School of Electronics Engineering and Computer Science
Peking University

Mathematics in Olympiad in Informatics

Part 1

Frederica Haoyue Shi
hyshi@pku.edu.cn

January 22, 2017

Content

Introduction

English is **very helpful** to our future study and work. Please try to use English as much as possible, from now on.

Therefore, the slides are written in English.

If there's no specific instruction, the complexity is always time complexity.

For the convenience of view, I recommend you to open it with Adobe Reader.

Fundamental theorem of arithmetic

Every integer greater than 1 either is prime itself or is the product of prime numbers, and that this product is unique, up to the order of the factors. i.e.

$$\forall n > 1, n \in \mathbb{N}, n = p_1^{a_1} p_2^{a_2} ... p_n^{a_n}$$

where $p_k (1 \leq k \leq n), p_1 < p_2 < ... < p_n$ is prime.

Fundamental theorem of arithmetic

Every integer greater than 1 either is prime itself or is the product of prime numbers, and that this product is unique, up to the order of the factors. i.e.

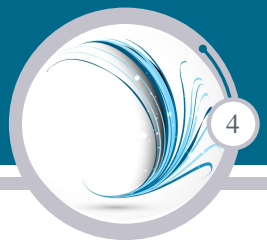
$$\forall n > 1, n \in \mathbb{N}, n = p_1^{a_1} p_2^{a_2} ... p_n^{a_n}$$

where $p_k (1 \leq k \leq n), p_1 < p_2 < ... < p_n$ is prime.

Prime Number Theorem

$$\pi(N) \sim \frac{N}{\log(N)}$$

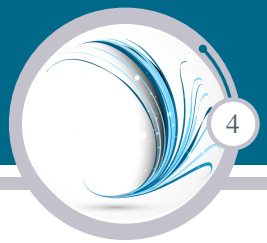
where $\pi(N)$ is the prime counting function, and $\log(N)$ is the natural logarithm of N. The equation means that for large enough N, the probability that a random natural number not greater than N is prime is very close to $\frac{1}{\log(N)}$.



How to check whether a number N is prime?

Pseudo-code for prime number checking.

```
1: for i = 2 to √N do
2:     if i is divisor of N then
3:         return False
4:     end if
5: end for
```

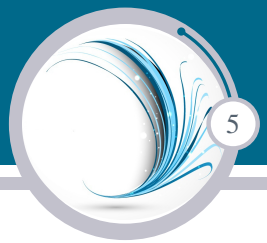


How to check whether a number N is prime?

Pseudo-code for prime number checking.

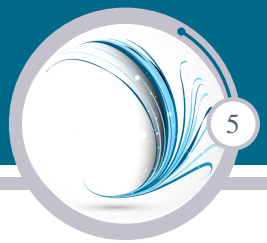
```
1: for i = 2 to √N do
2:     if i is divisor of N then
3:         return False
4:     end if
5: end for
```

The time complexity of such algorithm is $O(\sqrt{N})$.
The space complexity of such algorithm is $O(1)$.



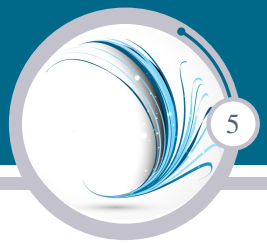
How to generate a prime number list from 1 to N?

1. Check whether a number is prime one by one.



How to generate a prime number list from 1 to N?

