Determinant is in NC

Csanky's algorithm and Pippenger's algorithm

Irish Debbarma, Upamanyu Yeddanapudi

November 22, 2022

- Naive algorithm (Laplace expansion) takes $O(n!)$.
- Gaussian elimination takes $O(n^3)$;
- faster algorithms exists (Strassen- $O(n^{2.807})$) with at best $O(n^{2.376})$ by Coppersmith-Winograd time.
- However, these algorithms cannot be efficiently performed in parallel.

Theorem (Csanky, 1976)

Let $A = (a_{ij})_{n \times n}$ be a matrix with a_{ij} a m-bit integer. Then we can compute the following in $(\log mn)^{O(1)}$ time using $(mn)^{O(1)}$ many processors.

- *characteristic polynomial of a A $(p_A(\lambda))$*
- *determinant of A $(\det A)$*
- *inverse of A (A^{-1})*

Take two vectors $(a_1, \ldots, a_n)$ and $(b_1, \ldots, b_n)$. The inner product is computed by first multiplying

$$a_i \cdot b_i, i = 1, 2, \ldots, n$$

and then adding the products in tree like fashion. Thus, the entire process is done in $O(\log n)$ parallel steps using $O(n)$ arithmetic processors.

Take two matrices $A_{m \times n}, B_{n \times p}$. Say c_{ij} is an element of AB. Then, $c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$ which is basically the inner product of

$$(a_{i1}, \ldots, a_{in}) \text{ and } (b_{1j}, \ldots, b_{nj})$$

By previous slide, this can be done in $O(\log n)$ steps with $O(n)$ processors.
And, this has to be done for each c_{ij} so we will have $O(mnp)$ processors running in parallel followed by addition (appropriately) in the subsequent steps in $O(\log n)$ time.

Take the $n \times n$ matrix A. From previous slide, $A \cdot A = A^2$ is in NC. So, to compute A^k, we can just perform repeated squaring (atmost $O(\log k)$ many multiplications) and since matrix multiplication is in NC, this process is also in NC.

The process to compute all powers of A, i.e., $A, A^2, A^3, \cdots, A^n$ can be done in $O((\log n)^2)$ time with $O(n^4)$ processors.
In the 0-th stage we have A. In the 1-st stage we will compute $A \cdot A = A^2$. In the 2-nd stage we will compute A^3, A^4 and in the 3-rd we will compute A^5, A^6, A^7, A^8. So, to compute all n powers we will take $O(\log n)$ time and generation of each power is a matrix multiplication that is done in $O(\log n)$ time. So the total time taken is $O((\log n)^2)$. And, we have total of $n-1$ multiplications so the number of processors required is $O(n^4)$.

Look at the lower triangular $n \times n$ matrix A. A can be written in the following manner:

$$\left(\begin{array}{c|c} \mathbf{B} & \mathbf{0} \\ \hline \mathbf{C} & \mathbf{D} \end{array}\right)$$

where $\mathbf{B}$ is $\lfloor n/2 \rfloor \times \lfloor n/2 \rfloor$ matrix, $\mathbf{C}$ is $\lceil n/2 \rceil \times \lfloor n/2 \rfloor$ matrix and $\mathbf{D}$ is $\lceil n/2 \rceil \times \lceil n/2 \rceil$ matrix.
Compute $\mathbf{B}^{-1}$ and $\mathbf{D}^{-1}$ recursively and get

$$\mathbf{A}^{-1} = \left(\begin{array}{c|c} \mathbf{B}^{-1} & \mathbf{0} \\ \hline -\mathbf{D}^{-1}\mathbf{C}\mathbf{B}^{-1} & \mathbf{D}^{-1} \end{array}\right)$$

The computation time of this algorithm follows the relation:

$$T(n) = T(n/2) + 2M(n/2)$$

where $T(n/2)$ is the time needed to invert B and D in parallel and $2M(n/2)$ corresponds to matrix multiplication $\mathbf{D}^{-1}\mathbf{C}\mathbf{B}^{-1}$. From previous calculations we know that $M(n) = O(\log n)$ and therefore $T(n) = O((\log n)^2)$.

Consider the general linear recurrence:

$$
\begin{aligned}
x_1 &= c_1 \\
x_2 &= a_{21}x_1 + c_2 \\
x_3 &= a_{31}x_1 + a_{32}x_2 + c_3 \\
&\vdots \\
x_n &= a_{n1}x_1 + a_{n2}x_2 + \ldots + c_n
\end{aligned}
$$

We can write this in matrix form as $Ax + c = x$ where A is the matrix with $a_{ij} = 0 \; \forall j \geq i$ and a_{ij} as in the relations for $i < j$, $c = (c_1 \; c_2 \; \ldots c_n)^T, x = (x_1 \ldots x_n)^T$.

