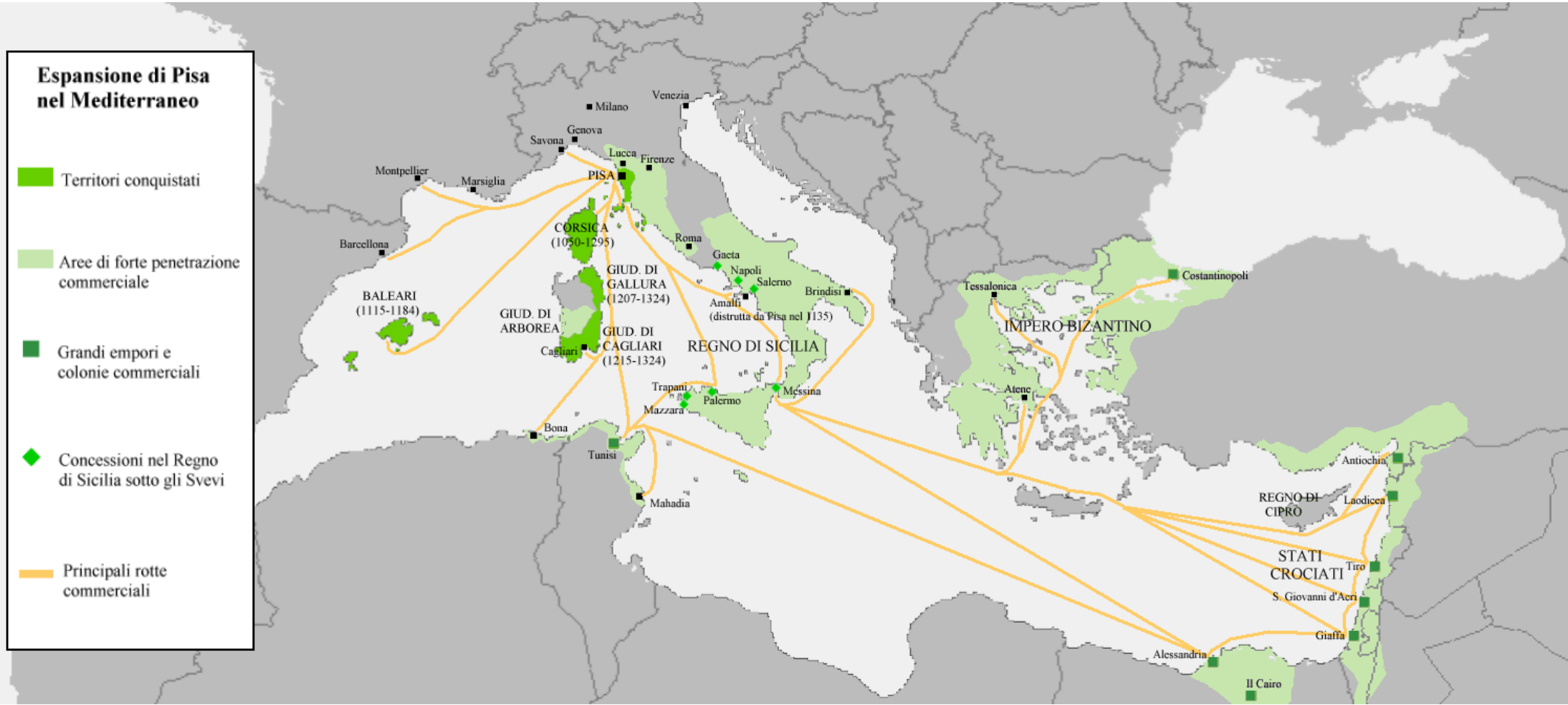
Generisk programmering

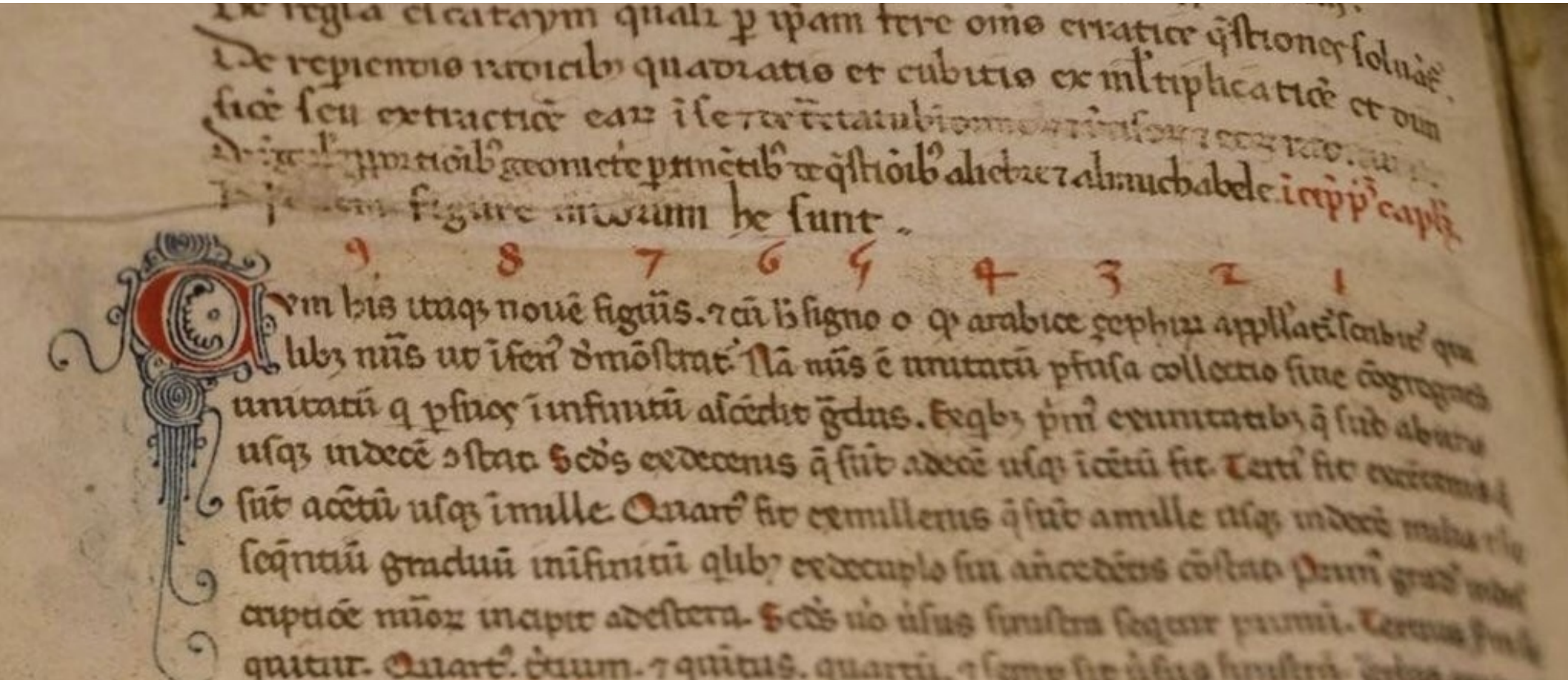
Petter Holmberg – Biolit – Mars 2022

Pisas medeltida handelsrutter



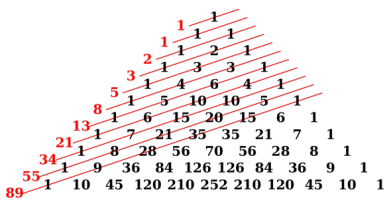
Leonardo Pisano ~(1170-1250)