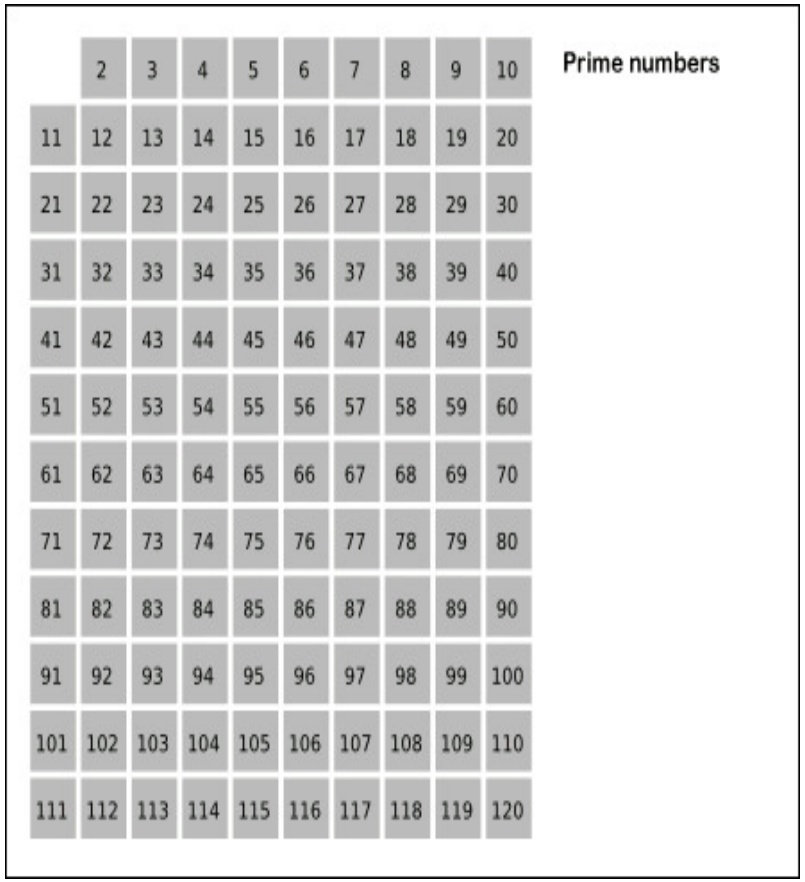
1. Check whether a number is prime one by one. $O(N\sqrt{N})$



How to generate a prime number list from 1 to N?

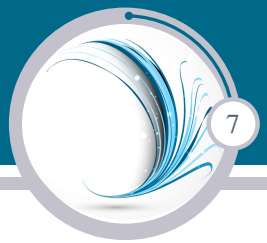
1. Check whether a number is prime one by one. $O(N\sqrt{N})$
2. Sieve of Eratosthenes

Sieve of Eratosthenes



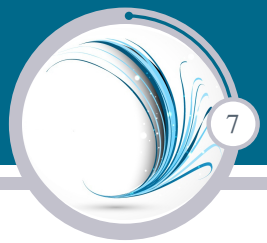
How to generate a prime number list from 1 to N?

1. Check whether a number is prime one by one. $O(N\sqrt{N})$
2. Sieve of Eratosthenes



How to generate a prime number list from 1 to N?

1. Check whether a number is prime one by one. $O(N\sqrt{N})$
2. Sieve of Eratosthenes $O(N\log(N))$

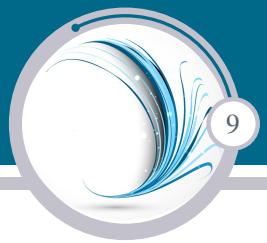


How to generate a prime number list from 1 to N?

1. Check whether a number is prime one by one. $O(N\sqrt{N})$
2. Sieve of Eratosthenes $O(N\log(N))$
3. Linear Sieve

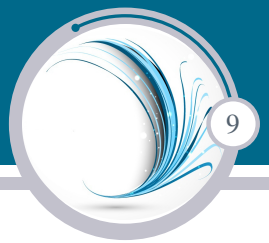
Pseudo-code for linear sieve.

```
1:  for i = 2 to N do
2:      set IsPrime[i] = True
3:  end for
4:  for i = 2 to N do
5:      if IsPrime[i] then
6:          push i into PrimeNumberList
7:      end if
8:      j = 0
9:      while j < CurrentPrimeNumber and i·PrimeNumberList[j] < N do
10:         IsPrime[i·PrimeNumberList[j]] = False
11:         if PrimeNumberList[j] is divisor of i then
12:             break
13:         end if
14:         increase j
15:     end while
16: end for
```



Linear Sieve

The time complexity of such algorithm is $O(N)$.
The space complexity of such algorithm is $O(N)$.

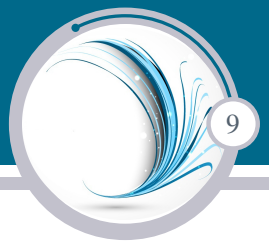


Linear Sieve

The time complexity of such algorithm is $O(N)$.
The space complexity of such algorithm is $O(N)$.

