th line

Right multiply: change columns
LUP decomposition

\[
\begin{bmatrix}
a \\
v \\
A'
\end{bmatrix}
= \begin{bmatrix}
1 \\
v/a \\
\vdots
\end{bmatrix}
\times
\begin{bmatrix}
a \\
w^T \\
A' - vw^T/a
\end{bmatrix}
\]

If we change the order of the 2nd line and the i-th line

\[
\begin{pmatrix}
1 & 0 & 1 \\
0 & 1 & 0 \\
\vdots & \ddots & \vdots \\
0 & 0 & 1 \\
1 & 0 & 1
\end{pmatrix}
\times
\begin{pmatrix}
1 \\
B
\end{pmatrix}
\]

Left multiply: change rows
LUP decomposition

If we change the order of the 2nd line and the i-th line

Still a lower triangle matrix!
LUP decomposition summary

- LU-decomposition $A = LU$
- When the element $a_{ii} = 0$
  - Change the order of the rows by multiplying a permutation matrix $Q$
- Finally $A = E_1 \ldots E_k U$, for a sequence of elementary matrix

\[
\begin{pmatrix}
1 & 1 \\
a_{21} & 1 \\
a_{31} & 1 \\
\vdots & \ddots \\
a_{il} & \ddots & 1 \\
a_{n1} & \ddots & \ddots & 1
\end{pmatrix}
\begin{pmatrix}
1 & 0 & 1 \\
1 & 1 & \ddots \\
1 & \ddots & 1 \\
& \ddots & \ddots & 1 \\
& & \ddots & \ddots & 1
\end{pmatrix}
\begin{pmatrix}
1 & 0 & 1 \\
0 & 1 & \ddots \\
& \ddots & \ddots & 1 \\
& & \ddots & \ddots & 1 \\
& & & \ddots & \ddots & 1
\end{pmatrix}
\]

- So finally we can write $A = PLU$
  - $L$ is a unit lower-triangular matrix
  - $U$ is an upper-triangular matrix
  - $P$ is a permutation matrix
LUP-decomposition

- After the LUP-decomposition $A=PLU$, every operation becomes easy:
  - find inversion
  - compute determinant
    \[ \det(A) = \det(P)\det(L)\det(U) \]
  - Solve linear system $(Ax=b)$
    \[ (PLU)x=b \iff \{Py=b; Lz=y; Ux=z\}, \text{takes } O(n^2) \text{ time.} \]
LUP-decomposition

- After the LUP-decomposition, every operation becomes easy:
  - find inversion
  - compute determinant
  - Solve linear system \((Ax=b)\)

- How can we find the inverse of a triangle matrix?
  - better than \(O(n^3)\)
LUP-decomposition

- After the LUP-decomposition, every operation becomes easy:
  - find inversion
  - compute determinant
  - Solve linear system \((Ax=b)\)

- How can we find the inverse of a triangle matrix?

  \[
  \begin{bmatrix}
  A & C \\
  0 & B
  \end{bmatrix}^{-1} = \begin{bmatrix}
  A^{-1} & -A^{-1}CB^{-1} \\
  0 & B^{-1}
  \end{bmatrix}
  \]

  - So we can recursively compute \(A^{-1}\) and \(B^{-1}\), (also triangle)
  - Time: \(T(n) = 2T(n/2) + \Theta(n^\omega) = \Theta(n^\omega)\)
    (the same as matrix multiplication)
What’s the running time for Gaussian elimination?
What’s the running time for Gaussian elimination?

- $O(n^3)$

How can we do the LUP-decomposition faster?
How to accelerate LUP decomposition?

- **Gaussian elimination:**
  - Eliminate elements by order:
  - $O(n^3)$ time.
How to accelerate LUP decomposition?

