This class is about matrix multiplication and how it can be applied to graph algorithms. We will also consider faster approximation algorithms that solve problems without resorting to matrix multiplication technique.

1 Prior work on matrix multiplication

Definition 1.1. *(Matrix multiplication) Let A and B be n-by-n matrices with entries over a field K. Then the product C, where $AB = C$ is an n-by-n matrix defined by $C[i,j] = \sum_{k=1}^{n} A(i,k) \cdot B(k,j)$. Here $+$ and $\cdot$ are operations over K.*

If K is an arbitrary field, we will assume that addition and multiplication of field elements takes $O(1)$ time. Later on, if K is the field of rationals, we will assume that operations (addition and multiplication) on $O(\log n)$ bit numbers takes $O(1)$ time, i.e. we'll be working in a word-RAM model of computation with word size $O(\log n)$.

There has been much effort to improve the runtime of matrix multiplication. The trivial algorithm follows the definition and multiplies $n \times n$ matrices in $O(n^3)$ time. Strassen (1969) surprised everyone by giving an $O(n^{2.81})$ time algorithm. This began a long line of improvements until in 1986, Coppersmith and Winograd achieved $O(n^{2.376})$. After 24 years of no progress, in 2010 Andrew Stothers, a graduate student in Edinburgh, improved the running time to $O(n^{2.374})$. In 2011, Virginia Vassilevska Williams got $O(n^{2.37288})$, which was the best bound until Le Gall got $O(n^{2.37287})$ in 2014. This bound was only recently (in 2021) improved by Alman and Vassilevska Williams to $O(n^{2.37286})$. Many believe that the ultimate bound will be $n^{2+o(1)}$, but this has yet to be proven. There are no nontrivial lower bounds except in specialized models of computation.

Matrix multiplication is an extremely useful tool that can be used to solve systems of linear equations and solve linear programs among many other linear algebraic and graph problems.

Today we'll discuss the relationship between the problems of matrix inversion and matrix multiplication, and also that between Boolean matrix multiplication and triangle detection.

2 Matrix multiplication is equivalent to matrix inversion

Matrix inversion is important because it is used to solve linear systems of equations. Multiplication is equivalent to inversion, in the sense that any multiplication algorithm can be used to obtain an inversion algorithm with similar runtime, and vice versa.

2.1 Multiplication can be reduced to inversion

The following theorem is due to Winograd (1970) and it holds over arbitrary fields.

Theorem 2.1. *If one can invert a nonsingular n-by-n matrix in $T(n)$ time, then one can multiply n-by-n matrices in $O(T(3n))$ time.*

Proof. Let A and B be matrices. Consider the following $3n \times 3n$ matrix:

$$D = \begin{bmatrix} I & A & 0 \\ 0 & I & B \\ 0 & 0 & I \end{bmatrix}$$

where I is the n-by-n identity matrix. One can verify by direct calculation that

$$D^{-1} = \begin{bmatrix} I & -A & AB \\ 0 & I & -B \\ 0 & 0 & I \end{bmatrix}$$

Inverting D takes $O(T(3n))$ time and we can find AB by inverting D. Note that D is always invertible since its determinant is 1. $\square$

2.2 Inversion can be reduced to multiplication

Here we will show that using an algorithm for matrix multiplication we can get an algorithm to invert matrices. The proof will work for matrices over the **reals**. It can be extended for matrices over the complex numbers by looking the natural extensions of the properties we want for complex matrices (e.g. symmetric becomes Hermitian etc.).

Theorem 2.2. *Let $T(n)$ be such that $T(2n) \geq (2+\varepsilon)T(n)$ for some $\varepsilon > 0$ and all n. If one can multiply n-by-n matrices in $T(n)$ time, then one can invert n-by-n matrices in $O(T(n))$ time.*

Notice that since $T(n) \geq \Omega(n^2)$, we have that $T(n) = n^2 \cdot f(n)$. If $f(n)$ is nondecreasing, e.g. if $T(n)$ is of the form n^w, then we actually have that $T(2n) \geq 4T(n)$ and the above condition $T(2n) \geq (2+\varepsilon)T(n)$ is satistied for $\varepsilon = 2$. This is since $T(2n) = (2n)^2 f(2n) \geq 4n^2 f(n) = 4T(n)$ as f is nondecreasing. Thus for most natural functions $T(n)$, the condition holds easily.

2.2.1 Proof outline

First, we give an algorithm to invert *symmetric positive definite* (spd) matrices. Then we use this to invert arbitrary invertible matrices. That is, we will prove the following claim:

Claim 1. *If we can invert spd matrices in $T(n)$ time and can do matrix multiplication in $O(T(n))$ time, then we can invert any invertible matrix in $O(T(n))$ time.*

First, we define an spd matrix.

Definition 2.1. *A matrix A is symmetric positive definite (spd) if*

1. *A is symmetric, i.e. $A = A^t$, so $A(i,j) = A(j,i)$ for all i, j*
2. *A is positive definite, i.e. for all $x \neq 0$, $x^t A x > 0$.*

We prove Claim 1 in 4 steps:

1. Compute $A^t A$ using matrix multiplication.
2. Show that for every invertible matrix A, $A^t A$ is spd (Claim 4 below).
3. Use our spd inversion algorithm to compute $(A^t A)^{-1}$.
4. Compute $(A^t A)^{-1} A^t = A^{-1}(A^t)^{-1} A^t = A^{-1}$ using matrix multiplication.