Observe that $Ax + c = x \Rightarrow x(I - A) = c$ where $I - A$ is a lower triangular matrix and from previous result we know that finding inverse of a lower triangular matrix is in NC.
This enables us to find the inverse of $I - A$ giving

$$x = (I - A)^{-1}c$$

- $A = (a_{ij})_{i,j=1}^{n}$ a square matrix.
- $\lambda_1, \lambda_2, \ldots, \lambda_n$ be the eigenvalues of A.
- $\mathrm{Tr}(A) = \sum_{i=1}^{n} \lambda_i = \sum_{i=1}^{n} a_{ii}$ and $\det(A) = \prod_{i=1}^{n} \lambda_i$
- $\lambda_i^k, i = 1, 2, \ldots, n$ are eigenvalues of A^k
- $P_k := \sum_{i=1}^{n} \lambda_i^k$. Clearly, $P_i = \mathrm{Tr}(A^i)$
- $p_A(x) := \det(xI - A) = x^n + s_1 x^{n-1} + \cdots + s_n = \prod_{i=1}^{n} (x - \lambda_i)$

$$E_1 := \sum_{i=1}^{n} \lambda_i$$

$$E_2 := \sum_{i<j} \lambda_i \lambda_j$$

$$E_3 := \sum_{i<j<k} \lambda_i \lambda_j \lambda_k$$

$$\vdots$$

$$E_n := \lambda_1 \cdots \lambda_n$$

Clearly, $s_i = (-1)^i E_i$.

Using definition of E_i, P_i, we get the following recurrence relation.

$$\begin{aligned}
E_1 &= P_1 \\
2E_2 &= (-1)P_2 + P_1E_1 \\
3E_3 &= (+1)P_3 - P_2E_1 + P_1E_2 \\
&\vdots \\
kE_k &= (-1)^{k-1}(P_k - P_{k-1}E_1 + P_{k-2}E_2 - \cdots + P_1E_{k-1}) \\
E_k &= \frac{(-1)^{k-1}(P_k - P_{k-1}E_1 + P_{k-2}E_2 - \cdots + P_1E_{k-1})}{k}
\end{aligned}$$

These recurrence relations can be written in the following matrix form:

$$\begin{pmatrix} E_1 \\ E_2 \\ E_3 \\ \vdots \\ E_n \end{pmatrix} = \begin{pmatrix} P_1 \\ -P_2/2 \\ P_3/3 \\ \vdots \\ (-1)^n P_n/n \end{pmatrix} + \begin{pmatrix} 0 & 0 & \cdots & 0 \\ P_1/2 & 0 & \cdots & 0 \\ -P_2/3 & P_1/3 & \cdots & 0 \\ \vdots & \vdots & \ddots & 0 \\ (-1)^{n-1}P_{n-1}/n & (-1)^{n-2}P_{n-2}/n & \cdots & 0 \end{pmatrix} \begin{pmatrix} E_1 \\ E_2 \\ E_3 \\ \vdots \\ E_n \end{pmatrix}$$

More compactly,

$$E = P + AE \Rightarrow P = (I - A)E = ME$$

where $M = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ -P_1/2 & 1 & \cdots & 0 \\ P_2/3 & -P_1/3 & \cdots & 0 \\ \vdots & \vdots & \ddots & 0 \\ (-1)^n P_{n-1}/n & (-1)^{n-1} P_{n-2}/n & \cdots & 1 \end{pmatrix}$

M is lower triangular and thus computing inverse is in NC. This allows us to solve $E = M^{-1}P$.

- Since we have shown that solving recurrence relations is in NC, therefore finding E_i and consequently s_i is also in NC.
- So, finding the characteristic polynomial is also in NC.
- $\det(A) = (-1)^n s_n = E_n$ and computing E_n is in NC therefore computing $\det(A)$ is also in NC.

From Cayley-Hamilton theorem, we know that a square matrix satisifies its characteristic equation. Therefore,

$$p_A(A) = A^n + s_1 A^{n-1} + \cdots + s_n I = 0$$
$$s_n I = -(A^n + s_1 A^{n-1} + \cdots + s_1 A)$$
$$A^{-1} = -\frac{1}{s_n}(A^{n-1} + s_1 A^{n-2} + \cdots + s_1 I)$$

The coefficients s_k and A^k are computed as stated previously.

- Computing s_k is a NC process and so is computing A^k. So, we can compute s_k's and A_k's in parallel and then compute each entry of A^{-1} in parallel, thereby making the whole process in NC.

1. First, we computed inverse of a lower triangular matrix.

1. First, we computed inverse of a lower triangular matrix.
2. Then, we find complexity of solving linear recurrence.

1. First, we computed inverse of a lower triangular matrix.
2. Then, we find complexity of solving linear recurrence.
3. We then compute the characteristic polynomial of the matrix using Newton identities (basically a recurrence relation for coefficients of characteristic polynomial).

1. First, we computed inverse of a lower triangular matrix.
2. Then, we find complexity of solving linear recurrence.
3. We then compute the characteristic polynomial of the matrix using Newton identities (basically a recurrence relation for coefficients of characteristic polynomial).
4. Determinant of the matrix then comes for free.