Practice: POJ 2689

Given $L, U (1 \leq L < U < 2^{31}, U - L \leq 1,000,000)$, you are to find the two adjacent primes C_1 and $C_2 (L \leq C_1 < C_2 \leq U)$ that are closest (i.e. $C_2 - C_1$ is the minimum), and the two adjacent primes D_1 and $D_2 (L \leq D_1 < D_2 \leq U)$ that are the most distant (i.e. $D_2 - D_1$ is the maximum).



Linear Sieve

The time complexity of such algorithm is $O(N)$.
The space complexity of such algorithm is $O(N)$.

Practice: POJ 2689

Given $L, U (1 \leq L < U < 2^{31}, U - L \leq 1,000,000)$, you are to find the two adjacent primes C_1 and $C_2 (L \leq C_1 < C_2 \leq U)$ that are closest (i.e. $C_2 - C_1$ is the minimum), and the two adjacent primes D_1 and $D_2 (L \leq D_1 < D_2 \leq U)$ that are the most distant (i.e. $D_2 - D_1$ is the maximum).

Hint: You are definitely required to generate a prime number list in the range of $[L, U]$. But how?

Optional: Mathematical Induction

Mathematical induction is a mathematical proof technique used to prove a given statement about any well-ordered set. Most commonly, it is used to establish statements for the set of all natural numbers.

Optional: Mathematical Induction

Mathematical induction is a mathematical proof technique used to prove a given statement about any well-ordered set. Most commonly, it is used to establish statements for the set of all natural numbers.

It contains two steps:

1. **base case**, to prove the given statement for the least element in the well-ordered set.

Optional: Mathematical Induction

Mathematical induction is a mathematical proof technique used to prove a given statement about any well-ordered set. Most commonly, it is used to establish statements for the set of all natural numbers.

It contains two steps:

1. **base case**, to prove the given statement for the least element in the well-ordered set.
2. **inductive step**, to prove that, if the statement is assumed to be true for any element, then it must be true for the next element as well.

Optional: Mathematical Induction

An Example

Prove $\sum_{i=0}^{n} = \frac{n(n+1)}{2}, \forall n \in \mathbb{N}$

Optional: Mathematical Induction

An Example

Prove $\sum_{i=0}^{n} = \frac{n(n+1)}{2}, \forall n \in \mathbb{N}$

Proof For the case $n = 0$, we have $\sum_{i=0}^{n} = \frac{n(n+1)}{2} = 0$.

Optional: Mathematical Induction

An Example

Prove $\sum_{i=0}^{n} = \frac{n(n+1)}{2}, \forall n \in \mathbb{N}$

Proof For the case $n = 0$, we have $\sum_{i=0}^{n} = \frac{n(n+1)}{2} = 0$.

Assume that $\sum_{i=0}^{n} = \frac{n(n+1)}{2}$ when $n = k$,

then $\sum_{i=0}^{k+1} = \frac{k(k+1)}{2} + k + 1 = \frac{k(k+1)+2(k+1)}{2} = \frac{(k+1)(k+2)}{2}$.

Hence, the statement is true when $n = k + 1$.

Optional: Mathematical Induction

An Example

Prove $\sum_{i=0}^{n} = \frac{n(n+1)}{2}, \forall n \in \mathbb{N}$

Proof For the case $n = 0$, we have $\sum_{i=0}^{n} = \frac{n(n+1)}{2} = 0$.

Assume that $\sum_{i=0}^{n} = \frac{n(n+1)}{2}$ when $n = k$,

then $\sum_{i=0}^{k+1} = \frac{k(k+1)}{2} + k + 1 = \frac{k(k+1)+2(k+1)}{2} = \frac{(k+1)(k+2)}{2}$.

Hence, the statement is true when $n = k + 1$.

Therefore, $\sum_{i=0}^{n} = \frac{n(n+1)}{2}, \forall n \in \mathbb{N}$. #

Euclidean Algorithm for Greatest Common Divisor(GCD)

Pseudo-code for Euclidean Algorithm.

```
1: function GCD(integer a,b)
2: if b == 0 then
3:     return a
4: else
5:     return GCD(b, a mod b)
6: end if
```

Euclidean Algorithm for Greatest Common Divisor(GCD)

Pseudo-code for Euclidean Algorithm.

```
1: function GCD(integer a,b)
2: if b == 0 then
3:     return a
4: else
5:     return GCD(b, a mod b)
6: end if
```

Most of you know how, but why?

Euclidean Algorithm for Greatest Common Divisor(GCD)

Proof

For any two integers a, b, a can be written as

$$a = bq + r$$

where q, r are non-negative integers, $0 \leq r < b$.