2.2.2 Properties of symmetric positive definite matrices

In this section we will prove some properties of spd matrices that are useful for both step 2 above and the next section where we reduce inversion of spd matrices to matrix multiplication.

Claim 2. *All symmetric positive definite matrices are invertible.*

Proof. Suppose that A is not invertible. Then there exists a nonzero vector x such that $Ax = 0$. But then $x^t Ax = 0$ and A is not symmetric positive definite. So we conclude that all symmetric positive definite matrices are invertible. $\square$

Claim 3. *Any principal submatrix of a symmetric positive definite matrix is symmetric positive definite. (An m-by-m matrix M is a principal submatrix of an n-by-n matrix A if M is obtained from A by removing its last $n - m$ rows and columns.)*

Proof. Let x be a vector with m entries. We need to show that $x^t Mx > 0$. Consider y, which is x padded with $n-m$ trailing zeros. Since A is symmetric positive definite, $y^t Ay > 0$. But $y^t Ay = x^t Mx$, since all but the first m entries are zero. $\square$

Claim 4. *For any invertible matrix A, $A^t A$ is symmetric positive definite.*

Proof. Let x be a nonzero vector. Consider $x^t(A^t A)x = (Ax)^t(Ax) = ||Ax||^2 \geq 0$. We now show $||Ax||^2 > 0$. For any $x \neq 0$, Ax is nonzero, since A is invertible. Thus, $||Ax||^2 > 0$ for any $x \neq 0$. So $A^t A$ is positive definite. Furthermore, it's symmetric since $(A^t A)^t = A^t A$. $\square$

Claim 5. *Let n be even and let A be an $n \times n$ symmetric positive definite matrix. Divide A into four square blocks (each one $n/2$ by $n/2$):*

$$A = \begin{bmatrix} M & B^t \\ B & C \end{bmatrix}.$$

Then the Schur complement, $S = C - BM^{-1}B^t$, is symmetric positive definite.

The proof of the above claim will be in the homework.

Note: We can invert an $n \times n$ matrix for any n, using a matrix inversion algorithm that only works when n is a power of 2. To do this, simply let N be the smallest power of 2 that is at most n, let I be the $(N-n) \times (N-n)$ identity matrix and invert the following matrix: $\begin{bmatrix} A & 0 \\ 0 & I \end{bmatrix}$.

This completes the proof of Claim 1.

2.2.3 Reduction for symmetric positive definite matrices

We will show that we can invert an spd matrix using matrix multiplication, which completes the proof of Theorem 2.1

Let A be an n by n symmetric positive definite matrix. Divide A into the $n/2$ by $n/2$ blocks M, B^t, B, and C. Again, let $S = C - BM^{-1}B^t$. By direct computation, we can verify that

$$A^{-1} = \begin{bmatrix} M^{-1} + M^{-1}B^t S^{-1} BM^{-1} & -M^{-1}B^t S^{-1} \\ -S^{-1}BM^{-1} & S^{-1} \end{bmatrix}$$

Therefore, we can compute A^{-1} recursively, as follows: (let the runtime be $t(n)$)

Algorithm 1: Inverting a symmetric positive definite matrix A

If $n = 1$, return $1/A$ as A is a scalar.
Compute M^{-1} recursively (we know M is invertible since A is spd, this takes $t(n/2)$ time)
Compute $S = C - BM^{-1}B^t$ using matrix multiplication (this takes $O(T(n))$ time)
Compute S^{-1} recursively (we know S is invertible since A is spd, this takes $t(n/2)$ time)
Compute all entries of A^{-1} (this takes $O(T(n))$ time)

The total runtime of the procedure is

$$t(n) \leq 2t(n/2) + O(T(n)) \leq O(\sum_j 2^j T(n/2^j)).$$

If $T(2n) \geq (2+\varepsilon)T(n)$ for some $\varepsilon > 0$, then

$$t(n) \leq O(\sum_j (2/(2+\varepsilon))^j T(n)) \leq O(T(n)).$$

3 Boolean Matrix Multiplication

Given two $n \times n$ matrices A, B over $\{0, 1\}$, we define Boolean Matrix Multiplication (BMM) as the following:

$$(AB)[i,j] = \bigvee_k (A(i,k) \wedge B(k,j))$$

Note that BMM can be computed using an algorithm for integer matrix multiplication, and so we have that BMM for $n \times n$ matrices is in $n^{\omega+o(1)}$ time, where $\omega < 2.373$ (the current bound for integer matrix multiplication).

Most theoretically fast matrix multiplication algorithms are impractical. Therefore, so called "combinatorial algorithms" are desirable. "Combinatorial algorithm" is loosely defined, but one has the following properties:

- Doesn't use subtraction
- All operations are relatively practical (like a lookup tables)

Remark 1. *No $O(n^{3-\varepsilon})$ time combinatorial algorithms for matrix multiplication are known for $\varepsilon > 0$, even for BMM! Such an algorithm would be known as "truly subcubic."*

Next lecture we will see some slightly subcubic combinatorial algorithms for BMM and some relationships between BMM and graph problems such as triangle detection.