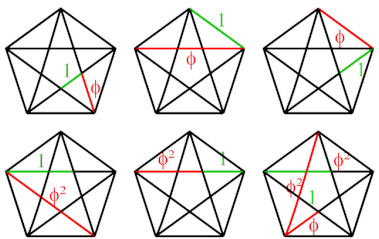
Liber Abaci (1202)

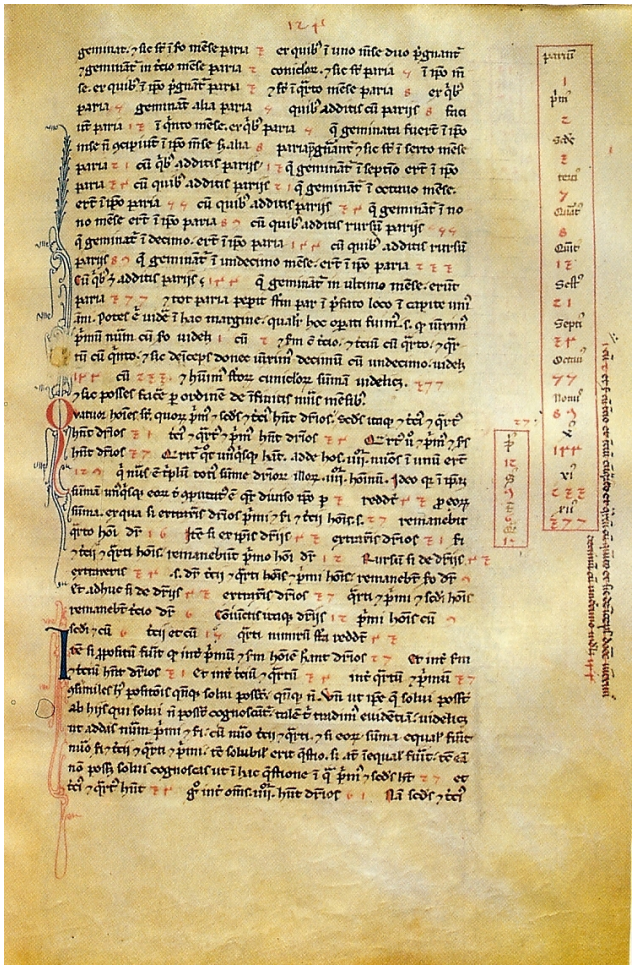


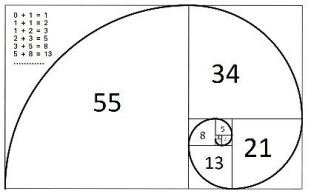
Fibonaccital

$$F_0 = 0$$
$$F_1 = 1$$
$$F_n = F_{n-1} + F_{n-2}$$

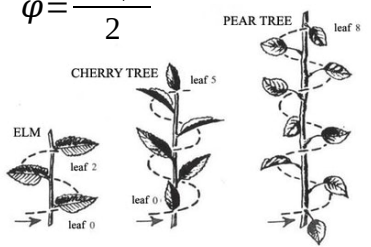








$$\varphi = \frac{1+\sqrt{5}}{2}$$



Romerska siffror

Addition med romerska siffror

XXXXI + LVIIII
LXXXXVIIIII
LXXXXVV
LXXXXX
LL
C

Multiplikation med romerska siffror

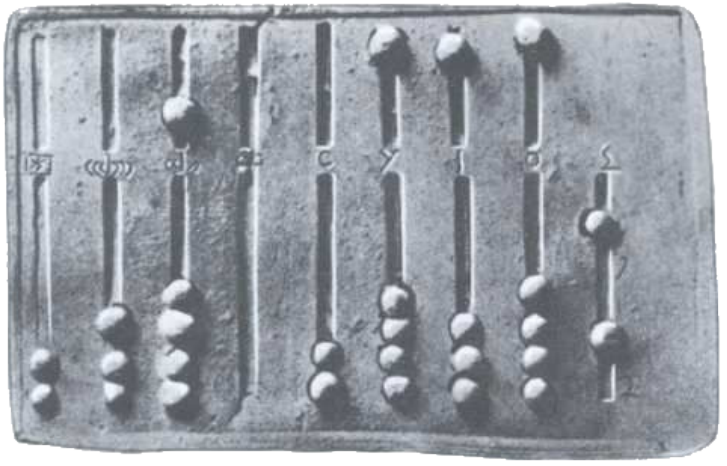
XXXXI × LVIIII
???

Multiplikation med romerska siffror

XXXXI × LVIIII

LVIIII + LVIIII

Multiplikation med romerska siffror



Naiv multiplikation i Python

```
def multiply_slow(n, a):
    if n == 1: return a
    return a + multiply_slow(n - 1, a)
```

Associativa lagen

$$(a + b) + c = a + (b + c)$$

Associativa lagen

$(a + b) + c = a + (b + c)$

XXXXI × LVIIII

LVIIII + LVIIII

Associativa lagen

$$(a + b) + c = a + (b + c)$$

XXXXI × LVIIII

(LVIIII + LVIIII + LVIIII + LVIIII + LVIIII + LVIIII + LVIIII + LVIIII + LVIIII + LVIIII) +
(LVIIII + LVIIII + LVIIII + LVIIII + LVIIII + LVIIII + LVIIII + LVIIII + LVIIII + LVIIII) +
(LVIIII + LVIIII + LVIIII + LVIIII + LVIIII + LVIIII + LVIIII + LVIIII + LVIIII + LVIIII) +
(LVIIII + LVIIII + LVIIII + LVIIII + LVIIII + LVIIII + LVIIII + LVIIII + LVIIII + LVIIII) +
LVIIII

