Programming Challenges (GB21802)

Week 8 - Mathematics

Claus Aranha
caranha@cs.tsukuba.ac.jp

University of Tsukuba, Department of Computer Sciences

2020/6/16

(last updated: June 15, 2020)

Version 2020.1

New Deadlines (More time to solve problems)

The deadlines for the final three classes, and the extra deadline for late exercises is the following:

- Lecture 8: Lecture: 6/16, Deadline: 6/25 (10 days)
- Lecture 9: Lecture: 6/23, Deadline: 7/02 (10 days)
- Lecture 10: Lecture: 6/30, Deadline: 7/09 (10 days)
- Late Submission Time: 7/10 to 7/21 (11 days)

If you have too many reports right now, please use the Late Submission Time!

Math Problems: Lecture Outline

Every computer program requires some amount of mathematics. So what does **"Math Problems"** mean in Programming Challenges?

Here we describe two kinds of problems as **"Math Problems"**:

The Challenge is The Implementation of Mathematical Concepts

- Problems with Big Numbers (above variable limits)
- Problems with Geometry (next lecture!)

The Challenge Requires Mathematical Planning Before Programming

In this case, it is sometimes possible to solve the entire problem in paper and quickly implement a solution to the problem.

- Number Theory (primality testing, factorization, rings)
- Combinatorics (sequences, counting, recurrences)

Implementation Tricks

- BigNums;
- Modulo Operations;

Dealing with Big Numbers

Some problems (specially math problems) require using very large numbers. For example:
$25! = 15511210043330985984000000 > 10^{26}$.

However:

- Maximum C++ unsigned int: $2^{32} < 10^{11}$
- Maximum C++ unsigned long long: $2^{64} < 10^{20}$

I usually recommend to use C++; but Java is better for BigNum progchal problems!

Bignum Example: 10925 – Krakovia

```
import java.util.Scanner;
import java.math.BigInteger;
class Main {
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    int caseNo = 1;
    while (true) {
      int N = sc.nextInt(), F = sc.nextInt();
      if (N == 0 && F == 0) break;
      BigInteger sum = BigInteger.ZERO;     // Bignum Constant
      for (int i = 0; i < N; i++) {
        BigInteger V = sc.nextBigInteger(); // Bignum I/O
        sum = sum.add(V); }
      System.out.println("Bill #" + (caseNo++)
        + " costs " + sum + ": each friend should pay "
        + sum.divide(BigInteger.valueOf(F)) + "\n" );}
  }
}
```

More functions from Java.math.BigInteger

Algebraic functions

BigInteger.add(), .subtract(), .multiply(), .divide(), .pow(), .mod(), .remainder()

Changing Number Base

```
BI = BigInteger(10); System.println(BI.toString(2))
// Result: 1010
```

Probabilistic Primality Test

```
isPrime = BI.isProbablePrime(int certainty)
// Chance of being correct is 1 - (1/2)^certainty
```

Other cool functions

BigInteger.gcd(BI) BigInteger.modPow(BI exponent, BI m)

Modulo Operation

We can use modulo arithmetic to operate on very large numbers without calculating the entire number.

Remember that:

1. $(a + b)\%s = ((a\%s) + (b\%s) + s)\%s$
2. $(a * b)\%s = ((a\%s) * (b\%s))\%s$
3. $(a^n)\%s = ((a^{n/2}\%s) * (a^{n/2}\%s) * (a^{n\%2}\%s))\%s$

Modulo Operation – UVA 10176, Ocean Deep!

Problem summary

Test if a binary number n (up to 100000 digits) is divisible by 131071

- The problem wants to know if $n\%13107 == 0$
- But n is too big!
- Use the recurrence in the previous slide to break down each digit to a reasonable value.

Number Theory

Number Theory studies the integer numbers and sets.

- Primality;
- Division and Remainders;
- Sequences of numbers;

Number Theory: Primality Testing

Prime Numbers: Only divisible by 1 and itself:

2,3,5,7,11,13...

How do you test if a number N is prime?

- Full search: For each $f \in 2..N-1$, test if $N\%f == 0$
  $O(N)$

- A little Pruning: For each $f \in 2..\mathrm{floor}(\sqrt{N})$, test if $N\%f == 0$
  $O(\sqrt{(}N))$
- Can you do it in $O(\sqrt{n}/\log(n))$?

Number Theory: Primality Testing

The Prime Number Theorem (simplified)

The probability of $i < N$ is prime is $1/\log(N)$

collorary[1] **1**: There are $N/\log(N)$ primes $< N$
collorary 2: We just need to test the **primes** between 1 and $\sqrt{N}$

But how do we find all primes between 1 and $\sqrt{N}$ fast?