1. First, we computed inverse of a lower triangular matrix.
2. Then, we find complexity of solving linear recurrence.
3. We then compute the characteristic polynomial of the matrix using Newton identities (basically a recurrence relation for coefficients of characteristic polynomial).
4. Determinant of the matrix then comes for free.
5. Finally, we compute the inverse of a GENERAL matrix.

1. First, we computed inverse of a lower triangular matrix.
2. Then, we find complexity of solving linear recurrence.
3. We then compute the characteristic polynomial of the matrix using Newton identities (basically a recurrence relation for coefficients of characteristic polynomial).
4. Determinant of the matrix then comes for free.
5. Finally, we compute the inverse of a GENERAL matrix.

Define

$$E_k^m := \sum_{1 \leq i_1 < \cdots < i_k \leq n, j \notin \{i_1, \ldots, i_k\}} \lambda_{i_1} \lambda_{i_2} \cdots \lambda_{i_k} \lambda_j^m$$

At the extremes

$$E_k^0 = (n-k)E_k, E_0^m = \mathrm{Tr}(A^m)$$

$$
\begin{aligned}
E_k \cdot \mathrm{Tr}(A^m) &= \left(\sum_{1 \le i_1 < \cdots < i_k \le n} \lambda_{i_1} \lambda_{i_2} \cdots \lambda_{i_k} \right) \left(\sum_{j=1}^{n} \lambda_j^m \right) \\
&= \sum_{1 \le i_1 < \cdots < i_k \le n, j \notin \{i_1, \ldots, i_k\}} \lambda_{i_1} \lambda_{i_2} \cdots \lambda_{i_k} \lambda_j^m \\
&+ \sum_{1 \le i_1 < \cdots < i_k \le n, j \in \{i_1, \ldots, i_k\}} \lambda_{i_1} \lambda_{i_2} \cdots \lambda_{i_k} \lambda_j^m \\
&= E_k^m + E_{k-1}^{m+1}
\end{aligned}
$$

$$
\begin{aligned}
& E_k \mathrm{Tr}(A^0) - E_{k-1}\mathrm{Tr}(A^1) + \cdots \pm E_1 \mathrm{Tr}(A^{k-1}) \mp \mathrm{Tr}(A^k) \\
& = (E_k^0 + E_{k-1}^1) - (E_{k-1}^1 + E_{k-2}^2) + \cdots \pm (E_1^{k-1} + E_0^k) \mp (E_0^k) \\
& = E_k^0 \\
& = (n-k)E_k \\
\therefore E_k & = \frac{1}{k}(E_{k-1}\mathrm{Tr}(A^1) - \cdots \mp E_1 \mathrm{Tr}(A^{k-1}) \pm \mathrm{Tr}(A^k))
\end{aligned}
$$

We will show that we can add two n-bit integers (a, b) in $O(\log n)$ time using n processors.

- First, we shall construct a string u to record the carry over.
- Then, we will add the two numbers and u bitwise using exclusive OR.

1. Start u with a 0 in the first position.
2. If the i-th bits of a, b are 0, then $i+1$-th bit of u will be 0 irrespective of the i-th bit.
3. If the i-th bits of a, b are 1, then $i+1$-th bit of u will be 1 irrespective of the i-th bit.
4. If the i-th bits of a, b are $0, 1$, then $i+1$-th bit of u will be same as i-th bit of u. We will say that the carry over has been "propagated".

We will represent u using three things $0, 1, p$ and encode the procedure in the following table:

$\cdot$	0	1	p
0	0	0	0
1	1	1	1
p	0	1	p

Notice that the string u can be constructed from a, b in constant time using n processors.

For example:

Let $a = 100101011101011$, $b = 110101001010001$. Then

$$\begin{aligned} u &= 1p01010p1ppp0p10 \\ carry &= 1001010110000110 \\ a &= \ \ 100101011101011 \\ b &= \ \ 110101001010001 \\ \hline sum &= 1011010100111100 \end{aligned}$$

The $\log n$ complexity comes from converting u to $carry$.

We shall show that multiplication of n bit integers can be done in $O((\log n)^2)$ time using $O(n^2)$ processors.

```
          101101
        × 101011
----------------
          101101
         101101
        000000
       101101
      000000
    + 101101
----------------
      11110001111
```

- Notice that there are n partial sums which can be computed using n^2 processors.
- Then, we can use our previous algorithm to add them up. It will take $O((\log n)^2)$ time.
- Since, addition was an NC process, so is multiplication.

- Csanky's algorithm involves a division by k for every $k \leq n$, so it works only on fields with characteristic 0 or $> n$.
- Other algorithms, like those of Berkowitz or Chistov, work over arbitrary fields.

- Csanky's algorithm involves a division by k for every $k \leq n$, so it works only on fields with characteristic 0 or $> n$.
- Other algorithms, like those of Berkowitz or Chistov, work over arbitrary fields.