- The Hopcroft-Bunch (1974) version:
  - Eliminate elements by order:
Hopcroft-Bunch algorithm
Hopcroft-Bunch algorithm
Hopcroft-Bunch algorithm

• $U$ becomes upper triangle!
• Find $U^{-1}$ in $O(n^\omega)$ time
Hopcroft-Bunch algorithm

• Block elimination!
Hopcroft-Bunch algorithm
Hopcroft-Bunch algorithm
Hopcroft-Bunch algorithm

- \(U\) is always upper-triangle, so it’s easy to find its inverse.
- We just need to pick a non-zero element at \(k\)-th row.
Hopcroft-Bunch algorithm

Recursively run this on $Y - VU^{-1}X$
Hopcroft-Bunch algorithm

Every time we just need to multiply a lower-triangle matrix:
Running Time:

- $M(n,p)$: time for multiplying an $n \times n$ and an $n \times p$ matrix
- $I(n)$: time for inverse an $n \times n$ matrix

Time:

Inversions:

\[
\frac{n}{2} I(1) + \frac{n}{4} I(2) + \ldots + I(n/2) = \sum_{i=1}^{\log_2 n} \frac{n}{2^i} I(2^{i-1})
\]
Running Time:

- \( M(n, p) \): time for multiplying an \( n \times n \) and an \( n \times p \) matrix
- \( I(n) \): time for inverse an \( n \times n \) matrix

Time:

- Inversions:
  \[
  \frac{n}{2} I(1) + \frac{n}{4} I(2) + \ldots + I(n/2) = \sum_{i=1}^{\log_2 n} \frac{n}{2^i} I(2^{i-1})
  \]

- Multiplications:
  every time we need to compute \( YVU^{-1}X \)
  \[
  \leq \sum_{i=1}^{\log_2 n} \frac{n}{2^i} M(2^{i-1}, n)
  \]
Running Time:

- **M(n,p):** time for multiplying an $n \times n$ and an $n \times p$ matrix
- **I(n):** time for inverse an $n \times n$ matrix

Time:

- **Inversions:**
  \[
  \frac{n}{2} I(1) + \frac{n}{4} I(2) + \ldots + I(n/2) = \sum_{i=1}^{\log n} \frac{n}{2^i} I(2^{i-1})
  \]

- **Multiplications:**
  every time we need to compute $Y - VU^{-1}X$
  \[
  \sum_{i=1}^{\log n} \frac{n}{2^i} M(2^{i-1}, n)
  \]
Running Time:

- **M(n,p):** \( O(pn^{\omega-1}) \)
  
  - a trivial method, there is good ways

- **I(n):** \( O(n^\omega) \)

Time:

- **Inversions:**
  \[
  \frac{n}{2} I(1) + \frac{n}{4} I(2) + \ldots + I(n/2) = \sum_{i=1}^{\log n} \frac{n}{2^i} I(2^{i-1})
  \]

- **Multiplications:**
  
  every time we need to compute \( Y-VU^{-1}X \)
  
  \[
  \leq \sum_{i=1}^{\log n} \frac{n}{2^i} M(2^{i-1},n)
  \]
  
  Thus, the total time is still \( O(n^\omega) \).
Summary

- A reduces to B: we can use an algorithm for B to solve A
  \((A \Rightarrow B)\)

- We have shown:
  
  - Matrix Multiplication
  
  - LUP-decomposition
  
  - Matrix inversion

- so LUP-decomposition and matrix inversion has \(O(n^\omega)\) time algorithm.
Summary

- A reduces to B: we can use an algorithm for B to solve A \((A \Rightarrow B)\)

- We have shown:
  - if we can show this reduction, then the three operations has the same time complexity!
    - there is an oracle for matrix inversion, compute \(A \times B\)?
Multiplication reduces to Inversion

Very tricky result:

To compute $AB$,

$$
\begin{pmatrix}
I & A \\
I & B \\
I & I
\end{pmatrix}^{-1} = \begin{pmatrix}
I & -A & AB \\
I & I & -B \\
I & I
\end{pmatrix}
$$
Summary

- A reduces to B: we can use an algorithm for B to solve A
  \((A \Rightarrow B)\)

Matrix Multiplication

LUP-decomposition

Compute \(\det(A)\)

Solve \(Ax = b\)

Matrix inversion

All operations can be solved in \(O(n^\omega)\) time.
Eigenvalues/Eigenvector?