[1] "Collorary" means "consequence"

Sieve of Eratosthenes

Idea

- Start with a set from 2 to $\sqrt{N}$.
- Test if each i in the set is prime.
- If i is prime, remove all multiples mi.

```
def sieve(k):                       ## Find all primes up to k
   primes = []
   sieve = [1]*(k+1)     ## all numbers start in the list
   sieve[0] = sieve[1] = 0              ## except 0 and 1
   for i in range(k+1):                           ## O(N)
      if (sieve[i] == 1):
         primes.append(i)               ## new prime found
         j = i*i   ## why can i start from i*i, not i*2?
         while (j < k+1):                    ## O(loglogN)
            sieve[j] = 0
            j += i                          ## next multiple
   return primes
```

Sieve of Eratosthenes

Amortized Complexity

- The complexity of the Sieve is $O(N \log \log N)$
- If we do the Sieve every time we test for primes, we are not saving much.
- But we can do the Sieve one time, and test many primes later!

When we do an expensive operation once, we call it **amortized complexity**

Finding Prime Factors

Any natural number N can be expressed as a unique set of prime numbers:

$$N = 1 p_1^{e_1} p_2^{e_2} \dots p_n^{e_n}$$

These are the Prime Factors of N. From this set, we can also obtain the set of Factors of N (all numbers i where $i|N$).

Factorization is a key issue in cryptography

Very Naive approach – Test all numbers!

For every $i \in 1..N/2$, test $i|N$ and isPrime(i).

Very Expensive!

Naive approach – Test all primes

Calculate a list of primes i up to N/2, test if $i|N$.

Wrong Answer, why?

Prime factorization: Divide and conquer approach

Recursive Idea

The prime factorization of N is equal to the union of p_i and the prime factorization of N/p_i, where p_i is the smallest prime factor of N.

The set of all factors is composed of all combinations of the set of prime factors (including repetitions).

```
def primefactors(n):
   primes = sieve(int(np.sqrt(n))+1)
   c = 0, i = n, factors = []
   while i > 1:
      if (i%primes[c] == 0):
         i = i/primes[c]
         factors.append(primes[c])
      else:
         c = c+1
   return factors
```

Working with Prime Factors: 10139 – Factovisors

Problem description

Calculate whether m divides $n!$ ($1 \leq m, n \leq 2^{31} - 1$)

Factorial of 22 is already bigint! But we can break down these numbers into their factors, which are all $\leq 2^{30}$.

- F_m: primefactors(m)
- $F_{n!}$: $\cup$(primefactors(1), primefactors(2)...,primefactors(n))

Having the factor sets, m divides $n!$ if $F_m \subset F_{n!}$.

Examples:

- $m = 48$ and $n = 6$
  $F_m = \{2, 2, 2, 2, 3\} F_{n!} = \{2, 3, 2, 2, 5, 2, 3\}$
- $m = 25$ and $n = 6$
  $F_m = \{5, 5\} F_{n!} = \{2, 3, 2, 2, 5, 2, 3\}$

Euclid Algorithm and Extended Euclid Algorithm

- Euclid Algorithm gives us the greatest common divisor D of a, b;
- Extended Euclid Algorithm also gives us x, y so that $ax + by = D$;
- Both are extremely simple to code:

```
int gcd(int a, int b) {return (a == 0?b:gcd(b%a,a));}

int x, y;
int egcd(int a, int b) {
   if (a==0)
      {x = 0; y = 1; return b;}        // stop condition
   int d = egcd(b%a, a);
   int tx = x;                          // gcd recurrence
   x = y - (b/a)*tx; y = tx; return d; }   // update x,y
```

Using EGCD: The Diophantine Equation

Problem Example (variations of this problem are common)

You have 839 yen. Xhoco candy costs 25 yen, Yanilla candy costs 18 yen. How many candies can we buy?

The equation $xA + yB = C$ is called the Linear Diophantine Equation. It has infinite solutions if GCD(A,B)|C, but none if it does not.

The first solution (x_0, y_0) can be derived from the extended GCD, and other solutons can be found from: expressed as:

- $x = x_0 + (b/d)n$
- $y = y_0 - (a/d)n$

Where d is GCD(A,B) and n is an integer.

Using EGCD: The Diophantine Equation

Problem Example (variations of this problem are common)

You have 839 yen. Xhoco candy costs 25 yen, Yanilla candy costs 18 yen. How many candies can we buy?

