3-15 Matrix

Jun Ma

majun@nju.edu.cn

December 31, 2020

TC 28.1-2

Find an LU decomposition of the matrix

$$\begin{pmatrix} 4 & -5 & 6 \\ 8 & -6 & 7 \\ 12 & -7 & 12 \end{pmatrix}.$$

TC 28.1-2

Find an LU decomposition of the matrix

$$\begin{pmatrix} 4 & -5 & 6 \\ 8 & -6 & 7 \\ 12 & -7 & 12 \end{pmatrix}.$$

$$L = \begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 3 & 2 & 1 \end{pmatrix}, U = \begin{pmatrix} 4 & -5 & 6 \\ 0 & 4 & -5 \\ 0 & 0 & 4 \end{pmatrix}$$

TC 28.1-3

Solve the equation

$$\begin{pmatrix} 1 & 5 & 4 \\ 2 & 0 & 3 \\ 5 & 8 & 2 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 12 \\ 9 \\ 5 \end{pmatrix}$$

by using an LUP decomposition.

TC 28.1-3

Solve the equation

$$\begin{pmatrix} 1 & 5 & 4 \\ 2 & 0 & 3 \\ 5 & 8 & 2 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 12 \\ 9 \\ 5 \end{pmatrix}$$

by using an LUP decomposition.

$$P = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}, L = \begin{pmatrix} 1 & 0 & 0 \\ 1/5 & 1 & 0 \\ 2/5 & -16/17 & 1 \end{pmatrix}, U = \begin{pmatrix} 5 & 8 & 2 \\ 0 & 17/5 & 18/5 \\ 0 & 0 & 95/17 \end{pmatrix}$$

TC 28.1-3

Solve the equation

$$\begin{pmatrix} 1 & 5 & 4 \\ 2 & 0 & 3 \\ 5 & 8 & 2 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 12 \\ 9 \\ 5 \end{pmatrix}$$

by using an LUP decomposition.

$$P = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}, L = \begin{pmatrix} 1 & 0 & 0 \\ 1/5 & 1 & 0 \\ 2/5 & -16/17 & 1 \end{pmatrix}, U = \begin{pmatrix} 5 & 8 & 2 \\ 0 & 17/5 & 18/5 \\ 0 & 0 & 95/17 \end{pmatrix}$$

$$LUx = Pb \Rightarrow LUx = \begin{pmatrix} 5 \\ 12 \\ 9 \end{pmatrix} \Rightarrow y = Ux = \begin{pmatrix} 5 \\ 11 \\ 295/17 \end{pmatrix} \Rightarrow x = \begin{pmatrix} -3/19 \\ -1/19 \\ 59/19 \end{pmatrix}$$

TC 28.1-6

Show that for all $n \geq 1$, there exists a singular $n \times n$ matrix that has an LU decomposition.

TC 28.1-6

Show that for all $n \geq 1$, there exists a singular $n \times n$ matrix that has an LU decomposition.

The idea behind LUP decomposition is to find three $n \times n$ matrices L, U, and P such that

$$PA = LU \ , \tag{28.4}$$

where

- L is a unit lower-triangular matrix,
- U is an upper-triangular matrix, and
- P is a permutation matrix.

TC 28.1-6

An ***upper-triangular matrix*** U is one for which $u_{ij} = 0$ if $i > j$. All entries below the diagonal are zero:

$$U = \begin{pmatrix} u_{11} & u_{12} & \dots & u_{1n} \\ 0 & u_{22} & \dots & u_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & u_{nn} \end{pmatrix}.$$

An upper-triangular matrix is ***unit upper-triangular*** if it has all 1s along the diagonal.

A ***lower-triangular matrix*** L is one for which $l_{ij} = 0$ if $i < j$. All entries above the diagonal are zero:

$$L = \begin{pmatrix} l_{11} & 0 & \dots & 0 \\ l_{21} & l_{22} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ l_{n1} & l_{n2} & \dots & l_{nn} \end{pmatrix}.$$

A lower-triangular matrix is ***unit lower-triangular*** if it has all 1s along the diagonal.

TC 28.1-6

The **zero matrix** always has an LU decomposition

L: any unit lower-triangular matrix

U: the zero matrix, which is upper triangular.

$$\begin{pmatrix} 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & \cdots & 0 \end{pmatrix} = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & \cdots & 1 \end{pmatrix} \begin{pmatrix} 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & \cdots & 0 \end{pmatrix}$$

TC 28.1-7

In LU-DECOMPOSITION, is it necessary to perform the outermost **for** loop iteration when $k = n$?
How about in LUP-DECOMPOSITION?

```
LU-DECOMPOSITION(A)
 1  n = A.rows
 2  let L and U be new n × n matrices
 3  initialize U with 0s below the diagonal
 4  initialize L with 1s on the diagonal and 0s above the diagonal
 5  for k = 1 to n
 6      u_kk = a_kk
 7      for i = k + 1 to n
 8          l_ik = a_ik / u_kk          // l_ik holds v_i
 9          u_ki = a_ki                 // u_ki holds w_i^T
10      for i = k + 1 to n
11          for j = k + 1 to n
12              a_ij = a_ij − l_ik u_kj
13  return L and U
```

TC 28.1-7

In LU-DECOMPOSITION, is it necessary to perform the outermost **for** loop iteration when $k = n$?
How about in LUP-DECOMPOSITION?

```
LU-DECOMPOSITION(A)
 1  n = A.rows
 2  let L and U be new n × n matrices
 3  initialize U with 0s below the diagonal
 4  initialize L with 1s on the diagonal and 0s above the diagonal
 5  for k = 1 to n
 6      u_kk = a_kk
 7      for i = k + 1 to n
 8          l_ik = a_ik / u_kk          // l_ik holds v_i
 9          u_ki = a_ki                 // u_ki holds w_i^T
10      for i = k + 1 to n
11          for j = k + 1 to n
12              a_ij = a_ij − l_ik u_kj
13  return L and U
```

TC 28.1-7

In LU-DECOMPOSITION, is it necessary to perform the outermost **for** loop iteration when $k = n$?
How about in LUP-DECOMPOSITION?