Euclidean Algorithm for Greatest Common Divisor(GCD)

Proof

For any two integers a, b, a can be written as

$$a = bq + r$$

where q, r are non-negative integers, $0 \leq r < b$.

For all intergers d, if $d|a$ and $d|b$, then

$$d|bq \Rightarrow d|(a - bq) \Rightarrow d|r$$

Euclidean Algorithm for Greatest Common Divisor(GCD)

Proof

For any two integers a, b, a can be written as

$$a = bq + r$$

where q, r are non-negative integers, $0 \leq r < b$.
For all intergers d, if $d|a$ and $d|b$, then

$$d|bq \Rightarrow d|(a - bq) \Rightarrow d|r$$

Namely, if d is a common divisor of a and b, then d is a common divisor of b and r, and vice versa.

Euclidean Algorithm for Greatest Common Divisor(GCD)

Proof

For any two integers a, b, a can be written as

$$a = bq + r$$

where q, r are non-negative integers, $0 \leq r < b$.
For all intergers d, if $d|a$ and $d|b$, then

$$d|bq \Rightarrow d|(a - bq) \Rightarrow d|r$$

Namely, if d is a common divisor of a and b, then d is a common divisor of b and r, and vice versa.
Hence, GCD of a and b (denoted as (a, b)) equals (b, r). #

Bézout's Identity(Lemma)

Let a and b be nonzero integers and let d be their greatest common divisor. Then there exist integers x and y such that

$$ax + by = d$$

In addition,

- the greatest common divisor d is the smallest positive integer that can be written as $ax + by$
- every integer of the form $ax + by$ is a multiple of the greatest common divisor d.

It can be proved similarly to the proof of Euclidean Algorithm, and the so called Extended Euclidean Algorithm comes out.

Extended Euclidean Algorithm

Purpose: Given 3 intergers a, b, c, to find out all solutions for

$$ax + by = c$$

.

Pseudo-code for Extended-Euclidean Algorithm

```
1: function EX-EUC(integer a,b)
2: if b == 0 then
3:     set x = 1, y = 0
4:     return a
5: else
6:     set d = EX-EUC(b, a mod b)
7:     set t = x, x = y, y = t - a / b * y
8:     return d
9: end if
```

Extended Euclidean Algorithm

Purpose: Given 3 intergers a, b, c, to find out all solutions for

$$ax + by = c$$

.

Remember to check the case of NO SOLUTION : $d \nmid c$.

Practice: Compute $\sum_{i=1}^{n} K \bmod i, 1 \leq K \leq 10^9$

Practice: Compute $\sum_{i=1}^{n} K \bmod i, 1 \leq K \leq 10^9$

Hint:

$$K \bmod i = K - \lfloor \frac{K}{i} \rfloor i$$

Practice: Compute $\sum_{i=1}^{n} K \bmod i, 1 \leq K \leq 10^9$

Hint:

$$K \bmod i = K - \lfloor \frac{K}{i} \rfloor i$$

For some successive i, $\lfloor \frac{K}{i} \rfloor$ remains constant.

Practice: Compute $\sum_{i=1}^{n} K \bmod i, 1 \leq K \leq 10^9$

Hint:

$$K \bmod i = K - \lfloor \frac{K}{i} \rfloor i$$

For some successive i, $\lfloor \frac{K}{i} \rfloor$ remains constant.
How many such intervals are there, in the range of $[1, K]$?

Practice: Compute $\sum_{i=1}^{n} K \bmod i, 1 \leq K \leq 10^9$

Hint:

$$K \bmod i = K - \lfloor \frac{K}{i} \rfloor i$$

For some successive i, $\lfloor \frac{K}{i} \rfloor$ remains constant.

How many such intervals are there, in the range of $[1, K]$? $O(\sqrt{K})$

Please finish the proof by yourselves and write the program after class.

Euler's totient function

$$\phi(n) = n \prod_{p|n} (1 - \frac{1}{p})$$

where p is a prime number.