Multiplikation med romerska siffror

41 × 59

_____

_____

Multiplikation med romerska siffror

41 × 59

———————

1

2

4

8

16

32

———————

Multiplikation med romerska siffror

41 × 59

1	*59*
2	*118*
4	*236*
8	*472*
16	*944*
32	*1888*

Multiplikation med romerska siffror

41 × 59

1	*59*	✓
2	*118*	
4	*236*	
8	*472*	✓
16	*944*	
32	*1888*	✓

Multiplikation med romerska siffror

41 × 59

1	*59*	✓
2	*118*	
4	*236*	
8	*472*	✓
16	*944*	
32	*1888*	✓
	2419	

41 *i binär form* = 101001

Romersk multiplikation i Python

```
def multiply(n, a):
    if n == 1: return a
    result = multiply(half(n), a + a)
    if odd(n): result = result + a
    return result
```

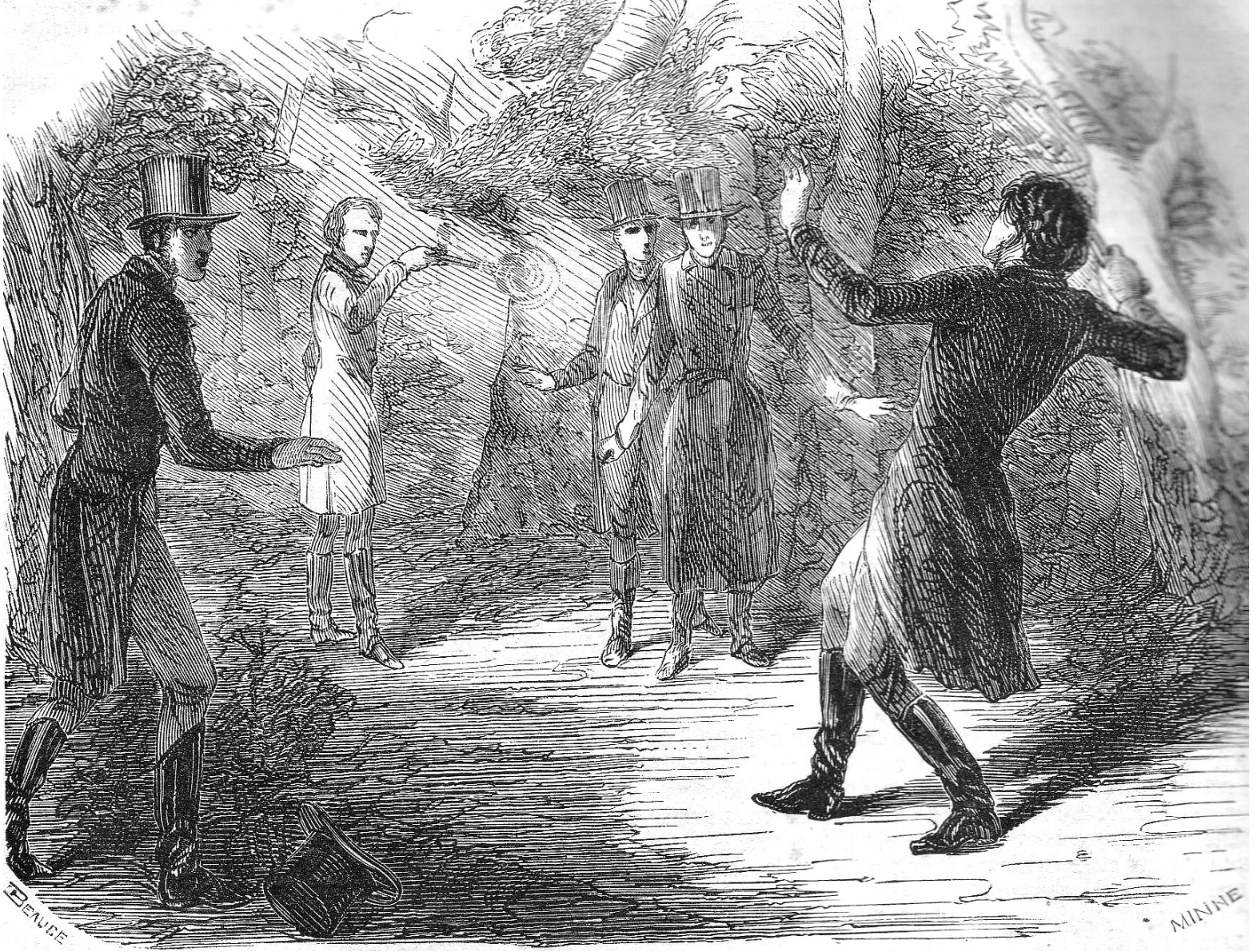
41 × 59

1	*59*	✓
2	*118*	
4	*236*	
8	*472*	✓
16	*944*	
32	*1888*	✓
	2419	

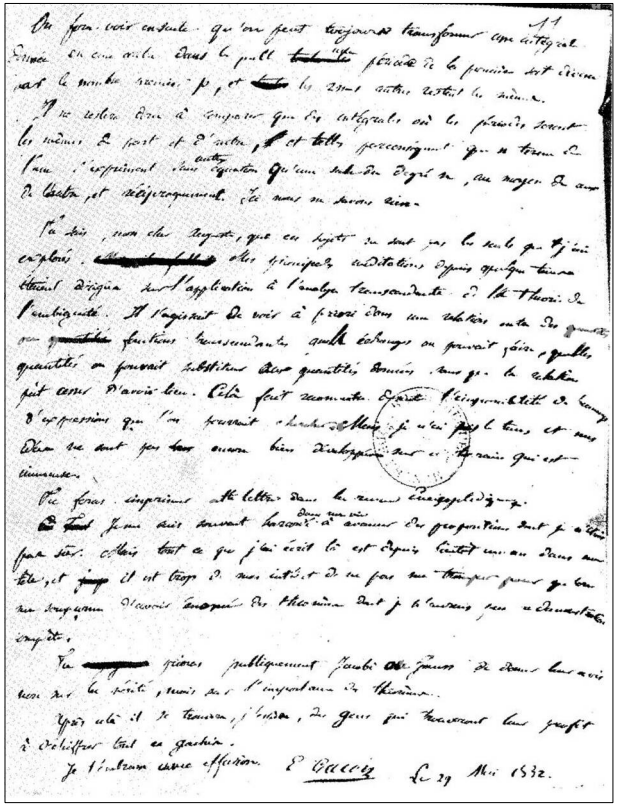
$$\textit{antal additioner} = \lfloor \log_2 n \rfloor + (\textit{popcount}(n) - 1) = O(\log n)$$

$$n = 41 \Rightarrow \lfloor \log_2 41 \rfloor + (3 - 1) = 5 + 2 = 7$$

Évariste Galois (1811-1832)


BEAUCE
MINNE

Brev till vännen Auguste Chevalier



Addition av heltal

$\{ \mathbb{Z}, +, 0 \}$:

