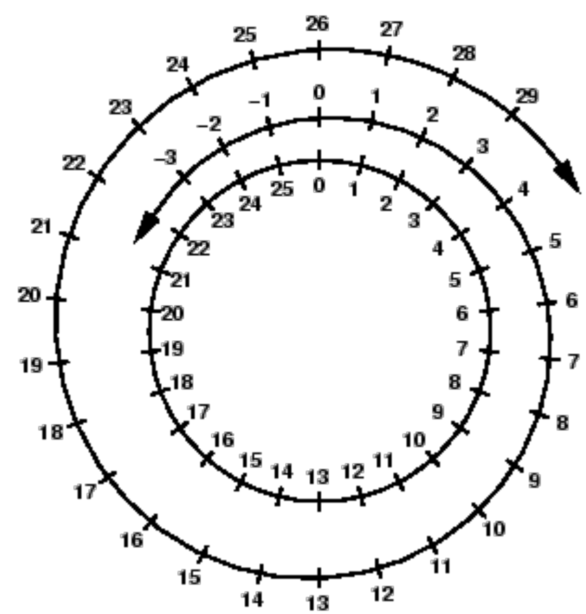
15-251
Great Theoretical Ideas in Computer Science

Lecture 21:
Computational Arithmetic

November 10th, 2015



This week

Computational arithmetic
(in particular, modular arithmetic)

+

Cryptography
(in particular, "public-key" cryptography)

Main goal of this lecture

Goal:

Understanding modular arithmetic: **theory + algorithms**

Why:

1. When we do addition or multiplication,
   the universe is infinite (e.g. $\mathbb{Z}, \mathbb{Q}, \mathbb{R}$.)

   Sometimes we prefer to restrict ourselves to a finite universe (e.g. the modular universe).

2. Some hard-to-do arithmetic operations in $\mathbb{Z}$ or $\mathbb{Q}$ is easy in the modular universe.

3. Some easy-to-do arithmetic operations in $\mathbb{Z}$ or $\mathbb{Q}$ seem to be hard in the modular universe.

   And this is great for cryptography applications!

Main goal of this lecture

Modular Universe

- How to view the elements of the universe?

- How to do basic operations:

  > addition

  > subtraction

  > multiplication

  > division

  > exponentiation

  > taking roots

  > logarithm

theory

+

algorithms

(efficient (?))

The plan

Start with algorithms on good old integers.

Then move to the modular universe.

Integers

Algorithms on numbers involve **BIG** numbers.

3618502788666131106986593281521497110455743021169260358536775932020762686101
7237846234873269807102970128874356021481964232857782295671675021393065473695
3943653222082116941587830769649826310589717739181525033220266350650989268038
3194839273881505432422077179121838888281996148408052302196889866637200606252
6501310964926475205090003984176122058711164567946559044971683604424076996342
7183046544798021168297013490774140090476348290671822743961203698142307099664
3455133414637616824423860107889741058131271306226214208636008224651510961018
9789006815067664901594246966730927620844732714004599013904409378141724958467
7228950143608277369974692883195684314361862929679227167524851316077587207648
7845058367231603173079817471417519051357029671991152963580412838184841733782

Integers

$B =$ 5693030020523999993479642904621911725098567020556258102766251487234031094429

$B \approx 5.7 \times 10^{75}$ (5.7 quattorvigintillion)

B is roughly the number of atoms in the universe or the age of the universe in Planck time units.

Definition: $\mathrm{len}(B) = \#$ bits to write B

$$\approx \log_2 B$$

For $B =$ 5693030020523999993479642904621911725098567020556258102766251487234031094429

$$\mathrm{len}(B) = 251$$

(for crypto purposes, this is way too small)

Integers: Arithmetic

In general, arithmetic on numbers is not free!

Think of algorithms as performing string-manipulation.

Think of adding two numbers up yourself.
(the longer the numbers, the longer it will take)

$$
\begin{array}{r}
36185027886661311069865932815214971104 \\
+\quad 65743021169260358536775932020762686101 \\
\hline
101928049055921669606641864835977657205
\end{array}
$$

The number of steps is measured with respect to the <u>length of the input numbers</u>.