- However, there is no such algorithm by FMM for computing eigenvalues.
Example of applications

- **Matching M:**
  - A set of vertex-disjoint edges
  - Matched vertices: the vertices associated with an edge in M
  - Free vertices: unmatched vertices

- **Perfect matching in bipartite graph:**
  - No free vertices
Another example

- Perfect matching in bipartite graph:
  - Problem: decide whether there is a perfect matching in $G$
Adjacency matrix

- The adjacency matrix $A$ of graph $G=(V, E)$:
  - $A_{i,j}=1$ iff $(i,j) \in E$
Perfect matching in bipartite graph:
- Whether there is a perfect matching in $G$

By determinant of adjacency matrix

$$
\begin{bmatrix}
1 & 1 & 1 & 0 \\
1 & 0 & 1 & 0 \\
0 & 1 & 0 & 1 \\
0 & 0 & 1 & 1 \\
\end{bmatrix}
$$
Another example

- Perfect matching in bipartite graph:
  - Whether there is a perfect matching in $G$

- By determinant of adjacency matrix

$$\det(A) = \sum_{\sigma \in S_n} \text{sgn}(\sigma) \prod_{i=1}^{n} a_{i,\sigma_i}.$$  

- $\sigma$ is the permutation of \{1,...,n\}
- $\text{sgn}(\sigma)$ is +1 or -1
Another example

- Perfect matching in bipartite graph:
  - Whether there is a perfect matching in $G$

- By determinant of adjacency matrix

\[
\det(A) = \sum_{\sigma \in S_n} \text{sgn}(\sigma) \prod_{i=1}^{n} a_{i,\sigma_i}.
\]

- $\sigma$ is the permutation of $\{1, \ldots, n\}$
- $\text{sgn}(\sigma)$ is $+1$ or $-1$

Permutation means every row and column contains exactly one element
A perfect matching in a bipartite graph with $n$ vertices each side can also be seen as a permutation of $\{1, \ldots, n\}$.
A perfect matching in a bipartite graph with $n$ vertices each side can also be seen as a permutation of $\{1, \ldots, n\}$.

Construct an $n \times n$ matrix $X$:

\[
\begin{bmatrix}
    x_{11} & x_{12} & x_{13} & 0 \\
    x_{21} & 0   & x_{23} & 0 \\
    0     & x_{32} & 0     & x_{34} \\
    0     & 0     & x_{43} & x_{44}
\end{bmatrix}
\]

- $x_{ij}$ is a variable if there is an edge from $A_i$ to $B_j$.
- $x_{ij} = 0$ if there is no edge from $A_i$ to $B_j$. 
A perfect matching in a bipartite graph with $n$ vertices each side can also be seen as a permutation of $\{1, \ldots, n\}$.

Construct an $n \times n$ matrix $X$:

$$
\begin{bmatrix}
 x_{11} & x_{12} & x_{13} & 0 \\
 x_{21} & 0 & x_{23} & 0 \\
 0 & x_{32} & 0 & x_{34} \\
 0 & 0 & x_{43} & x_{44}
\end{bmatrix}
$$

From the definition of determinant, $\det(X) \equiv 0$ if there is no perfect matching.

$$
\det(A) = \sum_{\sigma \in S_n} \text{sgn}(\sigma) \prod_{i=1}^{n} a_{i,\sigma_i}.
$$
The determinant is a polynomial of degree $n$

$$\det(A) = \sum_{\sigma \in S_n} \text{sgn}(\sigma) \prod_{i=1}^{n} a_{i,\sigma_i}.$$ 

We can randomly pick each nonzero $x_{ij}$ from $\{1, \ldots, 2n\}$, and compute $\det(X)$

- if there is no perfect matching, $\det(X)$ is always 0
- otherwise, $\det(X)$ is not 0 unless we are unlucky enough to pick the root of the polynomial $\det(X)$.
- Computing $\det(X)$ takes $O(n^\omega)$ time.

$$
\begin{array}{cccc}
5 & 3 & 8 & 0 \\
1 & 0 & 4 & 0 \\
0 & 7 & 0 & 3 \\
0 & 0 & 2 & 6 \\
\end{array}
$$
The determinant is a polynomial of degree $n$

- We can randomly pick each nonzero $x_{ij}$ from $\{1, \ldots, 2n\}$, and compute $\det(X)$
  - As in the polynomial identity testing
  - Computing $\det(X)$ takes $O(n^\omega)$ time.