Euler's totient function

$$\phi(n) = n \prod_{p|n} (1 - \frac{1}{p})$$

where p is a prime number.
Proof: Inclusion-Exclusion Principle

Euler's totient function

$$\phi(n) = n \prod_{p|n} (1 - \frac{1}{p})$$

where p is a prime number.
Proof: Inclusion-Exclusion Principle

Euler's Theorem

If n and a are coprime positive integers, then

$$a^{\phi(n)} \equiv 1 (\text{mod } n)$$

Proof of Euler's Theorem

Proof Let R be a reduced residue system (mod n) and let a be any integer coprime to n.

Proof of Euler's Theorem

Proof Let R be a reduced residue system $(\mathrm{mod}\ n)$ and let a be any integer coprime to n.

The proof hinges on the **fundamental fact** that multiplication by a permutes the sequence $\{x_i\}$: in other words if $ax_j \equiv ax_k (\mathrm{mod}\ n)$ then $j = k$.

Proof of Euler's Theorem

Proof Let R be a reduced residue system $(\mathrm{mod}\ n)$ and let a be any integer coprime to n.

The proof hinges on the **fundamental fact** that multiplication by a permutes the sequence $\{x_i\}$: in other words if $ax_j \equiv ax_k (\mathrm{mod}\ n)$ then $j = k$.

That is, the sets R and $aR = \{ax_1, ax_2, \ldots, ax_{\phi(n)}\}$.

Proof of Euler's Theorem

Proof Let R be a reduced residue system $(\mathrm{mod}\ n)$ and let a be any integer coprime to n.

The proof hinges on the **fundamental fact** that multiplication by a permutes the sequence $\{x_i\}$: in other words if $ax_j \equiv ax_k (\mathrm{mod}\ n)$ then $j = k$.

That is, the sets R and $aR = \{ax_1, ax_2, \ldots, ax_{\phi(n)}\}$.

Therefore,

$$\prod_{i=1}^{\phi(n)} x_i \equiv \prod_{i=1}^{\phi(n)} ax_i \equiv a^{\phi(n)} \prod_{i=1}^{\phi(n)} x_i$$

Proof of Euler's Theorem

Proof Let R be a reduced residue system $(\text{mod } n)$ and let a be any integer coprime to n.

The proof hinges on the **fundamental fact** that multiplication by a permutes the sequence $\{x_i\}$: in other words if $ax_j \equiv ax_k (\text{mod } n)$ then $j = k$.

That is, the sets R and $aR = \{ax_1, ax_2, \ldots, ax_{\phi(n)}\}$.

Therefore,

$$\prod_{i=1}^{\phi(n)} x_i \equiv \prod_{i=1}^{\phi(n)} ax_i \equiv a^{\phi(n)} \prod_{i=1}^{\phi(n)} x_i$$

$$\Rightarrow a^{\phi(n)} \equiv 1 (\text{mod } n)$$

Fermat's Little Theorem

If p is a prime number, then for any integer a, the number $a^p - a$ is an integer multiple of p. In the notation of modular arithmetic, this is expressed as

$$a^p \equiv a \pmod{p}$$

Fermat's Little Theorem

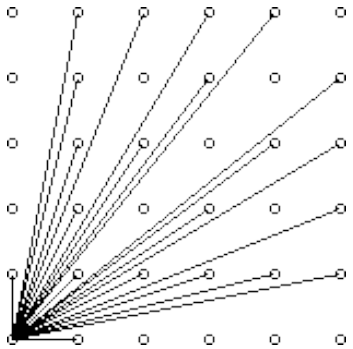
If p is a prime number, then for any integer a, the number $a^p - a$ is an integer multiple of p. In the notation of modular arithmetic, this is expressed as

$$a^p \equiv a \pmod{p}$$

Proof: Just a special case for Euler's Theorem.

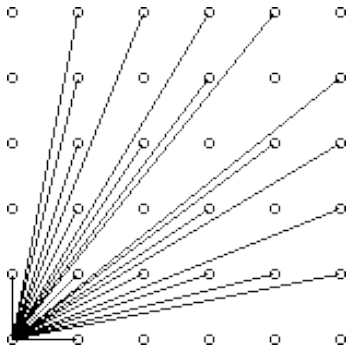
Practice: POJ3090

Write a program which, given a value for the size, N, computes the number of visible points (x, y) with $0 \leq x, y \leq N$. $(1 \leq N \leq 1000)$



Practice: POJ3090

Write a program which, given a value for the size, N, computes the number of visible points (x, y) with $0 \leq x, y \leq N$. $(1 \leq N \leq 1000)$



Hint: $GCD(x, y) = 1$

Modular multiplicative inverse

This concept is analogous to the concept of the multiplicative inverse over the set of rational numbers.