Integers: Addition

$$
\begin{array}{rrl}
 & 36185027886661311069865932815214971104 & A \\
+ & 65743021169260358536775932020762686101 & B \\
\hline
 & 101928049055921669606641864835977657205 & C
\end{array}
$$

Grade school addition is linear time:

if $\operatorname{len}(A), \operatorname{len}(B) \leq n$

number of steps to produce C is $O(n)$

Integers: Multiplication

```
                 36185027886661311069865932815214971104   A
               x                       5932020762686101   B
                -----------------------------------------
                XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
               XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
              XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
             XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
            XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
           XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
          XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
         XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
        XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
       XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
      XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
     XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
    XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
   XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
  XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
-------------------------------------------------------
214650336722050463946651358202698404452609868137425504   C
```

 # steps: $O(\mathrm{len}(A) \cdot \mathrm{len}(B))$

$= O(n^2)$ **if** $\mathrm{len}(A), \mathrm{len}(B) \leq n$

Integers: Division

```
                          6099949635084593037586  Q
B  5932020762686101 | 36185027886661311069865932815214971104  A
                      XXXXXXXXXXXXXXXXX
                       XXXXXXXXXXXXXXXXX
                        XXXXXXXXXXXXXXXXX
                         XXXXXXXXXXXXXXXXX
                          XXXXXXXXXXXXXXXXX
                           XXXXXXXXXXXXXXXXX
                            XXXXXXXXXXXXXXXXX
                             XXXXXXXXXXXXXXXXX
                              XXXXXXXXXXXXXXXXX
                               XXXXXXXXXXXXXXXXX
                                XXXXXXXXXXXXXXXXX
                                 XXXXXXXXXXXXXXXXX
                                  XXXXXXXXXXXXXXXXX
                                   XXXXXXXXXXXXXXXXX
                                    XXXXXXXXXXXXXXXXX
                                     XXXXXXXXXXXXXXXXX
                                      XXXXXXXXXXXXXXXXX
                                       XXXXXXXXXXXXXXXXX
                                        XXXXXXXXXXXXXXXXX
                                      ---------------------
                                        3960087002178918  R
```

$$A = Q \cdot B + R$$

$$\boxed{R = A \bmod B}$$

 # steps: $O(\mathrm{len}(A) \cdot \mathrm{len}(B))$

Integers: Exponentiation

Given as input B, compute 2^B.

If

$B =$ 5693030020523999993479642904621911725098567020556258102766251487234031094429

$\text{len}(B) = 251$

but $\text{len}(2^B)$ ~ 5.7 quattorvigintillion

(output length exceeds number of particles in the universe)

exponential in input length

Integers: Factorization

$A =$ 5693030020523999993479642904621911725098567020556258102766251487234031094429

Goal: find one (non-trivial) factor of A

> **for** B = 2, 3, 4, 5, …
>   test **if** A mod B = 0.

It turns out:

$A =$ 68452332409801603635385895997250919383 x
83167801886452917478124266362673045163

Each factor ≈ age of the universe in Planck time.

worst case: $\sqrt{A}$ iterations.

$$\sqrt{A} = \sqrt{2^{\log_2 A}} = \sqrt{2^{\mathrm{len}(A)}} = 2^{\mathrm{len}(A)/2}$$

exponential in input length

Integers: Factorization

Fastest known algorithm is exponential time!

That turns out to be a good thing:

If there is an efficient algorithm to solve
the factoring problem

can break most cryptographic systems
used on the internet

Integers: Primality testing

Your favorite function from 15-112

```
def isPrime(n):
    if (n < 2):
        return False
    for factor in range(2,n):
        if (n % factor == 0):
            return False
    return True
```

\# iterations: ≈ n

$$n = 2^{\log_2 n} = 2^{\text{len}(n)}$$

exponential in input length

Integers: Primality testing

```
def fasterIsPrime(n):
    if (n < 2):
        return False
    if (n == 2):
        return True
    if (n % 2 == 0):
        return False
    maxFactor = round(n**0.5)
    for factor in range(3,maxFactor+1,2):
        if (n % factor == 0):
            return False
    return True
```

<u>Exercise</u>: Show that this is still exponential time.