Theorem [Pippenger, 2022]

We can compute det of an $n \times n$ matrix using a circuit of size $O(n^4 \log n)$ and depth $O((\log n)^2)$, over *any commutative ring*.

The formula

$$\det A = (-1)^n [x^n] \sum_{0 \le i \le n} \left(1 - \prod_{k \in [n]} \sum_{0 \le j \le n} \left((A_k)^j \right)_{k,k} x^j \right)^i ,$$

where

- $[x^n]$ denotes the coefficient of x^n in the given expression
- A_k is upper-left $k \times k$ submatrix of A
- $M_{i,j}$ denotes the i,jth entry of M.

Let R be a commutative ring.

$R[[x]]$, the ring of formal power series over R, is the set of all elements of the form $p(x) = \sum_{i \geq 0} p_i x^i$.

Let R be a commutative ring.
$R[[x]]$, the ring of formal power series over R, is the set of all elements of the form $p(x) = \sum_{i \geq 0} p_i x^i$.

Multiplication is defined in the usual way:
$r(x) := p(x)q(x) = (p_0 + p_1 x + p_2 x^2 + \dots)(q_0 + q_1 x + q_2 x^2 + \dots)$,
where $r_k = p_0 q_k + p_1 q_{k-1} + \cdots + p_k q_0 = \sum_{0 \leq i \leq k} p_i q_{k-i}$.

Let R be a commutative ring.
$R[[x]]$, the ring of formal power series over R, is the set of all elements of the form $p(x) = \sum_{i \geq 0} p_i x^i$.

Multiplication is defined in the usual way:
$r(x) := p(x)q(x) = (p_0 + p_1 x + p_2 x^2 + \dots)(q_0 + q_1 x + q_2 x^2 + \dots)$,
where $r_k = p_0 q_k + p_1 q_{k-1} + \cdots + p_k q_0 = \sum_{0 \leq i \leq k} p_i q_{k-i}$.

If $p_0 = 1$, we call p to be *unic*.

Lemma

If p_0 is invertible in R, then $p(x)$ has an inverse in $R[[x]]$.

Lemma

If p_0 is invertible in R, then $p(x)$ has an inverse in $R[[x]]$.

Proof.

We want $q(x)$ such that $p(x)q(x) = 1$.
We define the coefficients of $q(x)$ inductively, by comparing coefficients on both sides.
Base case: $p_0 q_0 = 1$, so $q_0 = p_0^{-1}$.

Lemma

If p_0 is invertible in R, then $p(x)$ has an inverse in $R[[x]]$.

Proof.

We want $q(x)$ such that $p(x)q(x) = 1$.
We define the coefficients of $q(x)$ inductively, by comparing coefficients on both sides.
Base case: $p_0 q_0 = 1$, so $q_0 = p_0^{-1}$.
Inductive step: suppose the coefficients of q have been calculated upto q_{k-1}. Observe that $\sum_{0 \le i \le k} p_i q_{k-i} = 0$.

$$\implies q_k = -p_0^{-1} \sum_{1 \le i \le k} p_i q_{k-i}.$$

□

Lemma

If p_0 is invertible in R, then $p(x)$ has an inverse in $R[[x]]$.

Proof.

We want $q(x)$ such that $p(x)q(x) = 1$.
We define the coefficients of $q(x)$ inductively, by comparing coefficients on both sides.
Base case: $p_0 q_0 = 1$, so $q_0 = p_0^{-1}$.
Inductive step: suppose the coefficients of q have been calculated upto q_{k-1}. Observe that $\sum_{0 \leq i \leq k} p_i q_{k-i} = 0$.

$$\implies q_k = -p_0^{-1} \sum_{1 \leq i \leq k} p_i q_{k-i}.$$

□

In particular, a unic power series has an inverse.

We will only be dealing with polynomials of degrees $\leq n$, so we can assume that all coefficients beyond x^n are zero.
More formally, we can work in $R[[x]]_n := R[[x]]/\langle x^{n+1}\rangle$.

We will only be dealing with polynomials of degrees $\leq n$, so we can assume that all coefficients beyond x^n are zero.
More formally, we can work in $R[[x]]_n := R[[x]]/\langle x^{n+1}\rangle$.

Lemma

If $p(x) \in R[[x]]_n$ is unic, then its inverse is $\sum_{0\leq i\leq n} (q(x))^i$, where $p(x) = 1 - q(x)$.

Proof.

$p(x)\cdot p(x)^{-1} = (1 - q(x))\,(1 + q(x) + \cdots + q(x)^n)$ is a telescoping sum, and is equal to $1 - q(x)^{n+1}$.
The constant term of $q(x)$ is zero, so this is equal to 1. □

Let $c(x) = \sum_{0 \leq i \leq n} c_i x^i$ be the characteristic polynomial of A.

Let $c(x) = \sum_{0 \le i \le n} c_i x^i$ be the characteristic polynomial of A.

Observations:

- $c_n = 1$, i.e. $c(x)$ is monic.
- $\det A = (-1)^n c_0 = (-1)^n [x^0] c(x)$.