- EGCD gives us: $x = -5, y = 7, d = 1$ or $25(-5) + 18(7) = 1$
- Multiply both sides by 839: $25(-4195) + 18(5873) = 839$
- So: $x_n = -4195 + 18n$ and $y_n = 5873 - 25n$
- We have to find n so that both x_n, y_n are > 0.
- $-4195 + 18n \geq 0$ and $5873 - 25n \geq 0$
- $n \geq 4195/18$ and $5873/25 \geq n$
- $4195/18 \leq n \leq 5873/25$
- $233.05 \leq n \leq 234.92$

Combinatorics problems

Definition

Combinatorics is the branch of mathematics concerning the study of countable discrete structures.

Combinatory problems involve understanding a sequence, and figuring one of:

- Recurrence: A formula that calculates the n^{th} member of a sequence, based on the value of previous members;
- Closed form: A formula that calculates the n^{th} member of a sequence independently from other members;

It is not uncommon to use Dynamic Programming or Bignum to solve combinatoric related problems.

Example: Triangular Numbers

Definition

The triangular numbers is the sequence where the n^{th} value is composed of the sum of all integers from 1 to n

- S(1) = 1
- S(2) = 1+2 = 3
- S(3) = 1+2+3 = 6
- ...
- S(7) = 1+2+3+4+5+6+7 = 28

What are the recurrence and the closed form for this sequence?

Example: Triangular Numbers

- S(1) = 1, S(2) = 3, S(3) = 6

Recurrence

The recursive form of a sequence:

$$S(n) = S(n-1) + n; S(1) = 1$$

Closed Form

The non-recursive form of a sequence:

$$S(n) = \frac{n(n+1)}{2}$$

Problem: Calculate the first triangle number with more than 500 factors!

A more famous sequence: Fibonacci Numbers

Definition – very famous sequence

Each number is the sum of the two numbers before it.

F() = 0,1,1,2,3,5,8,13,21,34...

The recurrence is well known

$$F(0) = 0, F(1) = 1, F(n) = F(n-1) + F(n-2)$$

When implementing the recurrence, don't forget the memoization table!

Closed Form

The Fibonacci numbers also have a less well known closed form:

$$F(n) = \frac{1}{\sqrt{5}}\left(\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n\right)$$

Square roots introduce floating point errors. What is the maximum n this can calculate with less than 0.1 error?

Fibonacci Facts

Zeckendorf's theorem

Every positive integer can be written in a unique way as a sum of one or more distinct fibonacci numbers, which are not consecutive.

```
def zeckenfy(n):
    fibs = []
    f = greatest fib =< n; fibs.append(f)
    fibs.append(zeckenfy(n-f))
    return fibs
```

Pisano's period

The last digits of the Fibonacci sequence repeat!

The last one/two/three/four digits repeat with a period of 60/300/1500/15000.
F(6) = 8
F(66) = 27777890035288
F(366) = 1380356705549181797202918793682511
3333650564850089197542855968899086435571688

Binomial Coefficients

Definition

Binomial Coefficients are the number series that correspond to the coefficients of the expansion of a binomial:

Binom(3) = $(a+b)^3 = 1a^3 + 3ab^2 + 3ab^2 + b^3 = \{1,3,3,1\}$

We are usually interested in the k^{th} coefficient of the n^{th} binomial:

$C(n,k) = C(3,2) = \{1, 3, 3, 1\} = 3$

Pascal's Triangle gives us a good representation of C(n,n):

0	1	0	0	0	0	0	0	0	0
0	1	1	0	0	0	0	0	0	0
0	1	2	1	0	0	0	0	0	0
0	1	3	3	1	0	0	0	0	0
0	1	4	6	4	1	0	0	0	0
0	1	5	10	10	5	1	0	0	0
0	1	6	15	20	15	6	1	0	0
0	1	7	21	35	35	21	7	1	0
0	1	8	28	56	70	56	28	8	1

Uses for the Binomial Coefficient

The value of $C(n, k)$ tells us how many ways we can choose n items, k at a time.

Some use cases:

- Probabilities: What is the probability of winning a loto when you choose 5 numbers out of 60? $1/C(60, 5)$
- Grids: How many ways are there to go from the bottom left end of a *mn* grid to the top right, if you can only go up and right? $C(m + n, n)$

Calculating the Binomial Coefficient

Closed form of C(n,k)

$$C(n,k) = \frac{n!}{(n-k)!k!}$$

Problem: Multiplying factorials tends to generate huge numbers even for small n and k.

Recurrence for C(n,k)

- C(n,0) = C(n,n) = 1;
- C(n,k) = C(n-1,k-1) + C(n-1,k)

Using a memoization table will cut the calculation time by half. In this case, top-down DP will usually be faster than bottom-up.