Integers: Primality testing

Amazing result from 2002:

There is a poly-time algorithm for primality testing.

Agrawal, Kayal, Saxena

undergraduate students at the time

However, best known implementation is ~ $O(n^6)$ time.

Not feasible when $n = 2048$.

Integers: Primality testing

So that's not what we use in practice.

Everyone uses the Miller-Rabin algorithm (1975).

CMU
Professor

The running time is ~ $O(n^2)$.

It is a Monte Carlo algorithm with tiny error probability
(say $1/2^{300}$)

Integers: Generating a random prime number

Suppose you need an n-bit long random prime number.

repeat:
  let A be a random n-bit number
  test if A is prime

Prime Number Theorem (informal):

About 1/n fraction of n-bit numbers are prime.

$\Longrightarrow$ expected # iterations of the above algorithm ~ $O(n^3)$.

No poly-time deterministic algorithm is known!!

The plan

Start with algorithms on good old integers.

Then move to the modular universe.

Main goal of this lecture

Modular Universe

- How to view the elements of the universe?

- How to do basic operations:

  > addition

  > subtraction

  > multiplication

  > division

  > exponentiation

  > taking roots

  > logarithm

theory
+
algorithms
(efficient (?))

Modular universe: How to view the elements

Hopefully everyone already knows:

Any integer can be reduced mod *N*.

$A \bmod N$ = remainder when you divide A by N

Example

$$N = 5$$

$$
\begin{array}{ccccc|ccccc|cccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & \cdots \\
\downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \bmod 5 \\
0 & 1 & 2 & 3 & 4 & 0 & 1 & 2 & 3 & 4 & 0 & 1 & 2 & \cdots
\end{array}
$$

Modular universe: How to view the elements

We write $A \equiv B \bmod N$ or $A \equiv_N B$

when $A \bmod N = B \bmod N.$

(In this case, we say A is congruent to B modulo N.)

Examples

$$5 \equiv_5 100$$

$$13 \equiv_7 27$$

Exercise

$$A \equiv_N B \iff N \text{ divides } A - B$$

Modular universe: How to view the elements

2 Points of View

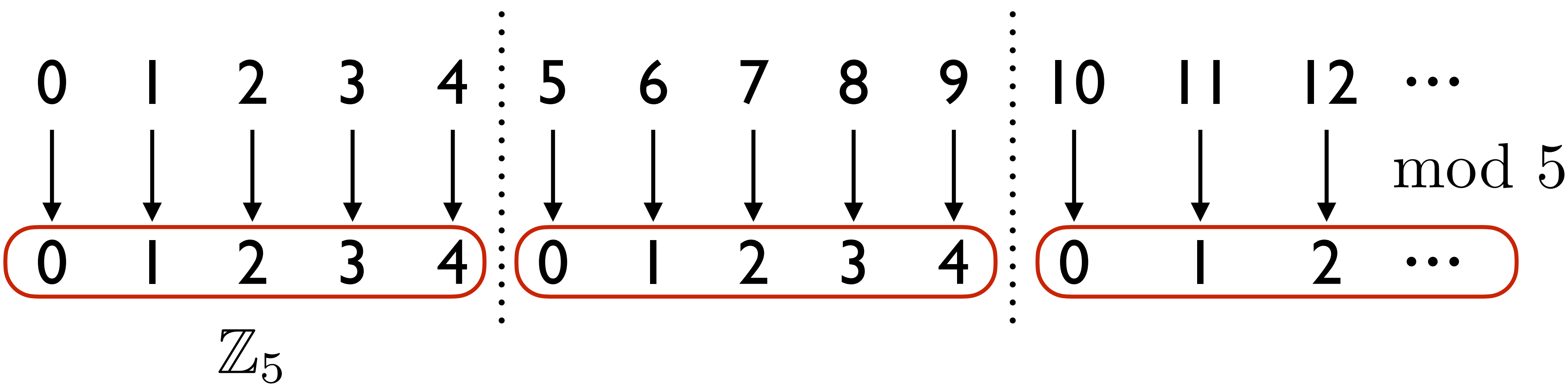
View 1

The universe is $\mathbb{Z}$.