Let $c(x) = \sum_{0 \leq i \leq n} c_i x^i$ be the characteristic polynomial of A.

Observations:

- $c_n = 1$, i.e. $c(x)$ is monic.
- $\det A = (-1)^n c_0 = (-1)^n [x^0] c(x)$.

Define $d(x) := x^n c(x^{-1})$, i.e. $d(x) = \sum_{0 \leq i \leq n} c_{n-i} x^i$.

Let $c(x) = \sum_{0 \le i \le n} c_i x^i$ be the characteristic polynomial of A.

Observations:

- $c_n = 1$, i.e. $c(x)$ is monic.
- $\det A = (-1)^n c_0 = (-1)^n [x^0] c(x)$.

Define $d(x) := x^n c(x^{-1})$, i.e. $d(x) = \sum_{0 \le i \le n} c_{n-i} x^i$.

Observations:

- $d(x)$ is unic.
- $\det A = (-1)^n [x^n] d(x)$.

Let $c(x) = \sum_{0 \leq i \leq n} c_i x^i$ be the characteristic polynomial of A.

Observations:

- $c_n = 1$, i.e. $c(x)$ is monic.
- $\det A = (-1)^n c_0 = (-1)^n [x^0] c(x)$.

Define $d(x) := x^n c(x^{-1})$, i.e. $d(x) = \sum_{0 \leq i \leq n} c_{n-i} x^i$.

Observations:

- $d(x)$ is unic.
- $\det A = (-1)^n [x^n] d(x)$.
- $c(x) = \det(xI - A)$, so $d(x) = \det(I - xA)$.

Suppose $\det M$ is invertible, and let $M[j,i]$ be the matrix obtained by removing row j and column i from M.

Recall that $\left(M^{-1}\right)_{i,j} = (-1)^{i+j} \det M[j,i] \cdot (\det M)^{-1}$.

Suppose $\det M$ is invertible, and let $M[j,i]$ be the matrix obtained by removing row j and column i from M.

Recall that $\left(M^{-1}\right)_{i,j} = (-1)^{i+j} \det M[j,i] \cdot (\det M)^{-1}$. In particular,

$$\left((B_k)^{-1}\right)_{k,k} = \frac{\det B_{k-1}}{\det B_k}.$$

Suppose det M is invertible, and let $M[j,i]$ be the matrix obtained by removing row j and column i from M.

Recall that $\left(M^{-1}\right)_{i,j} = (-1)^{i+j} \det M[j,i] \cdot (\det M)^{-1}$. In particular,

$$\left((B_k)^{-1}\right)_{k,k} = \frac{\det B_{k-1}}{\det B_k}.$$

Multiplying over all k, we get

$$\prod_{k \in [n]} \left((B_k)^{-1}\right)_{k,k} = \frac{\det B_0}{\det B} = (\det B)^{-1}.$$

Suppose $\det M$ is invertible, and let $M[j,i]$ be the matrix obtained by removing row j and column i from M.

Recall that $\left(M^{-1}\right)_{i,j} = (-1)^{i+j} \det M[j,i] \cdot (\det M)^{-1}$.
In particular,

$$\left((B_k)^{-1}\right)_{k,k} = \frac{\det B_{k-1}}{\det B_k}.$$

Multiplying over all k, we get

$$\prod_{k\in[n]} \left((B_k)^{-1}\right)_{k,k} = \frac{\det B_0}{\det B} = (\det B)^{-1}.$$

This gives us $\det B = \left(\prod_{k\in[n]} \left((B_k)^{-1}\right)_{k,k}\right)^{-1}$

Suppose $\det M$ is invertible, and let $M[j,i]$ be the matrix obtained by removing row j and column i from M.

Recall that $\left(M^{-1}\right)_{i,j} = (-1)^{i+j} \det M[j,i] \cdot (\det M)^{-1}$. In particular,

$$\left((B_k)^{-1}\right)_{k,k} = \frac{\det B_{k-1}}{\det B_k}.$$

Multiplying over all k, we get

$$\prod_{k \in [n]} \left((B_k)^{-1}\right)_{k,k} = \frac{\det B_0}{\det B} = (\det B)^{-1}.$$

This gives us $\det B = \left(\prod_{k \in [n]} \left((B_k)^{-1}\right)_{k,k}\right)^{-1}$

Note that $\det(I - xA)$ is unic, so it is invertible.

$$\det A = (-1)^n [x^n] d(x).$$

$$\det A = (-1)^n [x^n] d(x).$$

$$d(x) = \det(I - xA)$$

$$\det A = (-1)^n [x^n] d(x).$$

$$d(x) = \det(I - xA) = \left(\prod_{k \in [n]} \left(((I - xA)_k)^{-1} \right)_{k,k} \right)^{-1}$$

$\det A = (-1)^n [x^n] d(x)$.

$$d(x) = \det(I - xA) = \left(\prod_{k \in [n]} \left(((I - xA)_k)^{-1} \right)_{k,k} \right)^{-1}$$