The modulo multiplicative inverse for the congruence class $\overline{a}$ is the congruence class $\overline{x}$ such that $\overline{a} \cdot \overline{x} \equiv 1 \pmod{m}$

Modular multiplicative inverse

This concept is analogous to the concept of the multiplicative inverse over the set of rational numbers.

The modulo multiplicative inverse for the congruence class $\overline{a}$ is the congruence class $\overline{x}$ such that $\overline{a} \cdot \overline{x} \equiv 1 \pmod{m}$

When a congruence class $\overline{a}$ has modulo multiplicative inverse?

Modular multiplicative inverse

This concept is analogous to the concept of the multiplicative inverse over the set of rational numbers.

The modulo multiplicative inverse for the congruence class $\overline{a}$ is the congruence class $\overline{x}$ such that $\overline{a} \cdot \overline{x} \equiv 1 \pmod{m}$

When a congruence class $\overline{a}$ has modulo multiplicative inverse?

Answer: $(a, m) = 1$

Modular multiplicative inverse

This concept is analogous to the concept of the multiplicative inverse over the set of rational numbers.

The modulo multiplicative inverse for the congruence class $\overline{a}$ is the congruence class $\overline{x}$ such that $\overline{a} \cdot \overline{x} \equiv 1 \pmod{m}$

When a congruence class $\overline{a}$ has modulo multiplicative inverse?

Answer: $(a, m) = 1$

Usage: Compute $\frac{a}{b} \bmod m$, where $(b, m) = 1$

Chinese remainder theorem

Assert $m_1, m_2, ..., m_k$ are pairwise coprime integers, $m = m_1 m_2 ... m_k, M_i = \frac{m}{m_i}, M_i^{'} M_i \equiv 1 (\mathrm{mod} m_i), 1 \leq i \leq k$, then there exists an integer

$$x = \sum_{i=1}^{k} a_i M_i M_i^{'}$$

such that

$$x \equiv a_1 (\mathrm{mod}\ n_1)$$

$$x \equiv a_2 (\mathrm{mod}\ n_2)$$

$$...$$

$$x \equiv a_k (\mathrm{mod}\ n_k)$$

and any two x are congruent modulo M.

Task: Compute a^b, a and b are non-negative integers.

Pseudo-code for brute-force computing of a^b

```
1: set result = 0
2: for i = 1 to b do
3:     result = result ·a
4: end for
```

Task: Compute a^b, a and b are non-negative integers.

Pseudo-code for brute-force computing of a^b

```
1: set result = 0
2: for i = 1 to b do
3:     result = result ·a
4: end for
```

The time complexity for the algorithm is $O(N)$.
The space complexity for the algorithm is $O(1)$.

Task: Compute a^b, a and b are non-negative integers.

Pseudo-code for brute-force computing of a^b

```
1: set result = 0
2: for i = 1 to b do
3:     result = result ·a
4: end for
```

The time complexity for the algorithm is $O(N)$.
The space complexity for the algorithm is $O(1)$. Is there any faster algorithm? Yes.

Task: Compute a^b, a and b are non-negative integers.

Pseudo-code for faster computing of a^b

```
1:  function power(a, b)
2:  if b == 0 then
3:      return 1
4:  else
5:      temp = power(a, b/2)
6:      result = temp^2
7:      if b mod 2 == 1 then
8:          result = result · a
9:      end if
10:     return result
11: end if
```

Task: Compute a^b, a and b are non-negative integers.

Pseudo-code for faster computing of a^b

```
1:  function power(a, b)
2:  if b == 0 then
3:      return 1
4:  else
5:      temp = power(a, b/2)
6:      result = temp^2
7:      if b mod 2 == 1 then
8:          result = result · a
9:      end if
10:     return result
11: end if
```