Every element has a "mod *N*" representation.

View 2

The universe is the finite set $\mathbb{Z}_N = \{0, 1, 2, \ldots, N-1\}$.



Modular universe: Addition

Addition plays nice mod *N*

$$\boxed{A} \equiv_N \boxed{B}$$

$$A' \equiv_N B'$$

$$\implies \boxed{A} + A' \equiv_N \boxed{B} + B'$$

$$
\begin{array}{ccccc:ccccc:cccl}
0 & 1 & \boxed{2} & 3 & 4 & 5 & 6 & \boxed{7} & 8 & 9 & 10 & 11 & \boxed{12} & \cdots \\
\downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \bmod 5 \\
0 & 1 & 2 & 3 & 4 & 0 & 1 & 2 & 3 & 4 & 0 & 1 & 2 & \cdots
\end{array}
$$

$\bigcirc$ + $\square$ is always the same mod *N*

Modular universe: Addition

Addition table for $\mathbb{Z}_5$

+	**0**	**1**	**2**	**3**	**4**
0	0	1	2	3	4
1	1	2	3	4	0
2	2	3	4	0	1
3	3	4	0	1	2
4	4	0	1	2	3

0 is called the (additive) identity: $\mathbf{0} + A = A + \mathbf{0} = A$

for any A

Modular universe: Subtraction

How about subtraction in $\mathbb{Z}_N$?

What does $A - B$ mean?

It is actually addition in disguise: $A + (-B)$

Then what does $-B$ mean?

Given any B, we define $-B$ to be the number in $\mathbb{Z}_N$ such that $B + (-B) = 0$.

Modular universe: Subtraction

Addition table for $\mathbb{Z}_5$

+	0	1	2	3	4
0	0	1	2	3	4
1	1	2	3	4	0
2	2	3	4	0	2
3	3	4	0	1	2
4	4	0	1	2	3

$$-0 = 0$$
$$-1 = 4$$
$$-2 = 3$$
$$-3 = 2$$
$$-4 = 1$$

Modular universe: Subtraction

Addition table for $\mathbb{Z}_5$

+	0	1	2	3	4
0	0	1	2	3	4
1	1	2	3	4	0
2	2	3	4	0	2
3	3	4	0	1	2
4	4	0	1	2	3

Note:

For every $A \in \mathbb{Z}_N$, $-A$ exists.

Why? $-A = N - A$

This implies:

A row contains distinct elements.

i.e. every row is a permutation of $\mathbb{Z}_N$.

Fix row A

$$\underset{\text{row}}{A} + \underset{\text{col}}{B} = \underset{\text{row}}{A} + \underset{\text{col}}{B'} \implies \underset{\text{same col}}{B = B'}$$

Modular universe: Multiplication

Multiplication plays nice mod *N*

$$\boxed{A} \equiv_N \boxed{B}$$

$$A' \equiv_N B'$$

$$\Longrightarrow \quad \boxed{A} \cdot A' \equiv_N \boxed{B} \cdot B'$$

0	1	$\boxed{2}$	3	4	5	6	$\boxed{7}$	8	9	10	11	$\boxed{12}$	$\cdots$	
↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓		mod 5
0	1	2	3	4	0	1	2	3	4	0	1	2	$\cdots$	

$\boxed{} \cdot \bigcirc$ is always the same mod *N*

Modular universe: Multiplication

Multiplication table for $\mathbb{Z}_5$

$\cdot$	**0**	**1**	**2**	**3**	**4**
0	0	0	0	0	0
1	0	1	2	3	4
2	0	2	4	1	3
3	0	3	1	4	2
4	0	4	3	2	1

1 is called the (multiplicative) identity: $1 \cdot A = A \cdot 1 = A$
for any A

Modular universe: Division

How about division in $\mathbb{Z}_N$?

What does $A \div B$ mean?

It is actually multiplication in disguise: $A \cdot \dfrac{1}{B} = A \cdot B^{-1}$

Then what does B^{-1} mean?

Given any B, we define B^{-1} to be the number in $\mathbb{Z}_N$ such that $B \cdot B^{-1} = 1$.