Another useful sequence: Catalan Numbers

The Catalan sequence

$$C(n) = 1, 1, 2, 5, 14, 42, 132, 429, 1430$$

The Recurrence

$$C(n) = \sum_{k=0}^{n-1} C(k)C(n-1-k)$$

Closed Form

$$C(n) = \frac{1}{n+1}\binom{2n}{n}$$

Catalan Numbers – Uses

- Number of ways that you can match n parenthesis.
  C(3):((())),()(()),(())(),()()(),(()())
- Number of ways that you can triangulate a poligon with $n+2$ sides
- Number of monotonic paths on an *nxn* grid that do not pass above the diagonal.
- Number of distinct binary trees with n vertices
- Etc...

Integer Partition

f(5,5) = (5),(4,1),(3,2),(3,1,1),(2,2,1),(2,1,1,1),(1,1,1,1,1)

Definition and calculation

$f(n,k)$ – number of ways that we can sum n, using integers equal or less than k.

Recurrence:

- $f(n,k) = f(n-k,k) + f(n,k+1)$
- $f(1,1) = 1$; $f(n,k) = 0$ if $k > n$

Class Summary

In this lecture, we discussed challenges in math-focused problems:

- Large Integers and Log Operations;
- Number Theory:
  - Primality Testing and Prime Number Sieve;
  - Factorization;
  - Diaphantyne Equation and Linear Combinations;
- Common Combinatorics Sequences in Programming Challenges;

Next Week we will discuss geometry problems!

Problems for this Week

- Ocean Deep! - Make it Shallow!!
- Sum of Consecutive Prime Numbers
- Divisibility of Factors
- Summation of Four Primes
- How Many Trees?
- Triangle Counting
- Self-Describing sequence
- Marbles

10176 – Ocean Deep! – Make it Shallow!!

Discussed in the Lecture

Outline

You receive many binary numbers (up to 100 digits), and you must determine if each number is divisible by 131071. Example:

- 0 – YES (0)
- 1010101 – NO (85)

- You can use some bignum library;
- Or you can use mod division too;

Sum of Consecutive Primes

Outline

For a number $N \leq 10000$, determine how many different ways you can write N as a sum of consecutive primes $(p_i + p_{i+1} + \ldots + p_{i+k})$.

- You have to solve for many numbers, but the primes are always the same, so you should pre-calculate the primes.
- Remember that the primes are consecutive, so you should be able to search without backtracking.

Divisibility of Factors

Outline

Given N and d, count how many factors of $N!$ are divisible by d.

- Hint 1: You don't need to calculate $N!$, just the factorization of $N!$
- Hint 2: Think about the relationship between **Prime Factorization** and **Divisibility**

Summation of Four Primes

Outline

For a given number N, find four primes that add up to N.

- Unlike the previous problem, the four primes do not need to be consecutive;
- However, you only need to find one solution;
- This is a search problem, but you can use mathematical properties to prune your search!

How Many Trees?

Outline

Given a number of nodes with increasing labels, how many **Binary Search Trees** can you make?

- Easy combinatoric problem. Which sequence describes this situation?
- Note that the output might be a large integer.

Triangle Counting

Outline

Given an integer N, how many triangles can you make by choosing three **different** sizes $\leq N$?

Example: $N = 5$, triangles: 2,3,4; 2,4,5; 3,4,5;

- Note that testing all pairs can be too slow for large N
- You should try to find the recurrence on paper first;
  - When you add a new n in the end, how many new triangles can you make with n?

Self Describing Sequence

Outline

In the **self describing sequence**, the value $f(n)$ indicates how many times n appears in the sequence. For example, the first few numbers are:

n :	1	2	3	4	5	6	7	8	9	10	11	12
f(n):	1	2	2	3	3	4	4	4	5	5	5	6

Given a value of $n \leq 2 \times 10^9$, calculate $f(n)$.

- To calculate $f(n)$, is it necessary to calculate every value between $f(1)$ and $f(n-1)$?
- Can we skip some values?

Marbles

Outline

You have n marbles to put in boxes. Box 1 fits n_1 marbles and costs c_1. Box 2 fits n_2 marbles and costs c_2. What is the minimum cost to put all n marbles in boxes?

- This is equivalent to the "candies" problem, but you also have to think about cost.
- Remember, that there are multiple linear combinations that satisfy $n = b_1 n_1 + b_2 n_2$.
- After you calculate one pair b_1, b_2, how do you find other pairs with possibly smaller cost?

About these Slides

These slides were made by Claus Aranha, 2020. You are welcome to copy, re-use and modify this material.

Individual images in some slides might have been made by other authors. Please see the references in each slide for those cases.

Image Credits I