$(a + b) + c = a + (b + c)$ *(associativa lagen)*

$a + 0 = 0 + a = a$ *(identitetselement)*

$a + (-a) = (-a) + a = 0$ *(invers)*

Addition av rationella tal

{ ℚ, +, 0 }:

(a + b) + c = a + (b + c) *(associativa lagen)*

a + 0 = 0 + a = a *(identitetselement)*

a + (-a) = (-a) + a = 0 *(invers)*

$$Addition: \frac{a}{b}+\frac{c}{d}=\frac{ad+bc}{bd}$$

Multiplikation av rationella tal

$\{ \mathbb{Q}, \times, 1 \}$:

$(a \times b) \times c = a \times (b \times c)$ *(associativa lagen)*

$a \times 1 = 1 \times a = a$ *(identitetselement)*

$a \times a^{-1} = a^{-1} \times a = 1$ *(invers)*

Multiplikation:

$$\frac{a}{b} \times \frac{c}{d} = \frac{ac}{bd}$$

Grupper

{ T, ∘, e } där:

$(a \circ b) \circ c = a \circ (b \circ c)$ *(associativa lagen)*

$a \circ e = e \circ a = a$ *(identitetselement)*

$a \circ a^{-1} = a^{-1} \circ a = e$ *(invers)*

```
{ int, +, 0 }

{ bool, xor, False }
```

xor :
False *xor* False = False
False *xor* True = True
True *xor* False = True
True *xor* True = False

Semigrupper

{ T, ∘ } där:

(a ∘ b) ∘ c = a ∘ (b ∘ c) *(associativa lagen)*

```
{ int, + } { int, * } { int, min } { int, max }

{ bool, xor } { bool, and } { bool, or }

{ str, + }
```

Krav på a:s typ

```
def multiply(n, a):
    if n == 1: return a
    result = multiply(half(n), a + a)
    if odd(n): result = result + a
    return result
```

Från multiplikation till potenser

```
def power(a, n):
    if n == 1: return a
    result = multiply(a * a, half(n))
    if odd(n): result = result * a
    return result
```

En generisk algoritm

```
def power_semigroup(a, n, bin_op):
    if n == 1: return a
    result = multiply(bin_op(a, a), half(n))
    if odd(n): result = bin_op(result, a)
    return result
```

Matrismultiplikation

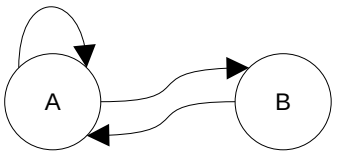
$$\begin{bmatrix} 0 & 1 \\ 1 & 2 \end{bmatrix} \times \begin{bmatrix} 1 & 1 \\ 2 & 3 \end{bmatrix} = \begin{bmatrix} 0 \cdot 1 + 1 \cdot 2 & 0 \cdot 1 + 1 \cdot 3 \\ 1 \cdot 1 + 2 \cdot 2 & 1 \cdot 1 + 2 \cdot 3 \end{bmatrix} = \begin{bmatrix} 2 & 3 \\ 5 & 7 \end{bmatrix}$$

Multiplikation av kvadratiska matriser i Python

```
def multiply_square_matrices(a, b):
    ns = range(len(a))
    result = [[0 for col in ns] for row in ns]
    for i in ns:
        for j in ns:
            for k in ns:
                result[i][j] = result[i][j] + a[i][k] * b[k][j]
    return result


def power_square_matrix(a, n):
    return power_semigroup(a, n, multiply_square_matrices)
```

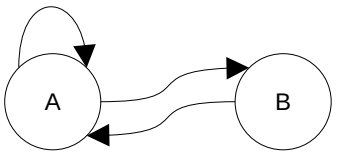
Grafer och matriser

	A	B
A	1	1
B	1	0



Grafer och matriser

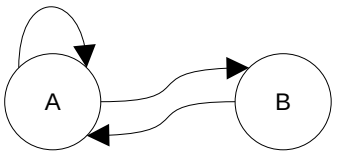
	A	B
A	1	1
B	1	0



$$\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^1 = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$$

Grafer och matriser

	A	B
A	1	1
B	1	0

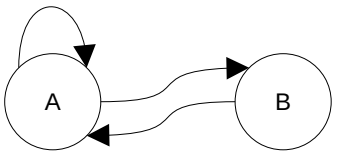


$$\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^1 = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^2 = \begin{bmatrix} 2 & 1 \\ 1 & 1 \end{bmatrix}$$

Grafer och matriser

	A	B
A	1	1
B	1	0



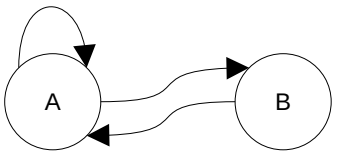
$$\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^1 = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^2 = \begin{bmatrix} 2 & 1 \\ 1 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^3 = \begin{bmatrix} 3 & 2 \\ 2 & 1 \end{bmatrix}$$

Grafer och matriser

	A	B
A	1	1
B	1	0



$$\begin{bmatrix}1 & 1\\1 & 0\end{bmatrix}^1=\begin{bmatrix}1 & 1\\1 & 0\end{bmatrix}$$

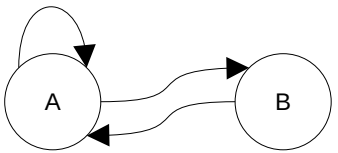
$$\begin{bmatrix}1 & 1\\1 & 0\end{bmatrix}^2=\begin{bmatrix}2 & 1\\1 & 1\end{bmatrix}$$

$$\begin{bmatrix}1 & 1\\1 & 0\end{bmatrix}^3=\begin{bmatrix}3 & 2\\2 & 1\end{bmatrix}$$

$$\begin{bmatrix}1 & 1\\1 & 0\end{bmatrix}^4=\begin{bmatrix}5 & 3\\3 & 2\end{bmatrix}$$

Grafer och matriser

	A	B
A	1	1
B	1	0



$$\begin{bmatrix}1 & 1\\1 & 0\end{bmatrix}^1=\begin{bmatrix}1 & 1\\1 & 0\end{bmatrix}$$

$$\begin{bmatrix}1 & 1\\1 & 0\end{bmatrix}^2=\begin{bmatrix}2 & 1\\1 & 1\end{bmatrix}$$

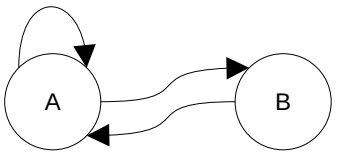
$$\begin{bmatrix}1 & 1\\1 & 0\end{bmatrix}^3=\begin{bmatrix}3 & 2\\2 & 1\end{bmatrix}$$

$$\begin{bmatrix}1 & 1\\1 & 0\end{bmatrix}^4=\begin{bmatrix}5 & 3\\3 & 2\end{bmatrix}$$

$$\begin{bmatrix}1 & 1\\1 & 0\end{bmatrix}^5=\begin{bmatrix}8 & 5\\5 & 3\end{bmatrix}$$