Modular universe: Division

Multiplication table for $\mathbb{Z}_5$

$\cdot$	**0**	**1**	**2**	**3**	**4**
0	0	0	0	0	0
1	0	1	2	3	4
2	0	2	4	1	3
3	0	3	1	4	2
4	0	4	3	2	1

$$0^{-1} = \text{undefined}$$
$$1^{-1} = 1$$
$$2^{-1} = 3$$
$$3^{-1} = 2$$
$$4^{-1} = 4$$

Modular universe: Division

Multiplication table for $\mathbb{Z}_6$

$\cdot$	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	1	2	3	4	5
2	0	2	4	0	2	4
3	0	3	0	3	0	3
4	0	4	2	0	4	2
5	0	5	4	3	2	1

$0^{-1} = \text{undefined}$

$1^{-1} = 1$

$2^{-1} = \text{undefined}$

$3^{-1} = \text{undefined}$

$4^{-1} = \text{undefined}$

$5^{-1} = 5$

WTF?

Modular universe: Division

Multiplication table for $\mathbb{Z}_7$

•	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6
2	0	2	4	6	1	3	5
3	0	3	6	2	5	1	4
4	0	4	1	5	2	6	3
5	0	5	3	1	6	4	2
6	0	6	5	4	3	2	1

Every number except 0 has a multiplicative inverse.

Modular universe: Division

Multiplication table for $\mathbb{Z}_8$

•	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7
2	0	2	4	6	0	2	4	6
3	0	3	6	1	4	7	2	5
4	0	4	0	4	0	4	0	4
5	0	5	2	7	4	1	6	3
6	0	6	4	2	0	6	4	2
7	0	7	6	5	4	3	2	1

$\{1, 3, 5, 7\}$ have inverses. Others don't.

Modular universe: Division

Fact: $A^{-1} \in \mathbb{Z}_N$ exists if and only if $\gcd(A, N) = 1.$

$\gcd(a, b)$ = greatest common divisor of a and b.

Examples:

$$\gcd(12, 18) = 6$$

$$\gcd(13, 9) = 1$$

$$\gcd(1, a) = 1 \quad \forall a$$

$$\gcd(0, a) = a \quad \forall a$$

If $\gcd(a, b) = 1$, we say a and b are relatively prime.

Modular universe: Division

Fact: $A^{-1} \in \mathbb{Z}_N$ exists if and only if $\gcd(A, N) = 1.$

Definition: $\mathbb{Z}_N^* = \{A \in \mathbb{Z}_N : \gcd(A, N) = 1\}.$

Definition: $\varphi(N) = |\mathbb{Z}_N^*|$

Note that $\mathbb{Z}_N^*$ is "closed" under multiplication,

i.e., $A, B \in \mathbb{Z}_N^* \implies AB \in \mathbb{Z}_N^*$

(Why?)

Modular universe: Division

$$\mathbb{Z}_5^*$$

•	0	1	2	3	4
0	0	0	0	0	0
1	0	1	2	3	4
2	0	2	4	1	3
3	0	3	1	4	2
4	0	4	3	2	1

$$\varphi(5) = 4$$

Modular universe: Division

$$\mathbb{Z}_5^*$$

•	1	2	3	4
1	1	2	3	4
2	2	4	1	3
3	3	1	4	2
4	4	3	2	1

$$\varphi(5) = 4$$

Modular universe: Division

$$\mathbb{Z}_5^*$$

$\bullet$	**1**	**2**	**3**	**4**
1	1	2	3	4
2	2	4	1	3
3	3	1	4	2
4	4	3	2	1

For P prime, $\varphi(P) = P - 1.$

Modular universe: Division

$\mathbb{Z}_8^*$

·	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7
2	0	2	4	6	0	2	4	6
3	0	3	6	1	4	7	2	5
4	0	4	0	4	0	4	0	4
5	0	5	2	7	4	1	6	3
6	0	6	4	2	0	6	4	2
7	0	7	6	5	4	3	2	1