LU-DECOMPOSITION(A)

```
1   n = A.rows
2   let L and U be new n × n matrices
3   initialize U with 0s below the diagonal
4   initialize L with 1s on the diagonal and 0s above the diagonal
5   for k = 1 to n
6       u_kk = a_kk
7       for i = k + 1 to n
8           l_ik = a_ik / u_kk          // l_ik holds v_i
9           u_ki = a_ki                 // u_ki holds w_i^T
10      for i = k + 1 to n
11          for j = k + 1 to n
12              a_ij = a_ij − l_ik u_kj
13  return L and U
```

LUP-DECOMPOSITION(A)

```
1   n = A.rows
2   let π[1..n] be a new array
3   for i = 1 to n
4       π[i] = i
5   for k = 1 to n
6       p = 0
7       for i = k to n
8           if |a_ik| > p
9               p = |a_ik|
10              k' = i
11      if p == 0
12          error "singular matrix"
13      exchange π[k] with π[k']
14      for i = 1 to n
15          exchange a_ki with a_k'i
16      for i = k + 1 to n
17          a_ik = a_ik / a_kk
18          for j = k + 1 to n
19              a_ij = a_ij − a_ik a_kj
```

TC 28.1-7

In LU-DECOMPOSITION, is it necessary to perform the outermost **for** loop iteration when $k = n$?
How about in LUP-DECOMPOSITION?

LU-DECOMPOSITION(A)

```
1   n = A.rows
2   let L and U be new n × n matrices
3   initialize U with 0s below the diagonal
4   initialize L with 1s on the diagonal and 0s above the diagonal
5   for k = 1 to n
6       u_kk = a_kk
7       for i = k + 1 to n
8           l_ik = a_ik / u_kk          // l_ik holds v_i
9           u_ki = a_ki                 // u_ki holds w_i^T
10      for i = k + 1 to n
11          for j = k + 1 to n
12              a_ij = a_ij − l_ik u_kj
13  return L and U
```

LUP-DECOMPOSITION(A)

```
1   n = A.rows
2   let π[1..n] be a new array
3   for i = 1 to n
4       π[i] = i
5   for k = 1 to n
6       p = 0
7       for i = k to n
8           if |a_ik| > p
9               p = |a_ik|
10              k' = i
11      if p == 0
12          error "singular matrix"
13      exchange π[k] with π[k']
14      for i = 1 to n
15          exchange a_ki with a_k'i
16      for i = k + 1 to n
17          a_ik = a_ik / a_kk
18          for j = k + 1 to n
19              a_ij = a_ij − a_ik a_kj
```

TC 28.2-1

Let $M(n)$ be the time to multiply two $n \times n$ matrices, and let $S(n)$ denote the time required to square an $n \times n$ matrix. Show that multiplying and squaring matrices have essentially the same difficulty: an $M(n)$-time matrix-multiplication algorithm implies an $O(M(n))$-time squaring algorithm, and an $S(n)$-time squaring algorithm implies an $O(S(n))$-time matrix-multiplication algorithm.

TC 28.2-1

$M(n)$-time matrix-multiplication $\Rightarrow O(M(n))$-time squaring

TC 28.2-1

$M(n)$-time matrix-multiplication $\Rightarrow O(M(n))$-time squaring

TRIVIAL!!!

TC 28.2-1

$S(n)$-time matrix-squaring $\Rightarrow$ $O(S(n))$-time matrix-multiplication

TC 28.2-1

$S(n)$-time matrix-squaring $\Rightarrow$ $O(S(n))$-time matrix-multiplication

$$C = \begin{pmatrix} I & A \\ 0 & B \end{pmatrix}, C^2 = \begin{pmatrix} I & A + AB \\ 0 & B^2 \end{pmatrix}$$

TC 28.2-1

$S(n)$-time matrix-squaring $\Rightarrow$ $O(S(n))$-time matrix-multiplication

$$C = \begin{pmatrix} I & A \\ 0 & B \end{pmatrix}, C^2 = \begin{pmatrix} I & A + AB \\ 0 & B^2 \end{pmatrix}$$

Regularity Condition: $S(2n) = O(S(n))$?

TC 28.2-1

$S(n)$-time matrix-squaring $\Rightarrow$ $O(S(n))$-time matrix-multiplication

TC 28.2-1

$S(n)$-time matrix-squaring $\Rightarrow$ $O(S(n))$-time matrix-multiplication

$$C = \begin{pmatrix} 0 & A \\ B & 0 \end{pmatrix}, C^2 = \begin{pmatrix} AB & 0 \\ 0 & BA \end{pmatrix}$$

Regularity Condition: $S(2n) = O(S(n))$?

TC 28.2-2

Let $M(n)$ be the time to multiply two $n \times n$ matrices, and let $L(n)$ be the time to compute the LUP decomposition of an $n \times n$ matrix. Show that multiplying matrices and computing LUP decompositions of matrices have essentially the same difficulty: an $M(n)$-time matrix-multiplication algorithm implies an $O(M(n))$-time LUP-decomposition algorithm, and an $L(n)$-time LUP-decomposition algorithm implies an $O(L(n))$-time matrix-multiplication algorithm.

TC 28.2-2

$M(n)$-time matrix-multiplication $\Rightarrow O(M(n))$-time LUP

https://math.stackexchange.com/questions/1084486/show-that-matrices-multiplication-and-lup-decompositions-have-the-same-difficult

TC 28.2-2

$M(n)$-time matrix-multiplication $\Rightarrow$ $O(M(n))$-time LUP

Let A be an $n \times n$ matrix. Without loss of generality, assume n is a power of 2. If n is not a power of 2, we can always pad it with an identity matrix of size k:

$$\begin{pmatrix} A & 0 \\ 0 & I_k \end{pmatrix}$$

without affecting the asymptotic running time of the LUP factorization. We want to find matrices P, L, and U such that $PA = LU$, P is a permutation matrix (e.g. a permutation of the rows of I), L is unit lower triangular (with ones on diagonal), and U is upper-triangular (but not necessarily with ones on diagonal).