Grafer och matriser

	A	B
A	1	1
B	1	0



$$\begin{bmatrix}1 & 1\\1 & 0\end{bmatrix}^1=\begin{bmatrix}1 & 1\\1 & 0\end{bmatrix}$$

$$\begin{bmatrix}1 & 1\\1 & 0\end{bmatrix}^2=\begin{bmatrix}2 & 1\\1 & 1\end{bmatrix}$$

$$\begin{bmatrix}1 & 1\\1 & 0\end{bmatrix}^3=\begin{bmatrix}3 & 2\\2 & 1\end{bmatrix}$$

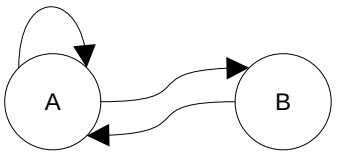
$$\begin{bmatrix}1 & 1\\1 & 0\end{bmatrix}^4=\begin{bmatrix}5 & 3\\3 & 2\end{bmatrix}$$

$$\begin{bmatrix}1 & 1\\1 & 0\end{bmatrix}^5=\begin{bmatrix}8 & 5\\5 & 3\end{bmatrix}$$

$$\begin{bmatrix}1 & 1\\1 & 0\end{bmatrix}^6=\begin{bmatrix}13 & 8\\8 & 5\end{bmatrix}$$

Grafer och matriser

	A	B
A	1	1
B	1	0



$$\begin{bmatrix}1 & 1\\1 & 0\end{bmatrix}^1=\begin{bmatrix}1 & 1\\1 & 0\end{bmatrix}$$

$$\begin{bmatrix}1 & 1\\1 & 0\end{bmatrix}^2=\begin{bmatrix}2 & 1\\1 & 1\end{bmatrix}$$

$$\begin{bmatrix}1 & 1\\1 & 0\end{bmatrix}^3=\begin{bmatrix}3 & 2\\2 & 1\end{bmatrix}$$

$$\begin{bmatrix}1 & 1\\1 & 0\end{bmatrix}^4=\begin{bmatrix}5 & 3\\3 & 2\end{bmatrix}$$

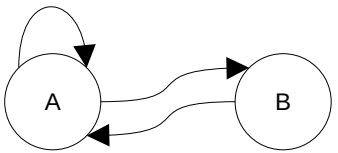
$$\begin{bmatrix}1 & 1\\1 & 0\end{bmatrix}^5=\begin{bmatrix}8 & 5\\5 & 3\end{bmatrix}$$

$$\begin{bmatrix}1 & 1\\1 & 0\end{bmatrix}^6=\begin{bmatrix}13 & 8\\8 & 5\end{bmatrix}$$

$$\begin{bmatrix}1 & 1\\1 & 0\end{bmatrix}^7=\begin{bmatrix}21 & 13\\13 & 8\end{bmatrix}$$

Grafer och matriser

	A	B
A	1	1
B	1	0



$$\begin{bmatrix}1 & 1\\1 & 0\end{bmatrix}^{1}=\begin{bmatrix}1 & 1\\1 & 0\end{bmatrix}$$

$$\begin{bmatrix}1 & 1\\1 & 0\end{bmatrix}^{2}=\begin{bmatrix}2 & 1\\1 & 1\end{bmatrix}$$

$$\begin{bmatrix}1 & 1\\1 & 0\end{bmatrix}^{3}=\begin{bmatrix}3 & 2\\2 & 1\end{bmatrix}$$

$$\begin{bmatrix}1 & 1\\1 & 0\end{bmatrix}^{4}=\begin{bmatrix}5 & 3\\3 & 2\end{bmatrix}$$

$$\begin{bmatrix}1 & 1\\1 & 0\end{bmatrix}^{5}=\begin{bmatrix}8 & 5\\5 & 3\end{bmatrix}$$

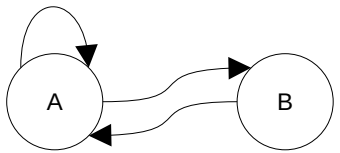
$$\begin{bmatrix}1 & 1\\1 & 0\end{bmatrix}^{6}=\begin{bmatrix}13 & 8\\8 & 5\end{bmatrix}$$

$$\begin{bmatrix}1 & 1\\1 & 0\end{bmatrix}^{7}=\begin{bmatrix}21 & 13\\13 & 8\end{bmatrix}$$

$$\begin{bmatrix}1 & 1\\1 & 0\end{bmatrix}^{8}=\begin{bmatrix}34 & 21\\21 & 13\end{bmatrix}$$

Fibonacci-matriser

	A	B
A	1	1
B	1	0



$0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, \ldots$

$$\begin{bmatrix}1 & 1\\ 1 & 0\end{bmatrix}^1 = \begin{bmatrix}1 & 1\\ 1 & 0\end{bmatrix}$$
$$\begin{bmatrix}1 & 1\\ 1 & 0\end{bmatrix}^2 = \begin{bmatrix}2 & 1\\ 1 & 1\end{bmatrix}$$
$$\begin{bmatrix}1 & 1\\ 1 & 0\end{bmatrix}^3 = \begin{bmatrix}3 & 2\\ 2 & 1\end{bmatrix}$$
$$\begin{bmatrix}1 & 1\\ 1 & 0\end{bmatrix}^4 = \begin{bmatrix}5 & 3\\ 3 & 2\end{bmatrix}$$
$$\begin{bmatrix}1 & 1\\ 1 & 0\end{bmatrix}^5 = \begin{bmatrix}8 & 5\\ 5 & 3\end{bmatrix}$$
$$\begin{bmatrix}1 & 1\\ 1 & 0\end{bmatrix}^6 = \begin{bmatrix}13 & 8\\ 8 & 5\end{bmatrix}$$
$$\begin{bmatrix}1 & 1\\ 1 & 0\end{bmatrix}^7 = \begin{bmatrix}21 & 13\\ 13 & 8\end{bmatrix}$$
$$\begin{bmatrix}1 & 1\\ 1 & 0\end{bmatrix}^8 = \begin{bmatrix}34 & 21\\ 21 & 13\end{bmatrix}$$

$$Fibonacci-matriser: \begin{bmatrix}F_{n+1} & F_n\\ F_n & F_{n-1}\end{bmatrix}$$

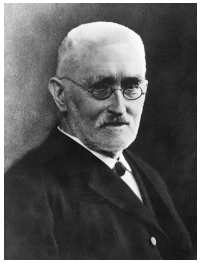
$$antal\ additioner\ och\ multiplikationer = O(\log n)$$