$$\varphi(8) = 4$$

Modular universe: Division

$$\mathbb{Z}_8^*$$

$\bullet$	**1**	**3**	**5**	**7**
1	1	3	5	7
3	3	1	7	5
5	5	7	1	3
7	7	5	3	1

$$\varphi(8) = 4$$

Modular universe: Division

$$\mathbb{Z}_{15}^*$$

•	1	2	4	7	8	11	13	14
1	1	2	4	7	8	11	13	14
2	2	4	8	14	1	7	11	13
4	4	8	1	13	2	14	7	11
7	7	14	13	4	11	2	1	8
8	8	1	2	11	4	13	14	7
11	11	7	14	2	13	1	8	4
13	13	11	7	1	14	8	4	2
14	14	13	11	8	7	4	2	1

$$\varphi(15) = 8$$

Modular universe: Division

$$\mathbb{Z}_{15}^*$$

•	1	2	4	7	8	11	13	14
1	1	2	4	7	8	11	13	14
2	2	4	8	14	1	7	11	13
4	4	8	1	13	2	14	7	11
7	7	14	13	4	11	2	1	8
8	8	1	2	11	4	13	14	7
11	11	7	14	2	13	1	8	4
13	13	11	7	1	14	8	4	2
14	14	13	11	8	7	4	2	1

Exercise: For P, Q distinct primes, $\varphi(PQ) = (P-1)(Q-1)$.

Modular universe: Division

$\mathbb{Z}_8^*$

$\bullet$	1	3	5	7
1	1	3	5	7
3	3	1	7	5
5	5	7	1	3
7	7	5	3	1

$\varphi(8) = 4$

For every $A \in \mathbb{Z}_N^*$, A^{-1} exists.

This implies:

A row contains distinct elements.

i.e. every row is a permutation of $\mathbb{Z}_N^*$.

$$A \cdot B = A \cdot B' \implies B = B'$$

Summary

+	0	1	2	3
0	0	1	2	3
1	1	2	3	0
2	2	3	0	1
3	3	0	1	2

$\mathbb{Z}_4$

behaves nicely with respect to *addition*

•	1	3	5	7
1	1	3	5	7
3	3	1	7	5
5	5	7	1	3
7	7	5	3	1

$\mathbb{Z}_8^*$

behaves nicely with respect to *multiplication*

Modular universe: Exponentiation

Given $A, B, N, \quad \text{len}(A), \text{len}(B), \text{len}(N) \leq n$

Compute $A^B \bmod N$.

We saw for integers, no hope for a poly-time algorithm.

In the modular universe, length of output not an issue.

In fact, we can compute this efficiently!

Modular universe: Exponentiation

Example

Compute $2337^{32} \bmod 100.$

Naïve strategy:

2337 x 2337 = 5461569

2337 x 5461569 = 12763686753

2337 x 12763686753 = …

$\vdots$ *(30 more multiplications later)*

626727565152155511653188886668668588313475824236665607396755008905770146236635537228216696030970612828922881

Modular universe: Exponentiation

Example

Compute $2337^{32} \bmod 100$.

2 improvements:

- Reduce mod 100 after every step.

- Don't multiply 32 times. Square 5 times.

$$2337 \longrightarrow 2337^2 \longrightarrow 2337^4 \longrightarrow 2337^8 \longrightarrow 2337^{16} \longrightarrow 2337^{32}$$

(what if the exponent was 53?)

Modular universe: Exponentiation

Example

Compute $2337^{53} \bmod 100$.

(what if the exponent was 53?)

Multiply powers 32, 16, 4, 1. (53 = 32 + 16 + 4 + 1)

$$2337^{53} = 2337^{32} \cdot 2337^{16} \cdot 2337^{4} \cdot 2337^{1}$$

53 in binary = 110101

Modular universe: Exponentiation

Given $A, B, N, \quad \mathrm{len}(A), \mathrm{len}(B), \mathrm{len}(N) \leq n$

Compute $A^B \bmod N$.

<u>Algorithm</u>:

- Repeatedly square A, always mod N.
  Do this n times.

- Multiply together the powers of A
  corresponding to the binary digits of B
  (again, always mod N).

Running time: a bit more than $O(n^2 \log n)$.

Modular universe: Exponentiation

Given $A, B, N, \quad \mathrm{len}(A), \mathrm{len}(B), \mathrm{len}(N) \leq n$

Compute $A^B \bmod N$.