- If there is no perfect matching, determinant is always 0.
- If there are perfect matchings, $\Pr[\det(X) \neq 0] \geq 1/2$
  - Proof in the “Theory of Computation” course.

- So we can repeat $k$ time to make error probability $\leq 1/2^k$. 
Graph algorithms by FMM

- Transitive Closure: $O(n^\omega)$
- All-pair shortest path (APSP):
  - undirected unweighted: $O(n^\omega)$
  - directed unweighted: $O(n^{\mu})$
  - $(1+\varepsilon)$-approximate with weights $[0…M]$: $O(n^\omega \log M/\varepsilon)$
- Maximum Matching: $O(n^\omega)$
- Dominance Product: $O(n^{(3+\omega)/2})$
- All-pair bottleneck path (APBP):
  - vertex-capacitated graphs: $O(n^{\mu})$
  - edge-capacitated graphs: $O(n^{(3+\omega)/2})$
Graph algorithms by FMM

- Transitive Closure: $O(n^\omega)$
- All-pair shortest path (APSP):
  - undirected unweighted: $O(n^\omega)$
  - directed unweighted: $O(n^\mu)$
  - $(1+\varepsilon)$-approximate with weights $[0\ldots M]$: $O(n^\omega \log M/\varepsilon)$
- Maximum Matching: $O(n^\omega)$
- Dominance Product: $O(n^{(3+\omega)/2})$
- All-pair bottleneck path (APBP):
  - vertex-capacitated graphs: $O(n^\mu)$
  - edge-capacitated graphs: $O(n^{(3+\omega)/2})$

$\omega \approx 2.373$, $\mu \approx 2.531$, $(3+\omega)/2 \approx 2.687$
Simple problem

- In a undirected graph $G$, check whether $G$ contains a triangle.
  - trivial algorithm: checking every triple of vertices $(u,v,w)$
  - time: $O(n^3)$
Simple problem

- In a undirected graph $G$, check whether $G$ contains a triangle.
- By FMM?
Adjacency matrix of $G=(V,E)$

- $A_{i,j}=1$ iff $(i,j) \in E$
- Consider $A^2$:
  - $(A^2)_{i,j}=\sum_k A_{i,k} \cdot A_{k,j}$

\[ A = (a_{ij}) \quad \times \quad B = (b_{ij}) \quad = \quad C = (c_{ij}) \]
Adjacency matrix of $G=(V,E)$

- $A_{i,j} = 1$ iff $(i,j) \in E$
- Consider $A^2$:
  - $(A^2)_{i,j} = \sum_k A_{i,k} \cdot A_{k,j}$
  - $(A^2)_{i,j} > 0$ equivalent to $\exists k$ such that $A_{i,k} = A_{k,j} = 1$,
  - that is $(i,k), (k,j) \in E$
Adjacency matrix of $G=(V,E)$

- $A_{i,j} = 1$ iff $(i,j) \in E$
- Consider $A^2$:
  - $(A^2)_{i,j} = \sum_k A_{i,k} \cdot A_{k,j}$
  - $(A^2)_{i,j} > 0$ equivalent to $\exists k$ such that $A_{i,k} = A_{k,j} = 1$, that is $(i,k), (k,j) \in E$
- So $G$ has a triangle: $\exists (i,j)$ such that $A_{i,j} = 1$ and $(A^2)_{i,j} \geq 1$
What about Quadrilateral?

- Checking whether a graph contains a 4-cycle:
What about Quadrilateral?

- Checking whether a graph contains a 4-cycle:
  - There must be \((i,j)\) such that \((A^2)_{i,j} \geq 2:\)
    - \((A^2)_{2,6} = 2\) since \(A_{2,7} = A_{7,6} = 1, A_{2,5} = A_{5,6} = 1\)
  - Thus time bound is also \(O(n^\omega)\)
Problem solved?

- What about the 4-node subgraph is exactly a 4 cycle?
  - subgraph on the 4 vertices has 4 edges

![Diagram of a graph with nodes labeled 1 to 7 and edges connecting them. The left diagram shows a correctly connected 4-node subgraph, while the right diagram shows an incorrectly connected 4-node subgraph.]
Problem solved?