Erlangen
Universitätstraße

Emmy Noether (1882 - 1935)

Universitetet i Göttingen

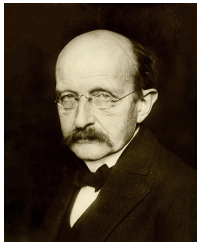
Carl Friedrich Gauss



Richard Dedekind

David Hilbert

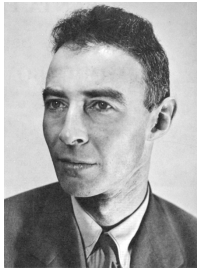
Felix Klein



Max Planck

Werner Heisenberg

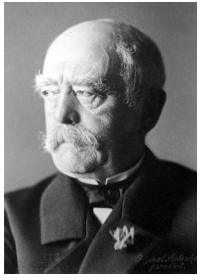
Bernhard Riemann



Robert Oppenheimer

John von Neumann

Bröderna Grimm



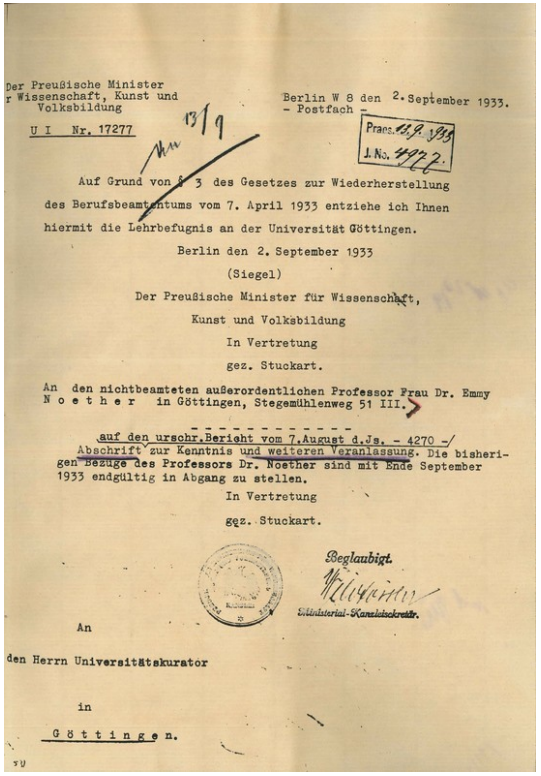
Otto Von Bismarck

J.P. Morgan

Max Weber

“Noethers pojkar”

Flykt till USA



Der Preußische Minister
r Wissenschaft, Kunst und
Volksbildung

Berlin W 8 den 2. September 1933.
- Postfach -

U I Nr. 17277

13/9

Prees. 13. 9. 1933
1 No. 4922.

Auf Grund von § 3 des Gesetzes zur Wiederherstellung des Berufsbeamtentums vom 7. April 1933 entziehe ich Ihnen hiermit die Lehrbefugnis an der Universität Göttingen.

Berlin den 2. September 1933

(Siegel)

Der Preußische Minister für Wissenschaft,
Kunst und Volksbildung
In Vertretung
gez. Stuckart.

An den nichtbeamteten außerordentlichen Professor Frau Dr. Emmy Noether in Göttingen, Stegemühlenweg 51 III.

- - - - - - - - - - - - -

auf den urschr. Bericht vom 7. August d. Js. - 4270 -/
Abschrift zur Kenntnis und weiteren Veranlassung. Die bisherigen Bezüge des Professors Dr. Noether sind mit Ende September 1933 endgültig in Abgang zu stellen.

In Vertretung
gez. Stuckart.

Beglaubigt.
Ministerial-Kanzleisekretär.

An
den Herrn Universitätskurator
in
Göttingen.

Semiringar

{ T, ⊕, 0_S, ⊗, 1_S } där:

{ T, ⊕, 0_S }	*(kommutativ monoid)*
{ T, ⊗, 1_S }	*(monoid)*
$0_S \otimes a = a \otimes 0_S = 0_S$	*(absorbtion)*
$a \otimes (b \oplus c) = a \otimes b \oplus a \otimes c$	*(distributiva lagen)*
$(b \oplus c) \otimes a = b \otimes a \oplus c \otimes a$	*(distributiva lagen)*

`{ int, +, 0, *, 1 }` `{ bool, or, False, and, True }`

Nollpotenser

```
def power_monoid(a, n, bin_op, identity_element):
    if (n == 0): return identity_element
    return power_semigroup(a, n, bin_op)
```

Nollpotens för heltal:

$$1 \times a = a \Rightarrow a^0 = 1$$

Multiplikation av kvadratiska matriser i Python

```
def multiply_square_matrices(a, b):
    ns = range(len(a))
    result = [[0 for col in ns] for row in ns]
    for i in ns:
        for j in ns:
            for k in ns:
                result[i][j] = result[i][j] + a[i][k] * b[k][j]
    return result


def power_square_matrix(a, n):
    ns = range(len(a))
    identity_matrix = [[1 if row == col else 0 for col in ns] for row in ns]
    return power_monoid(a, n, multiply_square_matrices, identity_matrix)
```

Identitetselement för matriser :

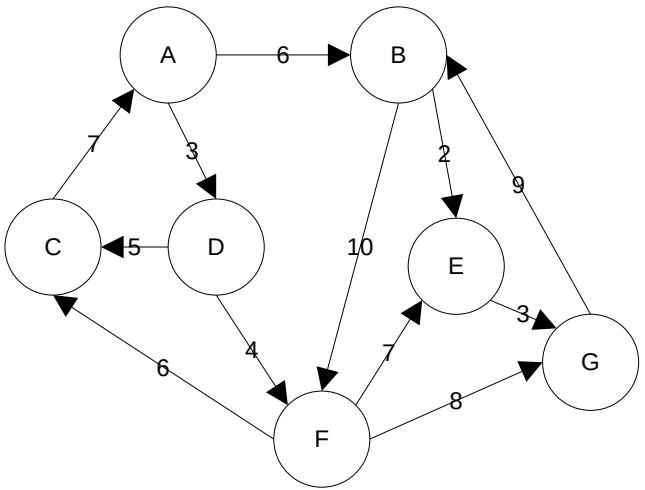
$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Matrismultiplikation över semiringar i Python

```
def multiply_square_matrices_semiring(a, b, add_op, zero, mul_op):
    ns = range(len(a))
    result = [[zero for col in ns] for row in ns]
    for i in ns:
        for j in ns:
            for k in ns:
                result[i][j] = add_op(result[i][j], mul_op(a[i][k], b[k][j]))
    return result


def power_square_matrix_semiring(a, n, add_op, zero, mul_op, one):
    def multiply(a, b):
        return multiply_square_matrices_semiring(a, b, add_op, zero, mul_op)
    ns = range(len(a))
    identity_matrix = [[one if row == col else zero for col in ns] for row in ns]]
    return power_monoid(a, n, multiply, identity_matrix)
```