Anything interesting we can do in the special case of

$\gcd(A, N) = 1?$ i.e. $A \in \mathbb{Z}_N^*$

Modular universe: Exponentiation

Euler's Theorem:

For any $A \in \mathbb{Z}_N^*$, $\quad A^{\varphi(N)} = 1$.

Equivalently, for A and N with $\gcd(A, N) = 1$,

$$A^{\varphi(N)} \equiv 1 \bmod N$$

When *N* is a prime, this is known as:

Fermat's Little Theorem:

Let P be a prime. For any $A \in \mathbb{Z}_P^*$, $\quad A^{P-1} = 1.$

Equivalently, for any A not divisible by P,

$$A^{P-1} \equiv 1 \bmod P$$

Modular universe: Exponentiation

Example

$\mathbb{Z}_8^*$

•	1	3	5	7
1	1	3	5	7
3	3	1	7	5
5	5	7	1	3
7	7	5	3	1

$\varphi(8) = 4$

1	1^2	1^3	1^4	1^5	1^6	1^7	1^8
1	1	1	1	1	1	1	1
3	3^2	3^3	3^4	3^5	3^6	3^7	3^8
3	1	3	1	3	1	3	1
5	5^2	5^3	5^4	5^5	5^6	5^7	5^8
5	1	5	1	5	1	5	1
7	7^2	7^3	7^4	7^5	7^6	7^7	7^8
7	1	7	1	7	1	7	1

Modular universe: Exponentiation

Example

$\mathbb{Z}_5^*$

$\cdot$	1	2	3	4
1	1	2	3	4
2	2	4	1	3
3	3	1	4	2
4	4	3	2	1

$\varphi(8) = 4$

1	1^2	1^3	$\boxed{1^4}$	1^5	1^6	1^7	$\boxed{1^8}$
1	1	1	1	1	1	1	1
2	2^2	2^3	$\boxed{2^4}$	2^5	2^6	2^7	$\boxed{2^8}$
2	4	3	1	2	4	3	1
3	3^2	3^3	$\boxed{3^4}$	3^5	3^6	3^7	$\boxed{3^8}$
3	4	2	1	3	4	2	1
4	4^2	4^3	$\boxed{4^4}$	4^5	4^6	4^7	$\boxed{4^8}$
4	1	4	1	4	1	4	1

2 and 3 are called generators.

Poll

What is $213^{248} \bmod 7$?

- 0
- 1
- 2
- 3
- 4
- 5
- 6
- Beats me.

Poll Answer

Euler's Theorem:

For any $A \in \mathbb{Z}_N^*$, $\quad A^{\varphi(N)} = 1\,.$

$$\begin{array}{cccc|cccc|cc}
A^0 & A^1 & A^2 & \cdots & A^{\varphi(N)} & A^{\varphi(N)+1} & & \cdots & A^{2\varphi(N)} & A^{2\varphi(N)+1} \\
\| & & & & \| & \| & & & \| & \| \\
1 & & & & A^0 & A^1 & & \cdots & A^0 & A^1
\end{array}$$

In other words, the exponent can be reduced $\bmod \varphi(N)$.

$$213^{248} \equiv_7 3^{248}$$

$$3^{248} \equiv_7 3^2 \qquad = 2$$

Poll Answer

When exponentiating elements $\boxed{A \in \mathbb{Z}_N^*}$

can think of the exponent living in the universe $\mathbb{Z}_{\varphi(N)}$.

Modular universe: Taking logarithms

Given A, B, P such that:

- P is prime
- $A \in \mathbb{Z}_P^*$
- $B \in \mathbb{Z}_P^*$ is a generator.

Find X such that $B^X \equiv_P A$.

It is like we want to compute $\log_B A$.

Poll

Find X such that $B^X \equiv_P A$.

What do you think of this algorithm:

```
DiscreteLog(A, B, P):
    for X = 0, 1, 2, …, P-2
        compute B^X    (use fast modular exponentiation)
        check whether P divides B^X - A
```

- simple and efficient. love it.
- simple but not efficient.
- loop should go up to X = P-1
- I don't understand why we are checking if P divides B^X - A.
- I don't understand what is going on right now.