---

https://math.stackexchange.com/questions/1084486/show-that-matrices-multiplication-and-lup-decompositions-have-the-same-difficult

TC 28.2-2

$M(n)$-time matrix-multiplication $\Rightarrow O(M(n))$-time LUP

Now split A into equal-sized blocks:

$$A = \begin{pmatrix} B & C \\ D & E \end{pmatrix}$$

Recursively perform a factorization $B = P_1 L_1 U_1$. We then have

$$A = \begin{pmatrix} P_1 & 0 \\ 0 & I \end{pmatrix} \begin{pmatrix} L_1 U_1 & P_1^T C \\ D & E \end{pmatrix}$$

$$= \begin{pmatrix} P_1 & 0 \\ 0 & I \end{pmatrix} \begin{pmatrix} L_1 & 0 \\ DU_1^{-1} & E - DU_1^{-1} L_1^{-1} P_1^T C \end{pmatrix} \begin{pmatrix} U_1 & L_1^{-1} P_1^T C \\ 0 & I \end{pmatrix}$$

---

https://math.stackexchange.com/questions/1084486/show-that-matrices-multiplication-and-lup-decompositions-have-the-same-difficult

TC 28.2-2

$M(n)$-time matrix-multiplication $\Rightarrow O(M(n))$-time LUP

Now perform a second factorization $E - DU_1^{-1}L_1^{-1}P_1^TC = P_2L_2U_2$. We then have

$$
\begin{aligned}
A &= \begin{pmatrix} P_1 & 0 \\ 0 & I \end{pmatrix} \begin{pmatrix} L_1 & 0 \\ DU_1^{-1} & P_2L_2U_2 \end{pmatrix} \begin{pmatrix} U_1 & L_1^{-1}P_1^TC \\ 0 & I \end{pmatrix} \\
&= \begin{pmatrix} P_1 & 0 \\ 0 & I \end{pmatrix} \begin{pmatrix} I & 0 \\ 0 & P_2 \end{pmatrix} \begin{pmatrix} L_1 & 0 \\ DU_1^{-1} & L_2 \end{pmatrix} \begin{pmatrix} U_1 & L_1^{-1}P_1^TC \\ 0 & U_2 \end{pmatrix} \\
&= \begin{pmatrix} P_1 & 0 \\ 0 & P_2 \end{pmatrix} \begin{pmatrix} L_1 & 0 \\ DU_1^{-1} & L_2 \end{pmatrix} \begin{pmatrix} U_1 & L_1^{-1}P_1^TC \\ 0 & U_2 \end{pmatrix}
\end{aligned}
$$

---

https://math.stackexchange.com/questions/1084486/show-that-matrices-multiplication-and-lup-decompositions-have-the-same-difficult

TC 28.2-2

$M(n)$-time matrix-multiplication $\Rightarrow O(M(n))$-time LUP

Hence, $A = PLU$, where

$$P = \begin{pmatrix} P_1 & 0 \\ 0 & P_2 \end{pmatrix}, \; L = \begin{pmatrix} L_1 & 0 \\ DU_1^{-1} & L_2 \end{pmatrix}, \; U = \begin{pmatrix} U_1 & L_1^{-1}P_1^T C \\ 0 & U_2 \end{pmatrix}$$

Let $T(n)$ be the running time for this LUP factorization algorithm. I'll use $T(n)$ instead of $L(n)$ to reduce confusion with the L matrix. We performed two recursive LUP factorizations on input size $n/2$, along with some matrix multiplications and inversions, both of which take time $M(n)$. Hence, the running time satisfies

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

Assuming $M(n) = \Omega(n^2)$ and using the Master theorem or just performing the summation manually, we obtain the result $T(n) = O(M(n))$. QED

https://math.stackexchange.com/questions/1084486/show-that-matrices-multiplication-and-lup-decompositions-have-the-same-difficult

TC 28.2-2

$L(n)$-time LUP $\Rightarrow$ $O(L(n))$-time matrix-multiplication

https://www.cs.dartmouth.edu/ thc/clrs-bugs/bugs-3e.php

TC 28.2-2

$L(n)$-time LUP $\Rightarrow$ $O(L(n))$-time matrix-multiplication

Page 832, Exercise 28.2-2. Eliminate one direction of the exercise, so that the exercise should read Let $M(n)$ be the time to multiply two $n \times n$ matrices. Show that an $M(n)$-time matrix-multiplication algorithm implies an $O(M(n))$-time LUP-decomposition algorithm.

Reported by Joel Seiferas. Posted 15 June 2010.
Severity level: 4
Corrected in the third printing of the third edition.

Severity levels

1. A minor typographical error that should not affect your understanding.
2. A minor technical or expository error.
3. A more significant technical or expository error.
4. A serious error in the exposition of an algorithm, or an error that requires significant change to the text.

https://www.cs.dartmouth.edu/ thc/clrs-bugs/bugs-3e.php

TC 28.2-3

Let $M(n)$ be the time to multiply two $n \times n$ matrices, and let $D(n)$ denote the time required to find the determinant of an $n \times n$ matrix. Show that multiplying matrices and computing the determinant have essentially the same difficulty: an $M(n)$-time matrix-multiplication algorithm implies an $O(M(n))$-time determinant algorithm, and a $D(n)$-time determinant algorithm implies an $O(D(n))$-time matrix-multiplication algorithm.