- What about the 4-node subgraph is exactly a 4 cycle?
  - subgraph on the 4 vertices has 4 edges
Randomized algorithm

- [Vassilevska, Wang, Williams, Yu 2014] all 4-node subgraphs except “clique” and “independent set” can be found in $O(n^{\omega})$ time with high probability.
Ideas

- Consider
  \[ \sum_{(u,v) \in E} \binom{(A^2)_{u,v}}{2} \]

- What if we have a “diamond”

- What if we have a “clique”
Ideas

- **Consider**
  \[ \sum_{(u,v) \in E} \binom{(A^2)_{u,v}}{2} \]

- if we have a “diamond”
  - can only be:

- if we have a “clique”
  - any edge can be \((u,v)\)
  - so it is counted 6 times
Ideas

- So

\[
\sum_{(u,v) \in E} \binom{(A^2)_{u,v}}{2} = 6 \times \#(\text{\begin{figure}[h] \centering \includegraphics[width=0.1\textwidth]{example.png} \end{figure}}) + \#(\text{\begin{figure}[h] \centering \includegraphics[width=0.1\textwidth]{example.png} \end{figure}})
\]

- But it’s still hard to separate them
Ideas

So

\[ X = \sum_{(u,v) \in E} \binom{(A^2)_{u,v}}{2} = 6 \times \#(\text{\textup{\small\textbullet\textbullet\textbullet\textbullet}}) + \#(\text{\textup{\small\textbullet\textbullet\textbullet\textbullet}}) \]

If \( X \mod 6 \neq 0 \), then of course the graph contains a diamond \( \text{\textup{\small\textbullet\textbullet\textbullet\textbullet}} \)

If \( X \mod 6 = 0 \), we randomly delete some vertices

- with some probability, the number of diamonds will not be a multiple of 6, then return true
- we can run this procedure many times, if the graph contains no diamond, it always return false
What about 4-cycle?
What about 4-cycle?

- Consider
  \[
  \sum_{(u,v) \notin E} \binom{(A^2)_{u,v}}{2}
  \]

- If we have a diamond…
- If we have a 4-cycle…
What about 4-cycle?

- Consider

\[
\sum_{(u,v) \notin E} \left( \binom{(A^2)_{u,v}}{2} \right)
\]

- If we have a diamond, it is counted once!!

- If we have a 4-cycle, it is counted twice
Ideas

So

\[ X = \sum_{(u,v) \in E} \binom{(A^2)_{u,v}}{2} = 6 \times \#(\text{clique}) + \#(\text{4-cycle}) \]

\[ Y = \sum_{(u,v) \notin E} \binom{(A^2)_{u,v}}{2} = 2 \times \#(\text{clique}) + \#(\text{4-cycle}) \]

So by \( X-Y=6 \times \#\text{(clique)}-2 \times \#\text{(4-cycle)} \), using similar idea we can check whether a graph contains a 4-cycle.
Dominance Product (♦)

- **Original product**
  - \( C[i,j] = \sum_k A[i,k] \cdot B[k,j] \)

- **Dominance Product**
  - \( M[i,j] = (A \, ♦ \, B)[i,j] = |\{k \mid A[i,k] \leq B[k,j]\}| \)
Dominance Product (♦)

- **Dominance Product**
  - \( M[i,j] = (A \Diamond B)[i,j] = |\{k \mid A[i,k] \leq B[k,j]\}| \)

\[ M[i,j] = 3 \text{ (Dominance Product)} \]
Dominance Product (♦)

- **Dominance Product**
  - $M[i,j] = (A \bowtie B)[i,j] = |\{k \mid A[i,k] \leq B[k,j]\}|$

- From Matoušek 1991, Dominance Product for any two $n \times n$ matrices can be computed in $O(n^{(3+\omega)/2})$ time.

- From V.Vassilevska et al 2007, for two sparse matrices, with $m_1$ and $m_2$ valid elements respectively, the running time will be

$$O(\sqrt{m_1 m_2} n^{(\omega-1)/2})$$
Algorithm for Dominance Product?