Kortaste vägen mellan två punkter

	A	B	C	D	E	F	G
A	0	6	∞	3	∞	∞	∞
B	∞	0	∞	∞	2	10	∞
C	7	∞	0	∞	∞	∞	∞
D	∞	∞	5	0	∞	4	∞
E	∞	∞	∞	∞	0	∞	3
F	∞	∞	6	∞	7	0	8
G	∞	9	∞	∞	∞	∞	0



Kortaste vägen mellan två punkter

	A	B	C	D	E	F	G
A	0	6	∞	3	∞	∞	∞
B	∞	0	∞	∞	2	10	∞
C	7	∞	0	∞	∞	∞	∞
D	∞	∞	5	0	∞	4	∞
E	∞	∞	∞	∞	0	∞	3
F	∞	∞	6	∞	7	0	8
G	∞	9	∞	∞	∞	∞	0

$$\begin{bmatrix} 0 & 6 & \infty & 3 & \infty & \infty & \infty \\ \infty & 0 & \infty & \infty & 2 & 10 & \infty \\ 7 & \infty & 0 & \infty & \infty & \infty & \infty \\ \infty & \infty & 5 & 0 & \infty & 4 & \infty \\ \infty & \infty & \infty & \infty & 0 & \infty & 3 \\ \infty & \infty & 6 & \infty & 7 & 0 & 8 \\ \infty & 9 & \infty & \infty & \infty & \infty & 0 \end{bmatrix}^{6}$$

```
{ int, min, float('inf'), +, 0 }
```

Tropisk semiring:
+ → *min*
0 → ∞
× → +
1 → 0

Kortaste vägen mellan två punkter

	A	B	C	D	E	F	G
A	0	6	8	3	8	7	11
B	23	0	16	26	2	10	5
C	7	13	0	10	15	14	18
D	12	18	5	0	11	4	12
E	35	12	28	38	0	22	3
F	13	17	6	16	7	0	8
G	32	9	25	35	11	19	0

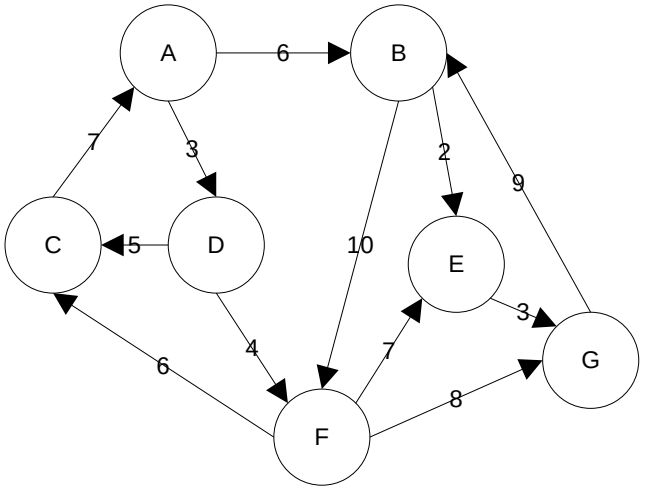
$$\begin{bmatrix} 0 & 6 & 8 & 3 & 8 & 7 & 11 \\ 23 & 0 & 16 & 26 & 2 & 10 & 5 \\ 7 & 13 & 0 & 10 & 15 & 14 & 18 \\ 12 & 18 & 5 & 0 & 11 & 4 & 12 \\ 35 & 12 & 28 & 38 & 0 & 22 & 3 \\ 13 & 17 & 6 & 16 & 7 & 0 & 8 \\ 32 & 9 & 25 & 35 & 11 & 19 & 0 \end{bmatrix}$$

```
{ int, min, float('inf'), +, 0 }
```

Tropisk semiring:
+ → *min*
0 → ∞
× → +
1 → 0

Kortaste vägen mellan två punkter

	A	B	C	D	E	F	G
A	0	6	8	3	8	7	11
B	23	0	16	26	2	10	5
C	7	13	0	10	15	14	18
D	12	18	5	0	11	4	12
E	35	12	28	38	0	22	3
F	13	17	6	16	7	0	8
G	32	9	25	35	11	19	0



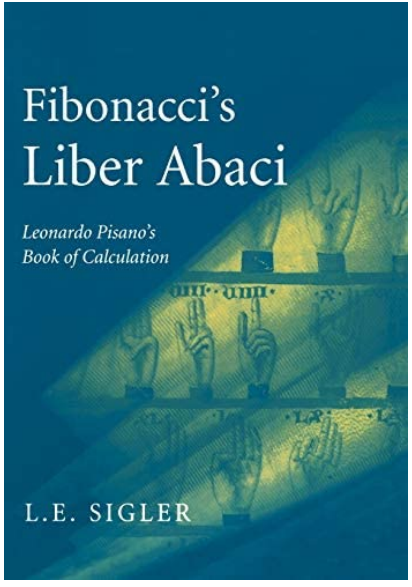
Kortaste vägar med tropiska semiringar i Python

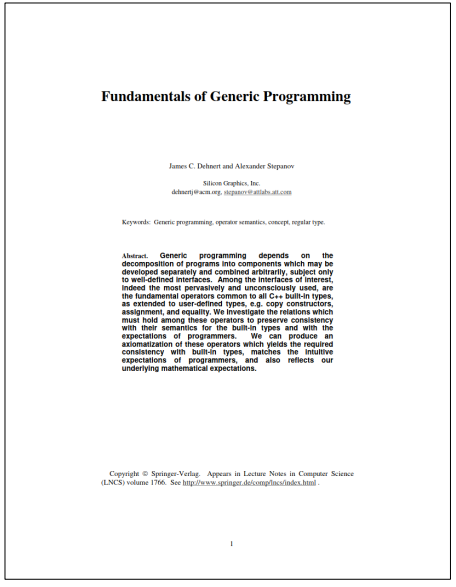
```
import operator

def multiply_square_matrices_tropical_semiring(a, b):
    return multiply_square_matrices_semiring(a, b, min, float('Inf'), operator.add)

def shortest_paths(graph):
    n = len(graph) - 1
    return power_monoid(graph, n, multiply_square_matrices_tropical_semiring, graph)
```

Slutsatser

- Börja skriva kod för konkreta problem, abstrahera sedan bort oviktiga egenskaper hos typerna som används
- Egenskaper hos specifika typer och värden möjliggör ofta optimeringar och alternativa algoritmer
- Kräver ett programmeringsspråk med stöd för funktioner som tar argument av olika typer (dynamisk typning, abstrakta interface, templates, generics, makron, protokoll...)
- Mycket att lära av historien, många personer att tacka för att vi är där vi är