TC 28.2-3

$M(n)$-time matrix-multiplication $\Rightarrow O(M(n))$-time determinant algorithm

TC 28.2-3

$M(n)$-time matrix-multiplication $\Rightarrow$ $O(M(n))$-time determinant algorithm

$$PA = LU$$

$$A = P^{-1}LU$$

$$\det(A) = \det(P^{-1})\det(L)\det(U) = \det(P^{-1})\det(U)$$

TC 28.2-3

$M(n)$-time matrix-multiplication $\Rightarrow$ $O(M(n))$-time determinant algorithm

$$PA = LU$$

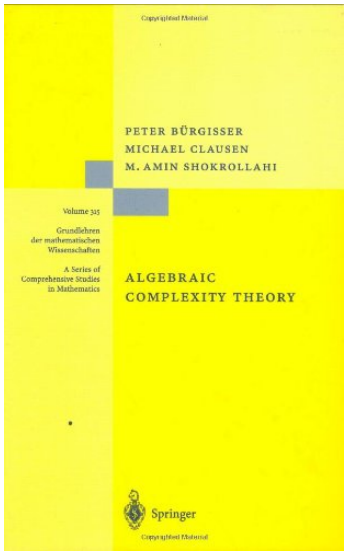
$$A = P^{-1}LU$$

$$\det(A) = \det(P^{-1})\det(L)\det(U) = \det(P^{-1})\det(U)$$

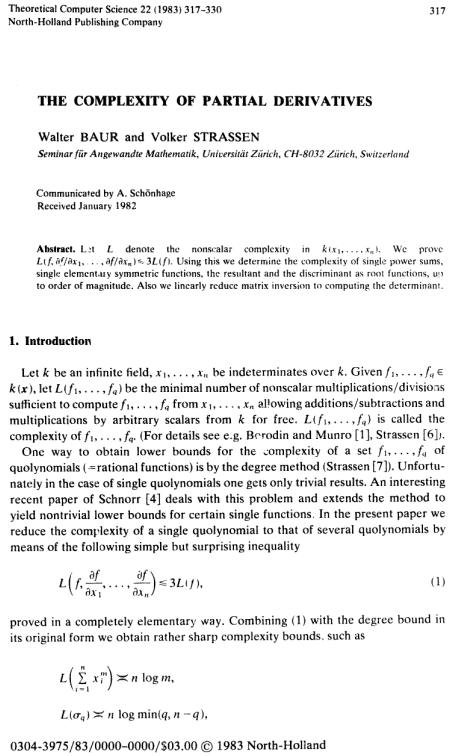
LUP: $O(M(n))$
$\det(L)$: $O(n)$
$\det(U)$: $O(n)$
$\det(P^{-1})$: $O(n)$
So, $\det(A)$: $O(M(n))$

TC 28.2-3

$D(n)$-time determinant algorithm $\Rightarrow$ $O(D(n))$-time matrix-multiplication



(16.7) Theorem. *We have* $\omega(LUP) = \omega(Det) = \omega$.



Theoretical Computer Science 22 (1983) 317–330
North-Holland Publishing Company

317

THE COMPLEXITY OF PARTIAL DERIVATIVES

Walter BAUR and Volker STRASSEN
Seminar für Angewandte Mathematik, Universität Zürich, CH-8032 Zürich, Switzerland

Communicated by A. Schönhage
Received January 1982

Abstract. Let L denote the nonscalar complexity in $k(x_1, \ldots, x_n)$. We prove $L(f, \partial f/\partial x_1, \ldots, \partial f/\partial x_n) \leq 3L(f)$. Using this we determine the complexity of single power sums, single elementary symmetric functions, the resultant and the discriminant as rational functions, up to order of magnitude. Also we linearly reduce matrix inversion to computing the determinant.

1. Introduction

Let k be an infinite field, $x_1, \ldots, x_n$ be indeterminates over k. Given $f_1, \ldots, f_q \in k(x)$, let $L(f_1, \ldots, f_q)$ be the minimal number of nonscalar multiplications/divisions sufficient to compute $f_1, \ldots, f_q$ from $x_1, \ldots, x_n$ allowing additions/subtractions and multiplications by arbitrary scalars from k for free. $L(f_1, \ldots, f_q)$ is called the complexity of $f_1, \ldots, f_q$. (For details see e.g. Borodin and Munro [1], Strassen [6]).

One way to obtain lower bounds for the complexity of a set $f_1, \ldots, f_q$ of polynomials (=rational functions) is by the degree method (Strassen [7]). Unfortunately in the case of single polynomials one gets only trivial results. An interesting recent paper of Schnorr [4] deals with this problem and extends the method to yield nontrivial lower bounds for certain single functions. In the present paper we reduce the complexity of a single polynomial to that of several polynomials by means of the following simple but surprising inequality

$$L\left(f, \frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_n}\right) \leq 3L(f), \tag{1}$$

proved in a completely elementary way. Combining (1) with the degree bound in its original form we obtain rather sharp complexity bounds. such as