Time Complexity: $O(\log N)$ Space Complexity: $O(\log N)$

Matrix Multiplication

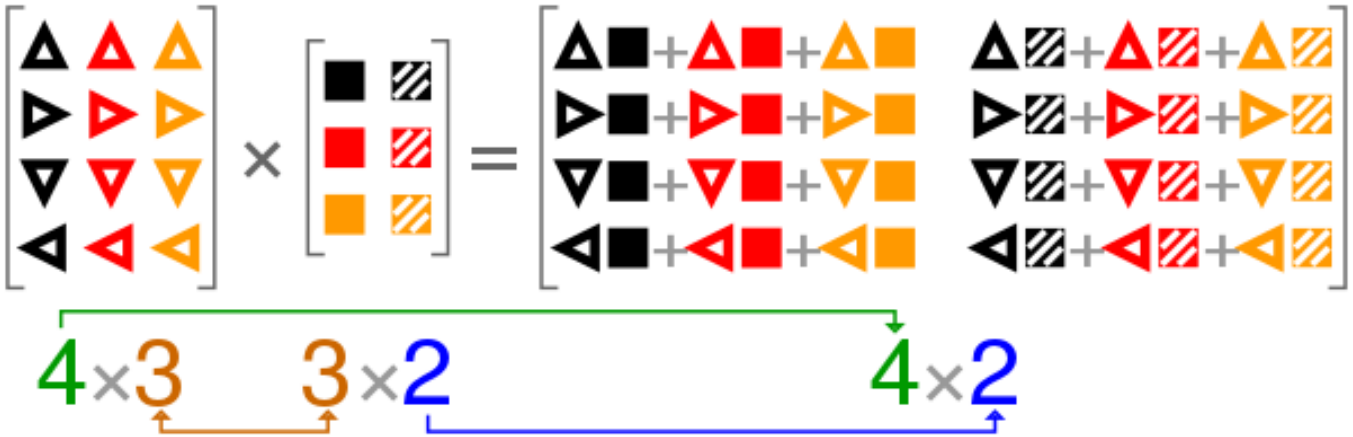
$$A \times B = C$$

$$C_{ij} = \sum_{k} A_{ik} B_{kj}$$

Matrix Multiplication

$$A \times B = C$$

$$C_{ij} = \sum_k A_{ik} B_{kj}$$



Properties of Matrix Multiplication

Properties of Matrix Multiplication

Associative Law?

Properties of Matrix Multiplication

Associative Law? Yes.

Properties of Matrix Multiplication

Associative Law? Yes.
Commutative Law?

Properties of Matrix Multiplication

Associative Law? Yes.
Commutative Law? **No!**

We could apply fast exponentiation method to matrix multiplication to compute A^n for a matrix A, but the code should be modify a little.

Task: Compute A^b, where A is a matrix and b is a non-negative integer.

Pseudo-code for faster computing of a^b

```
1:  function power(A, b)
2:  if b == 0 then
3:      return I
4:  else
5:      temp = power(A, b/2)
6:      result = temp^2
7:      if b mod 2 == 1 then
8:          result = result × A
9:      end if
10:     return result
11: end if
```

Usage: Compute the n^{th} item for Fibonacci Sequence.

Fibonacci Sequence: $f_0 = 1, f_1 = 1, f_n = f_{n-1} + f_{n-2} (k \geq 2)$

- Enumerate every i and compute f_i one by one.

Usage: Compute the n^{th} item for Fibonacci Sequence.

Fibonacci Sequence: $f_0 = 1, f_1 = 1, f_n = f_{n-1} + f_{n-2} (k \geq 2)$

- Enumerate every i and compute f_i one by one. $O(n)$

Usage: Compute the n^{th} item for Fibonacci Sequence.

Fibonacci Sequence: $f_0 = 1, f_1 = 1, f_n = f_{n-1} + f_{n-2} (k \geq 2)$

- Enumerate every i and compute f_i one by one. $O(n)$
- Consider matrix multiplication:

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

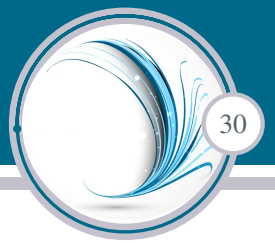
Usage: Compute the n^{th} item for Fibonacci Sequence.

Fibonacci Sequence: $f_0 = 1, f_1 = 1, f_n = f_{n-1} + f_{n-2} (k \geq 2)$

- Enumerate every i and compute f_i one by one. $O(n)$
- Consider matrix multiplication:

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

$O(\log n)$



Why it works?

When we compute

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

and

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

etc.

We actually did the recurrence procedure. Thus, according to the associative law of matrix multiplication, why not combine the exponentiation of the matrix in order to make the algorithm faster?


Part 1 ends here. Thank you for attention!
Powereed by LaTeX.