Now, $(I - xA)_k$ lies in $R^{k \times k}[[x]]_n$ (and is unic there). The subring of this generated by A is commutative, so we can invert the above formula there.

$$\det A = (-1)^n [x^n] d(x).$$

$$d(x) = \det(I - xA) = \left(\prod_{k \in [n]} \left(((I - xA)_k)^{-1} \right)_{k,k} \right)^{-1}$$

$$= \left(\prod_{k \in [n]} \left(\sum_{0 \le j \le n} (xA_k)^j \right)_{k,k} \right)^{-1} = \left(\prod_{k \in [n]} \sum_{0 \le j \le n} \left((A_k)^j \right)_{k,k} x^j \right)^{-1}$$

$\det A = (-1)^n [x^n] d(x)$.

$$d(x) = \det(I - xA) = \left(\prod_{k \in [n]} \left(((I - xA)_k)^{-1} \right)_{k,k} \right)^{-1}$$

$$= \left(\prod_{k \in [n]} \left(\sum_{0 \le j \le n} (xA_k)^j \right)_{k,k} \right)^{-1} = \left(\prod_{k \in [n]} \sum_{0 \le j \le n} \left((A_k)^j \right)_{k,k} x^j \right)^{-1}$$

Now each factor in the product is unic, so the product itself is unic and we can invert it.

$\det A = (-1)^n [x^n] d(x).$

$$d(x) = \det(I - xA) = \left(\prod_{k \in [n]} \left(((I - xA)_k)^{-1} \right)_{k,k} \right)^{-1}$$

$$= \left(\prod_{k \in [n]} \left(\sum_{0 \le j \le n} (xA_k)^j \right)_{k,k} \right)^{-1} = \left(\prod_{k \in [n]} \sum_{0 \le j \le n} \left((A_k)^j \right)_{k,k} x^j \right)^{-1}$$

Now each factor in the product is unic, so the product itself is unic and we can invert it.
Recall that $p(x)^{-1} = \sum_{0 \le i \le n} (1 - p(x))^i$. This gives us

$$\det A = (-1)^n [x^n] d(x).$$

$$d(x) = \det(I - xA) = \left(\prod_{k \in [n]} \left(((I - xA)_k)^{-1} \right)_{k,k} \right)^{-1}$$

$$= \left(\prod_{k \in [n]} \left(\sum_{0 \le j \le n} (xA_k)^j \right)_{k,k} \right)^{-1} = \left(\prod_{k \in [n]} \sum_{0 \le j \le n} \left((A_k)^j \right)_{k,k} x^j \right)^{-1}$$

$$= \sum_{0 \le i \le n} \left(1 - \prod_{k \in [n]} \sum_{0 \le j \le n} \left((A_k)^j \right)_{k,k} x^j \right)^i .$$

$$d(x) = \sum_{0 \le i \le n} \left(1 - \prod_{k \in [n]} \sum_{0 \le j \le n} \left((A_k)^j \right)_{k,k} x^j \right)^i .$$

- Recall that for a matrix A, finding $A^1, A^2, \ldots, A^n$ can be done with $O(n^4)$ operations in $O((\log n)^2)$ depth.

$$d(x) = \sum_{0\leq i\leq n} \left(1 - \prod_{k\in[n]} \sum_{0\leq j\leq n} \left((A_k)^j\right)_{k,k} x^j \right)^i .$$

- Recall that for a matrix A, finding $A^1, A^2, \ldots, A^n$ can be done with $O(n^4)$ operations in $O((\log n)^2)$ depth.
- Two polynomials p, q of degree $\leq n$ can be multiplied in $O(\log n)$ steps using $O(n^2)$ operations:

$$d(x) = \sum_{0\leq i\leq n}\left(1 - \prod_{k\in[n]}\sum_{0\leq j\leq n}\left((A_k)^j\right)_{k,k} x^j\right)^i .$$

- Recall that for a matrix A, finding $A^1, A^2, \ldots, A^n$ can be done with $O(n^4)$ operations in $O((\log n)^2)$ depth.
- Two polynomials p, q of degree $\leq n$ can be multiplied in $O(\log n)$ steps using $O(n^2)$ operations:
  we have $n+1$ coefficients; each of the form $\sum_{0\leq i\leq k} p_i q_{k-i}$.

$$d(x) = \sum_{0 \le i \le n} \left(1 - \prod_{k \in [n]} \sum_{0 \le j \le n} \left((A_k)^j \right)_{k,k} x^j \right)^i .$$

- Recall that for a matrix A, finding $A^1, A^2, \ldots, A^n$ can be done with $O(n^4)$ operations in $O((\log n)^2)$ depth.
- Two polynomials p, q of degree $\le n$ can be multiplied in $O(\log n)$ steps using $O(n^2)$ operations: we have $n + 1$ coefficients; each of the form $\sum_{0 \le i \le k} p_i q_{k-i}$.
- Our computation for $d(x)$ has three stages: finding $\left((A_k)^j \right)_{k,k}$ for every j and k, computing the product for each i, and finally taking ith powers and adding them.