$$L\left(\sum_{i=1}^{n} x_i^m\right) \asymp n \log m,$$

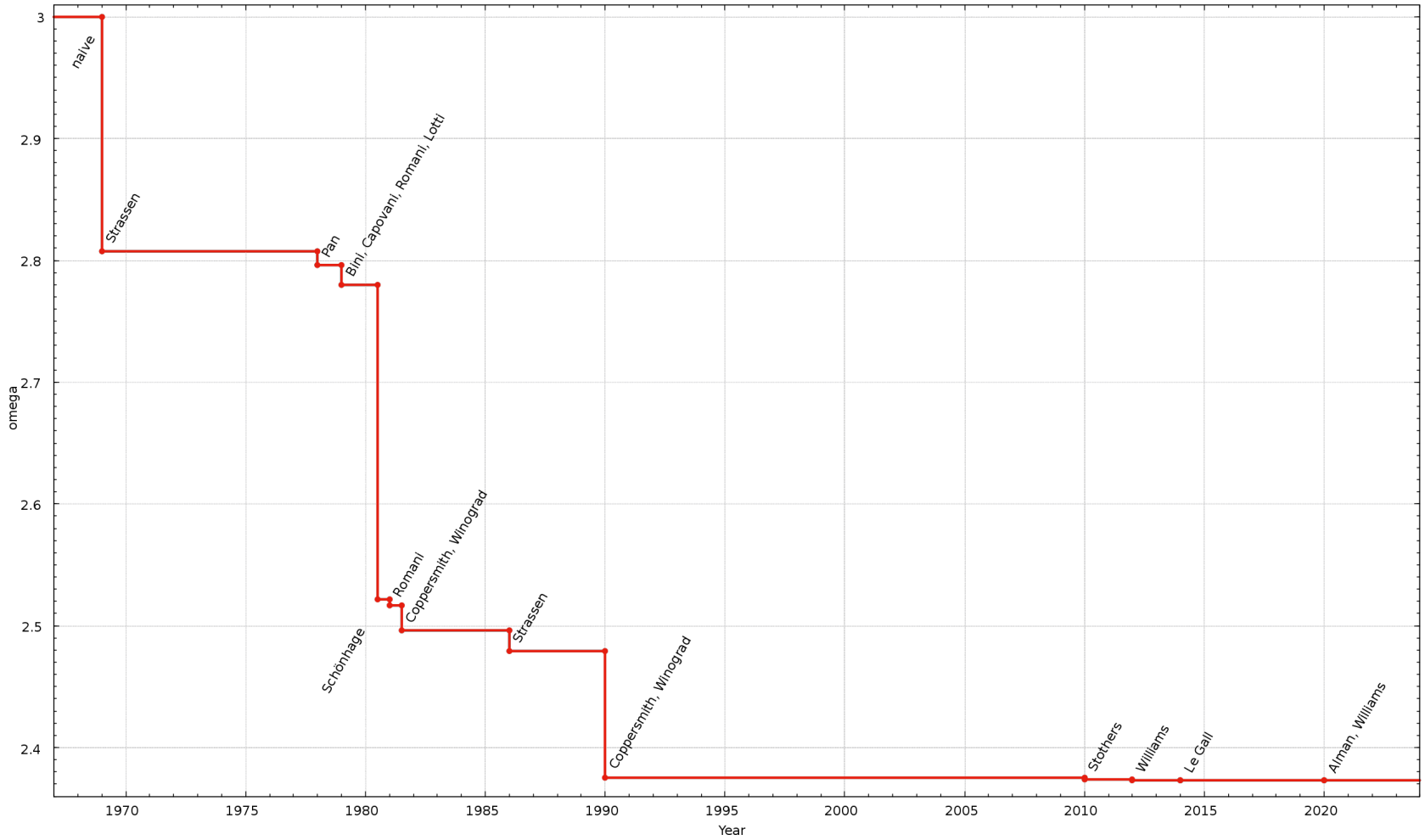
$$L(\sigma_q) \asymp n \log \min(q, n-q),$$

0304-3975/83/0000–0000/$03.00

https://link.springer.com/book/10.1007/978-3-662-03338-8

https://www.sciencedirect.com/science/article/pii/030439758390110X

Complexity of Matrix Multiplication



https://en.wikipedia.org/wiki/Matrix_multiplication#cite_note-27

Complexity of Matrix Multiplication

- Strassen (1969)[1], $O(n^{\log_2 7})$
- Coppersmith–Winograd (1990)[2], $O(n^{2.3755})$
- Stothers (2010)[3], $O(n^{2.3737})$
- Virginia Vassilevska Williams (2012)[4], $O(n^{2.3729})$
- François Le Gall (2014) [5], $O(n^{2.3728639})$
- Josh Alman and Virginia Vassilevska Williams (2020, up-to-date)[6], $O(n^{2.3728596})$

[1] Volker Strassen (Aug 1969). "Gaussian elimination is not optimal". Numerische Mathematik. 13 (4): 354–356.

[2] D. Coppersmith; S. Winograd (Mar 1990). "Matrix multiplication via arithmetic progressions". Journal of Symbolic Computation. 9 (3): 251–280.

[3] Stothers, Andrew James (2010). On the complexity of matrix multiplication (Ph.D. thesis). University of Edinburgh.

[4] Virginia Vassilevska Williams (2012). "Multiplying Matrices Faster than Coppersmith-Winograd". In Howard J. Karloff; Toniann Pitassi (eds.). Proc. 44th Symposium on Theory of Computing (STOC). ACM. pp. 887–898

[5] Le Gall, François (2014), "Powers of tensors and fast matrix multiplication", in Katsusuke Nabeshima (ed.), Proceedings of the 39th International Symposium on Symbolic and Algebraic Computation (ISSAC), pp. 296–303

[6] Alman, Josh; Williams, Virginia Vassilevska (2020), "A Refined Laser Method and Faster Matrix Multiplication", 32nd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2021)

Complexity of Matrix Multiplication

It is **unknown** whether $2 < \omega$.

The **largest known** lower bound for matrix-multiplication complexity is $\Omega(n^{2\log(n)})$

https://en.wikipedia.org/wiki/Matrix_multiplication#cite_note-27

TC 28.3-1

Prove that every diagonal element of a symmetric positive-definite matrix is positive.

TC 28.3-1

Prove that every diagonal element of a symmetric positive-definite matrix is positive.

Proof.

$$A \text{ is symmetric positive-definite}$$

$$\Downarrow$$

$$\forall i, 1 \leq i \leq n (e_i^T A e_i = A_{ii}) > 0$$

TC 28.3-3

Prove that the maximum element in a symmetric positive-definite matrix lies on the diagonal.

TC 28.3-3

Prove that the maximum element in a symmetric positive-definite matrix lies on the diagonal.

Proof.

- $n = 1$: obviously
- $n \geq 2$: assume the maximum element does not lie on the diagonal, and $a_{i_j} = a_{ji} (i < j)$ is the maximum element of A, then

$$\begin{pmatrix} \vdots \\ 1 \\ \vdots \\ -1 \\ \vdots \end{pmatrix}^{\mathrm{T}} \begin{pmatrix} a_{ii} & \cdots & a_{ij} \\ \vdots & \ddots & \vdots \\ a_{ji} & \cdots & a_{jj} \end{pmatrix} \begin{pmatrix} \vdots \\ 1 \\ \vdots \\ -1 \\ \vdots \end{pmatrix} = a_{ii} + a_{jj} - 2a_{ij} > 0$$

which is conflict with $a_{ij} > a_{ii}$ and $a_{ji} > a_{jj}$

TC Problem 28-1 (Tridiagonal systems of linear equations)

Consider the tridiagonal matrix

$$A = \begin{pmatrix} 1 & -1 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & -1 & 2 \end{pmatrix}.$$

a. Find an LU decomposition of A.

b. Solve the equation $Ax = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \end{pmatrix}^{\mathrm{T}}$ by using forward and back substitution.

c. Find the inverse of A.

d. Show how, for any $n \times n$ symmetric positive-definite, tridiagonal matrix A and any n-vector b, to solve the equation $Ax = b$ in $O(n)$ time by performing an LU decomposition. Argue that any method based on forming A^{-1} is asymptotically more expensive in the worst case.

e. Show how, for any $n \times n$ nonsingular, tridiagonal matrix A and any n-vector b, to solve the equation $Ax = b$ in $O(n)$ time by performing an LUP decomposition.