- **Dominance Product**
  \[ M[i,j] = (A \odot B)[i,j] = |\{ k \mid A[i,k] \leq B[k,j] \}| \]
Algorithm for Dominance Product?

- **Dominance Product**
  
  - $M[i,j] = (A \diamond B)[i,j] = |\{ k \mid A[i,k] \leq B[k,j] \}|$
  
Algorithm for Dominance Product?

- **Dominance Product**
  - $M[i,j] = (A \bowtie B)[i,j] = |\{ k \mid A[i,k] \leq B[k,j] \}|$
  - Thus, elements of the $i$-th column of $A$ are only compared with $i$-th row of $B$. 
Algorithm for Dominance Product?

- **Dominance Product**
  - $M[i,j] = (A \diamond B)[i,j] = |\{k \mid A[i,k] \leq B[k,j]\}|$
  - Thus, elements of the $i$-th column of $A$ are only compared with $i$-th row of $B$. 

![Diagram showing comparison process between matrices A and B]
Algorithm for Dominance Product?

- **Dominance Product**
  - \( M[i,j] = (A \diamond B)[i,j] = |\{k \mid A[i,k] \leq B[k,j]\}| \)
  - \( A[i,1] \) compare with \( B[1,j] \).
  - Thus, elements of the \( i \)-th column of \( A \) are only compared with \( i \)-th row of \( B \).
  - Thus, define the set \( L_k = \{A[i,k]\} \cup \{B[k,j]\} \), and sort \( L_k \)
  - Partition the sorted list of each \( L_k \) into \( r \) parts:
    - each with \( 2n/r \) elements

![Diagram of partitioned list](image)
Construct:

\[ A_b[i,k] = \begin{cases} 
1 & \text{if } A[i,k] \text{ is in } b-\text{th part of } L_k \\
0 & \text{otherwise} 
\end{cases} \]

\[ B_b[k,j] = \begin{cases} 
1 & \text{if } B[k,j] \text{ is in } b-\text{th part of } L_k \\
0 & \text{otherwise} 
\end{cases} \]

if \( i < j \), \( A_i \cdot B_j \) is always included in the dominance product
For each $b=1 \ldots r$, compute the Boolean product $C_b = A_b \cdot (B_{b+1} + B_{b+2} + \ldots + B_r)$, then compute $\sum_b C_b$.

Since the corresponding elements in $b$-th part of $A$ are $\leq$ the $(b+1)$-th part of $B$

What’s left?

$M[i,j] = (A \cdot B)[i,j] = |\{k \mid A[i,k] \leq B[k,j]\}|$
For each $b=1 \ldots r$, compute the Boolean product 

$$C_b = A_b \cdot (B_{b+1} + B_{b+2} + \ldots + B_r),$$

then compute $\sum_b C_b$.

Then we only need to compare elements in the same parts of the partition.

Running Time?
For each $b=1 \ldots r$, compute the Boolean product $C_b = A_b \cdot (B_{b+1} + B_{b+2} + \ldots + B_r)$, then compute $\sum_b C_b$.

- Time: $O(r \cdot n^\omega)$

Then we only need to compare elements in the same parts of the partition.

- Time: for each part, $O(n \cdot (n/r)^2)$, in total: $O(rn \cdot (n/r)^2) = O(n^3/r)$
- To balance, we let $r = n^{(3-\omega)/2}$, total time: $O(n^{(3+\omega)/2})$
For each $b=1\ldots r$, compute the Boolean product $C_b = A_b \cdot (B_{b+1} + B_{b+2} + \ldots + B_r)$, then compute $\sum_b C_b$.

- **Time:** $O(r \cdot n^\omega)$

Then we only need to compare elements in the same parts of the partition.