$$d(x) = \sum_{0 \le i \le n} \left(1 - \prod_{k \in [n]} \sum_{0 \le j \le n} \left((A_k)^j \right)_{k,k} x^j \right)^i .$$

For a given A_k, computing all jth powers takes $O(n^4)$ operations in $O((\log n)^2)$ stages.

$$d(x) = \sum_{0 \leq i \leq n} \left(1 - \prod_{k \in [n]} \sum_{0 \leq j \leq n} \left((A_k)^j \right)_{k,k} x^j \right)^i .$$

For a given A_k, computing all jth powers takes $O(n^4)$ operations in $O((\log n)^2)$ stages. We do this for all A_k, so it takes us $O(n^5)$ operations.

$$d(x) = \sum_{0 \le i \le n} \left(1 - \prod_{k \in [n]} \sum_{0 \le j \le n} \left((A_k)^j \right)_{k,k} x^j \right)^i .$$

For a given A_k, computing all jth powers takes $O(n^4)$ operations in $O((\log n)^2)$ stages. We do this for all A_k, so it takes us $O(n^5)$ operations.

We now need to find

$$d(x) = \sum_{0 \le i \le n} \left(1 - \prod_{k \in [n]} p_k(x) \right)^i .$$

$$d(x) = \sum_{0 \le i \le n} \left(1 - \prod_{k \in [n]} p_k(x)\right)^i .$$

$$d(x) = \sum_{0 \le i \le n} \left(1 - \prod_{k \in [n]} p_k(x)\right)^i.$$

The product of n polynomials can be computed in $\log n$ stages, by multiplying pairs in a tree-like fashion.

$$d(x) = \sum_{0 \le i \le n} \left(1 - \prod_{k \in [n]} p_k(x)\right)^i .$$

The product of n polynomials can be computed in $\log n$ stages, by multiplying pairs in a tree-like fashion. Each multiplication takes $O(\log n)$ steps using $O(n^2)$ operations, so the total is $O((\log n)^2)$ steps and $O(n^3)$ operations.

$$d(x) = \sum_{0 \le i \le n} \left(1 - \prod_{k \in [n]} p_k(x) \right)^i .$$

The product of n polynomials can be computed in $\log n$ stages, by multiplying pairs in a tree-like fashion. Each multiplication takes $O(\log n)$ steps using $O(n^2)$ operations, so the total is $O((\log n)^2)$ steps and $O(n^3)$ operations.

Our computation is now of the form $d(x) = \sum_{0 \le i \le n} q(x)^i$.

$$d(x) = \sum_{0 \le i \le n} \left(1 - \prod_{k \in [n]} p_k(x) \right)^i .$$

The product of n polynomials can be computed in $\log n$ stages, by multiplying pairs in a tree-like fashion. Each multiplication takes $O(\log n)$ steps using $O(n^2)$ operations, so the total is $O((\log n)^2)$ steps and $O(n^3)$ operations.

Our computation is now of the form $d(x) = \sum_{0 \le i \le n} q(x)^i$. Similar to what we did for matrix powers, we do this in $\log n$ rounds, computing $q(x)^{2^k}$ to $q(x)^{2^{k+1}-1}$ in the kth round.

$$d(x) = \sum_{0 \leq i \leq n} \left(1 - \prod_{k \in [n]} p_k(x)\right)^i .$$

The product of n polynomials can be computed in $\log n$ stages, by multiplying pairs in a tree-like fashion. Each multiplication takes $O(\log n)$ steps using $O(n^2)$ operations, so the total is $O((\log n)^2)$ steps and $O(n^3)$ operations.

Our computation is now of the form $d(x) = \sum_{0 \leq i \leq n} q(x)^i$. Similar to what we did for matrix powers, we do this in $\log n$ rounds, computing $q(x)^{2^k}$ to $q(x)^{2^{k+1}-1}$ in the kth round. This also takes $O((logn)^2)$ steps and $O(n^3)$ operations.

$$d(x) = \sum_{0 \leq i \leq n} \left(1 - \prod_{k \in [n]} p_k(x)\right)^i .$$

The product of n polynomials can be computed in $\log n$ stages, by multiplying pairs in a tree-like fashion. Each multiplication takes $O(\log n)$ steps using $O(n^2)$ operations, so the total is $O((\log n)^2)$ steps and $O(n^3)$ operations.

Our computation is now of the form $d(x) = \sum_{0 \leq i \leq n} q(x)^i$. Similar to what we did for matrix powers, we do this in $\log n$ rounds, computing $q(x)^{2^k}$ to $q(x)^{2^{k+1}-1}$ in the kth round. This also takes $O((logn)^2)$ steps and $O(n^3)$ operations. Adding up these powers takes an additional $O(logn)$ steps.

For the number of operations ($O(n^5)$), the first step is the dominating one. It is also inefficient - we are computing a whole matrix just for one entry.