TC Problem 28-1 (Tridiagonal systems of linear equations)

(a)Find an LU decomposition of A

$$\begin{pmatrix} 1 & -1 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & -1 & 2 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 & 0 \\ 0 & -1 & 1 & 0 & 0 \\ 0 & 0 & -1 & 1 & 0 \\ 0 & 0 & 0 & -1 & 1 \end{pmatrix} \begin{pmatrix} 1 & -1 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 1 & -1 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

TC Problem 28-1 (Tridiagonal systems of linear equations)

(b) Solve the equation $Ax = (1\ 1\ 1\ 1\ 1)^T$ by using forward and back substitution.

TC Problem 28-1 (Tridiagonal systems of linear equations)

(b) Solve the equation $Ax = (1\ 1\ 1\ 1\ 1)^T$ by using forward and back substitution.

$$y = Ux = (1\ 2\ 3\ 4\ 5)^T$$

$$x = (15\ 14\ 12\ 9\ 5)^T$$

TC Problem 28-1 (Tridiagonal systems of linear equations)

(c) Find the inverse of A.

TC Problem 28-1 (Tridiagonal systems of linear equations)

(c) Find the inverse of A.

$$\begin{pmatrix} 5 & 4 & 3 & 2 & 1 \\ 4 & 4 & 3 & 2 & 1 \\ 3 & 3 & 3 & 2 & 1 \\ 2 & 2 & 2 & 2 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

TC Problem 28-1 (Tridiagonal systems of linear equations)

(d) Show how, for any $n \times n$ **symmetric positive-definite, tridiagonal matrix** A and any n-vector b, to solve the equation $Ax = b$ in $O(n)$ time by performing an LU decomposition. Argue that any method based on forming A^{-1} is asymptotically more expensive in the worst case.

TC Problem 28-1 (Tridiagonal systems of linear equations)

An important class of matrices for which **LU decomposition always works** correctly is the class of **symmetric positive-definite matrices**.

TC Problem 28-1 (Tridiagonal systems of linear equations)

An important class of matrices for which **LU decomposition always works** correctly is the class of **symmetric positive-definite matrices**.

LU-DECOMPOSITION(A)

```
1  n = A.rows
2  let L and U be new n × n matrices
3  initialize U with 0s below the diagonal
4  initialize L with 1s on the diagonal and 0s above the diagonal
5  for k = 1 to n
6      u_kk = a_kk
7      for i = k + 1 to n
8          l_ik = a_ik / u_kk          // l_ik holds v_i
9          u_ki = a_ki                 // u_ki holds w_i^T
10     for i = k + 1 to n
11         for j = k + 1 to n
12             a_ij = a_ij − l_ik u_kj
13 return L and U
```

TC Problem 28-1 (Tridiagonal systems of linear equations)

An important class of matrices for which **LU decomposition always works** correctly is the class of **symmetric positive-definite matrices**.

LU-DECOMPOSITION(A)

```
1  n = A.rows
2  let L and U be new n × n matrices
3  initialize U with 0s below the diagonal
4  initialize L with 1s on the diagonal and 0s above the diagonal
5  for k = 1 to n
6      u_kk = a_kk
7      for i = k + 1 to  k + 1
8          l_ik = a_ik / u_kk          // l_ik holds v_i
9          u_ki = a_ki                 // u_ki holds w_i^T
10     for i = k + 1 to n
11         for j = k + 1 to n
12             a_ij = a_ij − l_ik u_kj
13 return L and U
```

TC Problem 28-1 (Tridiagonal systems of linear equations)

An important class of matrices for which **LU decomposition always works** correctly is the class of **symmetric positive-definite matrices**.

LU-DECOMPOSITION(A)

```
1   n = A.rows
2   let L and U be new n × n matrices
3   initialize U with 0s below the diagonal
4   initialize L with 1s on the diagonal and 0s above the diagonal
5   for k = 1 to n
6       u_kk = a_kk
7       for i = k + 1 to k + 1
8           l_ik = a_ik / u_kk          // l_ik holds v_i
9           u_ki = a_ki                 // u_ki holds w_i^T
10      for i = k + 1 to k + 1
11          for j = k + 1 to n
12              a_ij = a_ij − l_ik u_kj
13  return L and U
```

TC Problem 28-1 (Tridiagonal systems of linear equations)

An important class of matrices for which **LU decomposition always works** correctly is the class of **symmetric positive-definite matrices**.

LU-DECOMPOSITION(A)

```
1  n = A.rows
2  let L and U be new n × n matrices
3  initialize U with 0s below the diagonal
4  initialize L with 1s on the diagonal and 0s above the diagonal
5  for k = 1 to n
6      u_kk = a_kk
7      for i = k + 1 to  k + 1
8          l_ik = a_ik / u_kk          // l_ik holds v_i
9          u_ki = a_ki                 // u_ki holds w_i^T
10     for i = k + 1 to  k + 1
11         for j = k + 1 to  k + 1
12             a_ij = a_ij − l_ik u_kj
13 return L and U
```

TC Problem 28-1 (Tridiagonal systems of linear equations)

An important class of matrices for which **LU decomposition always works** correctly is the class of **symmetric positive-definite matrices**.

LU-DECOMPOSITION(A)

```
1  n = A.rows
2  let L and U be new n × n matrices
3  initialize U with 0s below the diagonal
4  initialize L with 1s on the diagonal and 0s above the diagonal
5  for k = 1 to n
6       u_kk = a_kk
7       for i = k + 1 to  k + 1
8            l_ik = a_ik / u_kk          // l_ik holds v_i
9            u_ki = a_ki                 // u_ki holds w_i^T
10      for i = k + 1 to  k + 1
11           for j = k + 1 to  k + 1
12                a_ij = a_ij − l_ik u_kj
13 return L and U
```

LUP-SOLVE(L, U, π, b)

```
1  n = L.rows
2  let x be a new vector of length n
3  for i = 1 to n
4       y_i = b_π[i] − Σ_{j=1}^{i−1} l_ij y_j
5  for i = n downto 1
6       x_i = (y_i − Σ_{j=i+1}^{n} u_ij x_j) / u_ii
7  return x
```

TC Problem 28-1 (Tridiagonal systems of linear equations)

An important class of matrices for which **LU decomposition always works** correctly is the class of **symmetric positive-definite matrices**.