Källor och kod





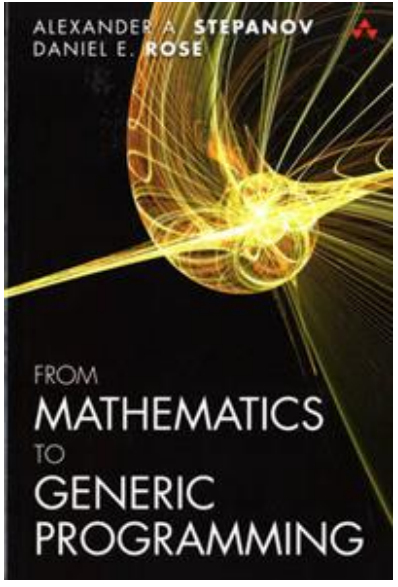
Fundamentals of Generic Programming

James C. Dehnert and Alexander Stepanov

Silicon Graphics, Inc.
dehnert@acm.org, stepanov@attlabs.att.com

Keywords: Generic programming, operator semantics, concept, regular type.

Abstract. **Generic programming depends on the decomposition of programs into components which may be developed separately and combined arbitrarily, subject only to well-defined interfaces. Among the interfaces of interest, indeed the most pervasively and unconsciously used, are the fundamental operators common to all C++ built-in types, as extended to user-defined types, e.g. copy constructors, assignment, and equality. We investigate the relations which must hold among these operators to preserve consistency with their semantics for the built-in types and with the expectations of programmers. We can produce an axiomatization of these operators which yields the required consistency with built-in types, matches the intuitive expectations of programmers, and also reflects our underlying mathematical expectations.**



Fun with Semirings

A functional pearl on the abuse of linear algebra

Stephen Dolan
Computer Laboratory, University of Cambridge
stephen.dolan@cl.cam.ac.uk

Abstract

Describing a problem using classical linear algebra is a very well-known problem-solving technique. If your question can be formulated as a question about real or complex matrices, then the answer can often be found by standard techniques.

It's less well-known that very similar techniques still apply where instead of real or complex numbers we have *a closed semiring*, which is a structure with some analogue of addition and multiplication that need not support subtraction or division.

We define a typeclass in Haskell for describing closed semirings, and implement a few functions for manipulating matrices and polynomials over them. We then show how these functions can be used to calculate transitive closures, find shortest or longest or widest paths in a graph, analyse the data flow of imperative programs, optimally pack knapsacks, and perform discrete event simulations, all by just providing an appropriate underlying closed semiring.

Categories and Subject Descriptors D.3.1 [*Programming Techniques*]: Applicative (functional) Programming; G.2.2 [*Discrete Mathematics*]: Graph Theory—graph algorithms

Keywords closed semirings; transitive closure; linear systems; shortest paths

1. Introduction

Linear algebra provides an incredibly powerful problem-solving toolbox. A great many problems in computer graphics and vision, machine learning, signal processing and many other areas can be solved by simply expressing the problem as a system of linear equations and solving using standard techniques.

Linear algebra is defined abstractly in terms of fields, of which the real and complex numbers are the most familiar examples. Fields are sets equipped with some notion of addition and multiplication as well as negation and reciprocals.

Many discrete mathematical structures commonly encountered in computer science do not have sensible notions of negation. Booleans, sets, graphs, regular expressions, imperative programs, datatypes and various other structures can all be given natural notions of product (interpreted variously as intersection, sequencing or conjunction) and sum (union, choice or disjunction), but generally lack negation or reciprocals.

Such structures, having addition and multiplication (which distribute in the usual way) but not in general negation or reciprocals, are called semirings. Many structures specifying sequential actions can be thought of as semirings, with multiplication as sequencing and addition as choice. The distributive law then states, intuitively, a followed by a choice between b and c is the same as a choice between a followed by b and a followed by c.

Plain semirings are a very weak structure. We can find many examples of them in the wild, but unlike fields which provide the toolbox of linear algebra, there isn't much we can do with something knowing only that it is a semiring.

However, we can build some useful tools by introducing the *closed semiring*, which is a semiring equipped with an extra operation called *closure*. With the intuition of multiplication as sequencing and addition as choice, closure can be interpreted as iteration. As we see in the following sections, it is possible to use something akin to Gaussian elimination on an arbitrary closed semiring, giving us a means of solving certain "linear" equations over any structure with suitable notions of sequencing, choice and iteration. First, though, we need to define the notion of semiring more precisely.

2. Semirings

We define a semiring formally as consisting of a set R, two distinguished elements of R named 0 and 1, and two binary operations $+$ and $\cdot$, satisfying the following relations for any $a, b, c \in R$:

$$a + b = b + a$$
$$a + (b + c) = (a + b) + c$$
$$a + 0 = a$$
$$a \cdot (b \cdot c) = (a \cdot b) \cdot c$$
$$a \cdot 0 = 0 \cdot a = 0$$
$$a \cdot 1 = 1 \cdot a = a$$
$$a \cdot (b + c) = a \cdot b + a \cdot c$$
$$(a + b) \cdot c = a \cdot c + b \cdot c$$

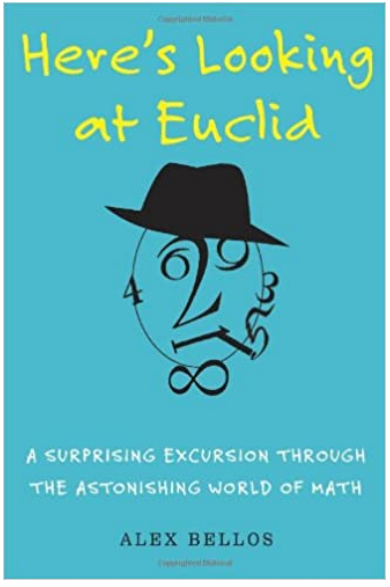
We often write $a \cdot b$ as ab, and $a \cdot a \cdot a$ as a^3.

Our focus will be on *closed semirings* [12], which are semirings with an additional operation called *closure* (denoted *) which satisfies the axioms:

$$a^* = 1 + a \cdot a^* = 1 + a^* \cdot a$$

If we have an *affine map* $x \mapsto ax + b$ in some closed semiring, then $x = a^*b$ is a fixpoint, since $a^*b = (aa^* + 1)b = a(a^*b) + b$. So, a closed semiring can also be thought of as a semiring where affine maps have fixpoints.

The definition of a semiring translates neatly to Haskell:



https://github.com/petter-holmberg/talks