- **Time:** for each part, $O(n \cdot (n/r)^2)$, in total: $O(rn \cdot (n/r)^2) = O(n^3/r)$
- **To balance, we let** $r = n^{(3-\omega)/2}$, **total time:** $O(n^{(3+\omega)/2})$

What if $A$ only contains $m$ valid elements?
- **other elements of $A$ are $+\infty$**
Sparse dominance product

- What if A only contains m valid elements?
  - other elements of A are $+\infty$

- For each $b=1 \ldots r$, compute the Boolean product $C_b = A_b \cdot (B_{b+1} + B_{b+2} + \ldots + B_r)$, then compute $\sum_b C_b$.
  - Time: $O(r \cdot n^\omega)$

- Then we only need to compare elements in the same parts of the partition
  - Every element in A needs to compare with $O(n/r)$ elements of $B_b$ for some $b$, in total: $O(mn/r + n^2)$
  - total time: $O(m^{1/2} \cdot n^{(1+\omega)/2})$
Bottleneck paths (Widest paths)

- Here “Bottleneck Paths” between $s$ and $t$ is the path with maximum flow from $s$ to $t$.
- For a graph $G=(V,E)$, and the capacity function for every edge: $w: E \rightarrow \mathbb{R}$, the maximum flow is defined as:

$$f(s, t) = \max_{p: \text{path } s \rightarrow t} \min_{e \in p} w(e)$$
Max-min Product

- **Max-Min Product** $C = A \circ B$
  - $C[i,j] = \max_k \min\{A[i,k], B[k,j]\}$
  - which is just the bottleneck path in 3-layer graph:
Max-min product and All-pair Bottleneck Paths

- Since a path can at most be composed of n edges, so the flows of the all-pair bottleneck paths is equal to
  - $G \circ G \circ \ldots \circ G = G^n = ((G^2)^2 \ldots)^2$
    - n times
    - square about log n times

- In fact, the time complexity for APBP is the same as the max-min product.
Another way to think of max-min

- Since in the result,
  - \( C[i,j] = \max_{1 \leq k \leq n} \{\min\{A[i,k], B[k,j]\}\} \)
  - So if \( A[i,k] \leq B[k,j], \min\{A[i,k], B[k,j]\} = A[i,k], \) otherwise it is \( B[k,j] \)
- Thus, we just need to compute:
Another way to think of max-min

- Since in the result,
  - \( C[i,j] = \max_{1 \leq k \leq n} \{\min\{A[i,k], B[k,j]\}\} \)
  - So if \( A[i,k] \leq B[k,j] \), \( \min\{A[i,k], B[k,j]\} = A[i,k] \), otherwise it is \( B[k,j] \)
- Thus, we just need to compute:
  - \( D[i,j] = \max\{A[i,k] | A[i,k] \leq B[k,j]\} \)
  - \( D'[i,j] = \max\{B[k,j] | A[i,k] > B[k,j]\} \)
  - And, \( C[i,j] = \max\{D[i,j], D'[i,j]\} \)
- We only consider the algorithm for \( D[i,j] \) here.
A Example To Show Dominance and Max-Min Product

- Consider the following example:
A Example To Show Dominance and Max-Min Product

\[ M[i,j] = 3 \text{ (Dominance Product)} \]

\[ D[i,j] = 7 \]

\[ D[i,j] = \max\{A[i,k] | A[i,k] \leq B[k,j]\} \]
Algorithm for max-min product

- Remember we want to compute:
  - $D[i,j] = \max\{A[i,k] | A[i,k] \leq B[k,j]\}$

- [Vassilevska, Williams, Yuster 2007] We sort the elements of A, and partition every row into s parts.
Algorithm for max-min product

- Remember we want to compute:
  - \( D[i,j] = \max\{A[i,k] \mid A[i,k] \leq B[k,j]\} \)

- [Vassilevska, Williams, Yuster 2007] We sort the elements of A, and partition every row into \( s \) parts, for \( b=1,\ldots,s \):
  - Then \( A_b[i,j] = \begin{cases} 1 & \text{if } A[i,j] \text{ is in } b\text{-th part row } i \\ 0 & \text{otherwise} \end{cases} \)
The matrix $A$ is then partitioned into submatrices $A_1, A_2, A_3, \ldots, A_s$

Compute the dominance products $A_1 \blacklozenge B, A_2 \blacklozenge B, A_3 \blacklozenge B, \ldots, A_s \blacklozenge B$
Just as in this graph:

For any \((i,j)\), we intend to find:

\[
D_{i,j} = \max \{ A_{i,k} \mid A_{i,k} \leq B_{k,j} \}
\]