LU-DECOMPOSITION(A)

```
1  n = A.rows
2  let L and U be new n × n matrices
3  initialize U with 0s below the diagonal
4  initialize L with 1s on the diagonal and 0s above the diagonal
5  for k = 1 to n
6       u_kk = a_kk
7       for i = k + 1 to  k + 1
8            l_ik = a_ik / u_kk          // l_ik holds v_i
9            u_ki = a_ki                 // u_ki holds w_i^T
10      for i = k + 1 to  k + 1
11           for j = k + 1 to  k + 1
12                a_ij = a_ij − l_ik u_kj
13 return L and U
```

LUP-SOLVE(L, U, π, b)

```
1  n = L.rows
2  let x be a new vector of length n
3  for i = 1 to n
4       y_i = b_π[i] − Σ_{j=max(1,i−1)}^{i−1} l_ij y_j
5  for i = n downto 1
6       x_i = (y_i − Σ_{j=i+1}^{n} u_ij x_j) / u_ii
7  return x
```

TC Problem 28-1 (Tridiagonal systems of linear equations)

An important class of matrices for which **LU decomposition always works** correctly is the class of **symmetric positive-definite matrices**.

LU-DECOMPOSITION(A)

```
1  n = A.rows
2  let L and U be new n × n matrices
3  initialize U with 0s below the diagonal
4  initialize L with 1s on the diagonal and 0s above the diagonal
5  for k = 1 to n
6       u_kk = a_kk
7       for i = k + 1 to  k + 1
8            l_ik = a_ik/u_kk          // l_ik holds v_i
9            u_ki = a_ki               // u_ki holds w_i^T
10      for i = k + 1 to  k + 1
11           for j = k + 1 to  k + 1
12                a_ij = a_ij − l_ik u_kj
13 return L and U
```

LUP-SOLVE(L, U, π, b)

```
1  n = L.rows
2  let x be a new vector of length n
3  for i = 1 to n
4       y_i = b_π[i] − Σ_{j=max(1,i−1)}^{i−1} l_ij y_j
5  for i = n downto 1
6       x_i = (y_i − Σ_{j=i+1}^{min(n,i+1)} u_ij x_j)/u_ii
7  return x
```

TC Problem 28-1 (Tridiagonal systems of linear equations)

An important class of matrices for which **LU decomposition always works** correctly is the class of **symmetric positive-definite matrices**.

$$O(n)$$

LU-DECOMPOSITION(A)

```
1   n = A.rows
2   let L and U be new n × n matrices
3   initialize U with 0s below the diagonal
4   initialize L with 1s on the diagonal and 0s above the diagonal
5   for k = 1 to n
6       u_kk = a_kk
7       for i = k + 1 to k + 1
8           l_ik = a_ik/u_kk          // l_ik holds v_i
9           u_ki = a_ki               // u_ki holds w_i^T
10      for i = k + 1 to k + 1
11          for j = k + 1 to k + 1
12              a_ij = a_ij − l_ik u_kj
13  return L and U
```

LUP-SOLVE(L, U, π, b)

```
1   n = L.rows
2   let x be a new vector of length n
3   for i = 1 to n
4       y_i = b_π[i] − Σ_{j=max(1,i−1)}^{i−1} l_ij y_j
5   for i = n downto 1
6       x_i = (y_i − Σ_{j=i+1}^{min(n,i+1)} u_ij x_j)/u_ii
7   return x
```

TC Problem 28-1 (Tridiagonal systems of linear equations)

Also works for **strictly diagonally dominant tridiagonal matrices**.

A is called **strictly diagonally dominant** if $|A_{ii}| > \sum_{j \neq i} |A_{ij}|$ for all i.

$$O(n)$$

LU-DECOMPOSITION(A)

```
1  n = A.rows
2  let L and U be new n × n matrices
3  initialize U with 0s below the diagonal
4  initialize L with 1s on the diagonal and 0s above the diagonal
5  for k = 1 to n
6      u_kk = a_kk
7      for i = k + 1 to k + 1
8          l_ik = a_ik/u_kk          // l_ik holds v_i
9          u_ki = a_ki               // u_ki holds w_i^T
10     for i = k + 1 to k + 1
11         for j = k + 1 to k + 1
12             a_ij = a_ij − l_ik u_kj
13 return L and U
```

LUP-SOLVE(L, U, π, b)

```
1  n = L.rows
2  let x be a new vector of length n
3  for i = 1 to n
4      y_i = b_π[i] − Σ_{j=max(1,i−1)}^{i−1} l_ij y_j
5  for i = n downto 1
6      x_i = (y_i − Σ_{j=i+1}^{min(n,i+1)} u_ij x_j)/u_ii
7  return x
```

TC Problem 28-1 (Tridiagonal systems of linear equations)

(e) Show how, for any $n \times n$ **nonsingular, tridiagonal matrix** A and any n-vector b, to solve the equation $Ax = b$ in $O(n)$ time by performing an LUP decomposition.