Modular universe: Taking logarithms

Given A, B, P such that:

- P is prime
- $A \in \mathbb{Z}_P^*$
- $B \in \mathbb{Z}_P^*$ is a generator.

Find X such that $B^X \equiv_P A$.

We don't know how to compute this efficiently!

Modular universe: Taking roots

As an example, let's consider taking cube roots

Given A, N such that $A \in \mathbb{Z}_N^*$.

Find B such that $B^3 \equiv_N A$.

We don't know how to compute this efficiently!

Main goal of this lecture

Modular Universe

- How to view the elements of the universe?

- How to do basic operations:

> addition
> subtraction
> multiplication
> division
> exponentiation
> taking roots
> logarithm

theory
+
algorithms
(efficient (?))

Back to division in the modular universe

(i.e. things you will prove in the homework)

😀

2 Questions remain

How do you prove:

$A^{-1} \in \mathbb{Z}_N$ exists <u>if and only if</u> $\gcd(A, N) = 1.$

How do you compute: $A \cdot B^{-1} \bmod N$

i.e., how do you compute B^{-1}?

How to compute the multiplicative inverse

How do you compute: $A \cdot B^{-1} \bmod N$

i.e., how do you compute B^{-1}?

To determine if B has an inverse, we need to compute

$$\gcd(B, N)$$

Euclid's Algorithm finds gcd in polynomial time.

Arguably the first ever algorithm. ~ 300 BC

How to compute the multiplicative inverse

Euclid's Algorithm

```
gcd(A, B):
    if B == 0, return A
    return gcd(B,  A mod B)
```

Homework

Why does it work?

Why is it polynomial time?

Major open problem in Computer Science

Is gcd computation efficiently parallelizable?

i.e., is there a circuit family of

- poly(n) size
- polylog(n) depth

that computes gcd?

How to compute the multiplicative inverse

Ok, Euclid's Algorithm tells us whether an element has an inverse. How do you find it if it exists?

Definition: We say that C is a miix of A and B if

$$C = k \cdot A + \ell \cdot B$$

not a real term 😋

for some $k, \ell \in \mathbb{Z}$.

Examples:

2 is a miix of 14 and 10: 2 = (-2)·14 + 3·10

Any multiple of 2 is a miix of 14 and 10.

7 is not a miix of 55 and 40: any miix would be divisible by 5.

How to compute the multiplicative inverse

Fact: C is a miix of A and B <u>if and only if</u> C is a multiple of $\gcd(A, B)$.

So $$\gcd(A, B) = k \cdot A + \ell \cdot B$$

The coefficients k and ℓ can be found by slightly modifying Euclid's Algorithm.

<u>Finding B^{-1}:</u>

If $\gcd(B, N) = 1$, we can find $k, \ell \in \mathbb{Z}$ such that

$$1 = k \cdot B + \ell \cdot N$$

$$||$$

Therefore found B^{-1}

2 Questions remain

How do you prove:

$A^{-1} \in \mathbb{Z}_N$ exists <u>if and only if</u> $\gcd(A, N) = 1.$

How do you compute: $A \cdot B^{-1} \bmod N$

i.e., how do you compute B^{-1}?

When does the inverse exist

How do you prove:

$A^{-1} \in \mathbb{Z}_N$ **exists <u>if and only if</u>** $\gcd(A, N) = 1.$

N **divides** $k \cdot A - 1$

<u>Proof</u>:

$$A^{-1} \text{ exists}$$

$$\iff \exists k \quad \text{such that} \quad k \cdot A \equiv_N 1$$

$$\iff \exists k, q \quad \text{such that} \quad k \cdot A - 1 = q \cdot N$$

$$\iff \exists k, q \quad \text{such that} \quad 1 = k \cdot A + (-q) \cdot N$$

$$\iff 1 \text{ is a miix of } A \text{ and } N$$

$$\iff \gcd(A, N) = 1$$

Main goal of this lecture

Modular Universe

- How to view the elements of the universe?

- How to do basic operations:

> addition
>
> subtraction
>
> multiplication
>
> division
>
> exponentiation
>
> taking roots
>
> logarithm

theory

+

algorithms

(efficient (?))

Next Time

Cryptography