For the number of operations ($O(n^5)$), the first step is the dominating one. It is also inefficient - we are computing a whole matrix just for one entry.
We can improve on this:
Let $\mathbf{v}$ be the $1 \times n$ vector $\begin{pmatrix} 0 & 0 & \dots & 0 & 1 \end{pmatrix}$. Observe that for a matrix B, $\mathbf{v}B$ is the last row of B.

For the number of operations ($O(n^5)$), the first step is the dominating one. It is also inefficient - we are computing a whole matrix just for one entry.

We can improve on this:

Let $\mathbf{v}$ be the $1 \times n$ vector $\begin{pmatrix} 0 & 0 & \dots & 0 & 1 \end{pmatrix}$. Observe that for a matrix B, $\mathbf{v}B$ is the last row of B.

Given $B = A_k$, we find $((A_k)^j)_{k,k}$ for each j in $\log n$ rounds:

For the number of operations ($O(n^5)$), the first step is the dominating one. It is also inefficient - we are computing a whole matrix just for one entry.
We can improve on this:
Let $\mathbf{v}$ be the $1 \times n$ vector $\begin{pmatrix} 0 & 0 & \dots & 0 & 1 \end{pmatrix}$. Observe that for a matrix B, $\mathbf{v}B$ is the last row of B.

Given $B = A_k$, we find $((A_k)^j)_{k,k}$ for each j in $\log n$ rounds:

- First, compute $\mathbf{v}B$.
- In the first round, compute B^2 and $\mathbf{v}B^2$, then multiply $\mathbf{v}B \cdot B^2$ to get $\mathbf{v}B^3$.

For the number of operations ($O(n^5)$), the first step is the dominating one. It is also inefficient - we are computing a whole matrix just for one entry.
We can improve on this:
Let $\mathbf{v}$ be the $1 \times n$ vector $\begin{pmatrix} 0 & 0 & \dots & 0 & 1 \end{pmatrix}$. Observe that for a matrix B, $\mathbf{v}B$ is the last row of B.

Given $B = A_k$, we find $((A_k)^j)_{k,k}$ for each j in $\log n$ rounds:

- First, compute $\mathbf{v}B$.
- In the first round, compute B^2 and $\mathbf{v}B^2$, then multiply $\mathbf{v}B \cdot B^2$ to get $\mathbf{v}B^3$.
- Next, compute B^4 and $\mathbf{v}B^4$, and multiply B^4 by the matrix with rows $\mathbf{v}B, \mathbf{v}B^2, \mathbf{v}B^3$. This gives the matrix with rows $\mathbf{v}B^5, \mathbf{v}B^6, \mathbf{v}B^7$.

After $\log n$ rounds, this gives us the $n \times n$ matrix with rows $\mathbf{v}B, \mathbf{v}B^2, \ldots, \mathbf{v}B^n$. The required numbers are just the last entries in each row.

After $\log n$ rounds, this gives us the $n \times n$ matrix with rows $\mathbf{v}B, \mathbf{v}B^2, \ldots, \mathbf{v}B^n$. The required numbers are just the last entries in each row.

The point is that we are only performing $O(\log n)$ matrix multiplications (a constant number at each step). So the number of operations is $O(n^3 \log n)$ instead of $O(n^4)$.

After $\log n$ rounds, this gives us the $n \times n$ matrix with rows $\mathbf{v}B, \mathbf{v}B^2, \ldots, \mathbf{v}B^n$. The required numbers are just the last entries in each row.

The point is that we are only performing $O(\log n)$ matrix multiplications (a constant number at each step). So the number of operations is $O(n^3 \log n)$ instead of $O(n^4)$.

Doing this for all A_k adds a factor of n, and gives us $O(n^4 \log n)$ operations.

- The determinant has some deep structure, which can be exploited to find efficient parallel algorithms. (Unlike the permanent, for example)
- The high-level idea in both algorithms is to work with polynomial identities.

[1] L. Csanky. "Fast Parallel Matrix Inversion Algorithms". In: *SIAM Journal on Computing* 5.4 (Dec. 1976), pp. 618–623. ISSN: 0097-5397, 1095-7111. DOI: 10.1137/0205040. URL: http://epubs.siam.org/doi/10.1137/0205040.

[2] Dexter C. Kozen. *The Design and Analysis of Algorithms*. New York, NY: Springer New York, 1992. ISBN: 9781461287575 9781461244004. DOI: 10.1007/978-1-4612-4400-4. URL: http://link.springer.com/10.1007/978-1-4612-4400-4.

[3] Nicholas Pippenger. "A Formula for the Determinant". In: (2022). DOI: 10.48550/ARXIV.2206.00134. URL: https://arxiv.org/abs/2206.00134.

[4] Michael Soltys. *An Introduction to the Analysis of Algorithms*. 3rd ed. WORLD SCIENTIFIC, Mar. 2018. ISBN: 9789813235908 9789813235915. DOI: 10.1142/10875. URL: http://www.worldscientific.com/worldscibooks/10.1142/10875.