TC Problem 28-1 (Tridiagonal systems of linear equations)

```
LUP-DECOMPOSITION(A)
 1  n = A.rows
 2  let π[1..n] be a new array
 3  for i = 1 to n
 4      π[i] = i
 5  for k = 1 to n
 6      p = 0
 7      for i = k to n
 8          if |a_ik| > p
 9              p = |a_ik|
10              k' = i
11      if p == 0
12          error "singular matrix"
13      exchange π[k] with π[k']
14      for i = 1 to n
15          exchange a_ki with a_k'i
16      for i = k + 1 to n
17          a_ik = a_ik / a_kk
18          for j = k + 1 to n
19              a_ij = a_ij − a_ik a_kj
```

$$\begin{pmatrix} a_{11} & a_{12} & 0 & \cdots & 0 & 0 \\ a_{21} & a_{22} & a_{23} & \cdots & 0 & 0 \\ 0 & a_{32} & a_{33} & \cdots & 0 & 0 \\ 0 & 0 & a_{43} & \cdots & 0 & 0 \\ & & & \ddots & & \\ \cdots & \cdots & \cdots & & \cdots & \cdots \\ 0 & 0 & 0 & \cdots & a_{n(n-1)} & a_{nn} \end{pmatrix} \quad A_1$$

$$\Downarrow$$

$$\begin{pmatrix} a'_{11} & a'_{12} & a'_{13} & \cdots & 0 & 0 \\ a'_{21} & a'_{22} & a'_{23} & \cdots & 0 & 0 \\ 0 & a_{32} & a_{33} & \cdots & 0 & 0 \\ 0 & 0 & a_{43} & \cdots & 0 & 0 \\ & & & \ddots & & \\ \cdots & \cdots & \cdots & & \cdots & \cdots \\ 0 & 0 & 0 & \cdots & a_{n(n-1)} & a_{nn} \end{pmatrix} \quad A_2$$

At the beginning of each iteration of the **for** loop at line 5, A_k is a tridiagonal matrix.

TC Problem 28-1 (Tridiagonal systems of linear equations)

```
LUP-DECOMPOSITION(A)
 1  n = A.rows
 2  let π[1..n] be a new array
 3  for i = 1 to n
 4      π[i] = i
 5  for k = 1 to n
 6      p = 0
 7      for i = k to n
 8          if |a_ik| > p
 9              p = |a_ik|
10              k' = i
11      if p == 0
12          error "singular matrix"
13      exchange π[k] with π[k']
14      for i = 1 to n
15          exchange a_ki with a_k'i
16      for i = k + 1 to n
17          a_ik = a_ik / a_kk
18          for j = k + 1 to n
19              a_ij = a_ij − a_ik a_kj
```

$$\begin{pmatrix} a_{11} & a_{12} & 0 & \cdots & 0 & 0 \\ a_{21} & a_{22} & a_{23} & \cdots & 0 & 0 \\ 0 & a_{32} & a_{33} & \cdots & 0 & 0 \\ 0 & 0 & a_{43} & \cdots & 0 & 0 \\ & & & \ddots & & \\ \cdots & \cdots & \cdots & \ddots & \cdots & \cdots \\ 0 & 0 & 0 & \cdots & a_{n(n-1)} & a_{nn} \end{pmatrix} \quad A_1$$

$$\Downarrow$$

$$\begin{pmatrix} a'_{11} & a'_{12} & a'_{13} & \cdots & 0 & 0 \\ a'_{21} & a'_{22} & a'_{23} & \cdots & 0 & 0 \\ 0 & a_{32} & a_{33} & \cdots & 0 & 0 \\ 0 & 0 & a_{43} & \cdots & 0 & 0 \\ & & & \ddots & & \\ \cdots & \cdots & \cdots & \ddots & \cdots & \cdots \\ 0 & 0 & 0 & \cdots & a_{n(n-1)} & a_{nn} \end{pmatrix} \quad A_2$$

How many cells would be modified in each iteration?

TC Problem 28-1 (Tridiagonal systems of linear equations)

```
1:  procedure LUP-Decomp-M(A)
2:      n = A.rows
3:      let δ[1..n] be a new array
4:      let π[1..n] be a new array
5:      for i = 1 to n do
6:          δ[i] = i
7:      for k = 1 to n do
8:          p = 0
9:          for i = k to n do
10:             if |a_{δ[i]k} > p| then
11:                 p = |a_{δ[i]k}|
12:                 k' = δ[i]
13:         exchange δ[k] with δ[k']
14:         for i = k + 1 to n do
15:             a_{δ[i]k} = a_{δ[i]k} / a_{δ[k]k}
16:             for j = k + 1 to n do
17:                 a_{δ[i]j} = a_{δ[i]j} − a_{δ[i]k} a_{δ[k]j}
18:     for i = 1 to n do
19:         π[δ[i]] = i
```

TC Problem 28-1 (Tridiagonal systems of linear equations)

```
1:  procedure LUP-DECOMP-M(A)
2:      n = A.rows
3:      let δ[1..n] be a new array
4:      let π[1..n] be a new array
5:      for i = 1 to  n do
6:          δ[i] = i
7:      for k = 1 to n do
8:          p = 0
9:          for i = k to n do
10:             if |a_{δ[i]k} > p| then
11:                 p = |a_{δ[i]k}|
12:                 k' = δ[i]
13:         exchange δ[k] with δ[k']
14:         for i = k + 1 to k + 1 do
15:             a_{δ[i]k} = a_{δ[i]k}/a_{δ[k]k}
16:             for j = k + 1 to k + 2 do
17:                 a_{δ[i]j} = a_{δ[i]j} − a_{δ[i]k} a_{δ[k]j}
18:     for i = 1 to n do
19:         π[δ[i]] = i
```

A solution?

The runtime of our LUP decomposition algorithm drops to being $O(n)$ because we know there are only ever a **constant** number of nonzero entries in each row and column, as before.
Once we have an LUP decomposition, we also know that that decomposition have both L and U having only a **constant** number of non-zero entries in **each row and column**.
This means that when we perform the forward and backward substitution, we only spend a **constant** amount of time per entry in x, and so, only takes $O(n)$ time.

https://walkccc.me/CLRS/Chap28/Problems/28-1/

How about this matrix?

$$A = \begin{pmatrix} 1 & 3 & 0 & 0 \\ 2 & 4 & 2 & 0 \\ 0 & 2 & 1 & 1 \\ 0 & 0 & 2 & 2 \end{pmatrix}$$

$$P = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{pmatrix}, L = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0.5 & 0.5 & -0.75 & 1 \end{pmatrix}, U = \begin{pmatrix} 2 & 4 & 2 & 0 \\ 0 & 2 & 1 & 1 \\ 0 & 0 & 2 & 2 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Is there anything wrong?


Thank
You!