First, we find which part \(D_{i,j}\) is in.
From the Dominance Products:

- If \((A_p \Diamond B)_{i,j} > 0\),
  - Then there exists a \(A_{i,k}\) in \(A_p\) such that \(A_{i,k} \leq B_{k,j}\), so \(D_{i,j}\) is in \(A_p\).
From the Dominance Products:

- If $(A_p \diamondsuit B)_{i,j} = 0$ and $(A_{p-1} \diamondsuit B)_{i,j} > 0$
  - Then there exists an $A_{i,k}$ in $A_{p-1}$ such that $A_{i,k} \leq B_{k,j}$, so $D_{i,j}$ is in $A_{p-1}$. 

![Diagram](image-url)
From the Dominance Products:

- So in general we just find the maximum part $q$ in which $(A_q \blacklozenge B)_{i,j} > 0$, which means $D_{i,j}$ is in part $q$
- Then check the elements of part $q$ in line $i$ of $A$ one by one.
Algorithm for max-min product

- Remember we want to compute:
  \[ D[i,j] = \max \{ A[i,k] \mid A[i,k] \leq B[k,j] \} \]
- \[Vassilevska, Williams, Yuster 2007\] We sort the elements of A, and partition every row into \( s \) parts.
  - Then \( A_b[i,j] = \begin{cases} 1 & \text{if } A[i,j] \text{ is in } b\text{-th part row } i \\ 0 & \text{otherwise} \end{cases} \)
  - Compute the dominance product \( A_b \cdot B \) for \( b=1\ldots s \)
  - Then for each pair of \((i,j)\), find the maximum \( b \) such that \( A_b \cdot B[i,j] > 0 \), then \( D[i,j] \) is in the \( b\)-th part of row \( i \) of A.
Algorithm for max-min product

- Remember we want to compute:
  - $D[i,j] = \max\{A[i,k] | A[i,k] \leq B[k,j]\}$

- [Vassilevska, Williams, Yuster 2007] We sort the elements of $A$, and partition every row into $s$ parts

- Compute the dominance product $A_b \cdot B$ for $b=1\ldots s$
  - Time: $O(m^{1/2} \cdot n^{(1+\omega)/2})$ for each sparse dominance product where $m=n^2/s$

- Then for each pair of $(i,j)$, find the maximum $b$ such that $A_b \cdot B[i,j] > 0$, then $D[i,j]$ is in the $b$-th part of row $i$ of $A$
  - Time: $O(n/s)$ for each $i,j$

- Total time: $O(n^2 + \omega/3)$
# Previous Results

<table>
<thead>
<tr>
<th>Algorithms for bottleneck paths</th>
<th>Running Time</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Edge-weighted</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Single-source</td>
<td>$O(m+n \log n)$</td>
<td></td>
</tr>
<tr>
<td>Single-source, single-destination</td>
<td>$O(m \log^* n)$</td>
<td>Gabow &amp; Tarjan</td>
</tr>
<tr>
<td>All-pair (trivial)</td>
<td>$O(mn)$</td>
<td></td>
</tr>
<tr>
<td>All-pair (2007)</td>
<td>$O(n^{2+\omega/3}) \approx O(n^{2.792})$</td>
<td>Vassilevska, Williams and Yuster</td>
</tr>
<tr>
<td>All-pair (2009)</td>
<td>$O(n^{(3+\omega)/2}) \approx O(n^{2.688})$</td>
<td>Duan, Pettie</td>
</tr>
<tr>
<td><strong>Vertex-weighted</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>All-pair (2007)</td>
<td>$O(n^\mu) \approx O(n^{2.575})$</td>
<td>Shapira, Yuster and Zwick</td>
</tr>
</tbody>
</table>

\[ \omega \approx 2.376 \quad \text{(3+\omega)/2} \quad 2+\omega/3 \quad 3 \]
Summary

- **Algorithms for matrix operations of bound** $O(n^{2.373})$:
  - matrix multiplication
  - inversion
  - LUP-decomposition
  - solving linear equations
  - determinant
  - **but not eigenvalues!**

- **Applications:**
  - Perfect matching in bipartite graphs
  - 4-node subgraphs
  - dominance product / (max,min)-product