GB21802 - Programming Challenges

Week 7 - Math Problems

Claus Aranha
caranha@cs.tsukuba.ac.jp

College of Information Science

2015-06-10,13

Last updated June 12, 2016

Last Week Results

Week 6 - Graph II

- From Dusk Until Dawn – 4/31
- Wormholes – 18/31
- Mice and Maze – 11/31
- Degrees of Separation – 9/31
- Avoiding your Boss – 5/31
- Arbitrage – 3/31
- Software Allocation – 4/31
- Sabotage – 1/31
- Little Red Riding Hood – 9/31
- Gopher II – 4/31

- 11 people: 0 problems;
- 8 people: 1-2 problems;
- 7 people: 3-4 problems;
- 3 people: 5-6 problems;
- 2 people: 7+ problems!

Special Notes

ASPS for single weighted graphs

Apparently, this is still an open problem!

- v times BFS: $O(ve + v^2)$
- A paper (2009) claims $O(v^2 \log v)$:
  `http://waset.org/publications/8870/`
  `all-pairs-shortest-paths-problem-for-unweighted-graphs-in-o-n2-log-n-time`
  (pseudocode included!)
- This is better for dense graphs ($e \to v^2$), but for sparse graphs does not make a difference;

Math problems in programming Competitions

Math problems have a wide variety of forms, just like Graph problems. However, unlike graph problems, the programming part is easy, and the formulation is hard.

A sample of math topics in programming challenges:

- Ad-hoc: Simulation, Probability;
- Big Num: Simple problems with $n > 1000000000000000000000$
- Number Theory: Primality, Divisibility, Modulo arithmetic;
- Combinatorics: Counting, closed forms, recurrences;

In this lecture, we will just scratch the surface, focusing on examples.

Experience is the best teacher!

Ad-hoc Maths Problems

What is ad-hoc?

ad-hoc means “single purpose”. In other words, you need to improvise a solution useful for only one (or few) problems.

Ad-hoc problems usually require only a bit of elementary maths and a bit of programming;

Common categories:

- Simulation (Brute Force): Execute one or more simple formulas on a set of numbers and report the result;
- Finding patterns/formulas: Similar to simulation, but trying to execute it directly will result in TLE. You have to find a function that summarizes the formula;

Ad Hoc example: Probability problems

"What is the chance of X happening" is a common ad-hoc problem. The general formula is "prob = events of interest / total events";

The dice problem

If I have n dice, what is the chance of rolling a total above m?

- Example: For $n = 3$ dice, and $m = 16$, the chance is $10/216$

- 6,6,6
- 6,6,5
- 6,5,6
- 5,6,6

- 6,5,5
- 5,6,5
- 5,5,6

- 4,6,6
- 6,4,6
- 6,6,4

Ad Hoc example: Probabilty Problems

The dice problem

If I have n dice, what is the chance of rolling a total above m?

- For $n = 0$, we have only one result: $r = 0$
- For $n = 1$, we have 6 results: $r = \{1, 2, 3, 4, 5, 6\}$
- The result for $n = i$ and $r_{n-1} = k$ is $r_n = k + \{1, 2, 3, 4, 5, 6\}$

- Can you start to see a bottom up DP here?
- With a state table (dice,result), we can count the number of dice combination above a certain value;

Ad Hoc example: Probability Problems

Example Code

```
int count(int dice_left, int score_left) {
   if (score_left < 1) return 1;
   if (dice_left == 0) return 0;
   if (result[dice_left][score_left] != -1)
      return result[dice_left][score_left];
   int sum = 0;
   for (int i = 0; i < 6; i++)
      sum += count(dice_left-1, score_left-(i+1))
   result[dice_left][score_left] = sum;
   return sum;
}

prob = count(n,m)/6**n;
```

Other hints for ad hoc problems

- Calculating $\log_b a$: #import<cmath>; log(a)/log(b);
- Counting Digits: (int) floor(1+log10((double)a));
- n^{th} root of a: pow((double) a, 1/(double) n);

Dealing with big numbers

Many math problems require the handling of numbers much bigger than C++'s long long.

- C++ unsigned int = unsigned long = 2^{32} (9-10 digits)
- C++ unsigned long long = 2^{64} (19-20 digits)
- Factorial of 21: $>$ 20 digits!

The "DIY" approach is to transform big numbers into strings, and implement digit operations.

The programmer efficient approach is to use the JAVA BigInteger class.

Java BigInteger Class

Java's BigInteger class handles arbitrarily big numbers: Input, Output, and many operations.

```
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;
        for (int i = 0; i < N; i++) {
            BigInteger V = sc.nextBigInteger();
            sum = sum.add(V);
}}}}
```

Java BigInteger algebraic functions

- BigInteger.add(BI): $A + BI$
- BigInteger.subtract(BI): $A - BI$
- BigInteger.multiply(BI): $A * BI$
- BigInteger.divide(BI): A/BI
- BigInteger.pow(int): A^{BI}
- BigInteger.mod(BI): $A\%BI$
- BigInteger.remainder(BI): $A - A//BI$
- BigInteger.divideAndRemainder(BI): $(A//BI, A - A//BI)$

Problem Example - 10925 - Krakovia

Problem Description

Given a number of costs, and a number of friends, divide the total cost among all friends.

- Super simple problem, but requires BigInteger;
- Use this problem to familiarize yourself with writing JAVA code;

Java BigInteger superpowers

The class BigInteger implements many other functions that might be useful in mathematical problems:

- .toString(int radix) – Outputs the Big Integer, can also be used to convert base
- .isProbablePrime(int certainty) – probabilistic primality test. The chance of the test being correct is $1 - \frac{1}{2}^{\text{certainty}}$
- .gcd(BI)
- .modPow(BI exponent, BI m)

Number Theory

The field of Number theory studies the properties of integers and sets. Some problems in field include primality and modular arithmetic.

An understanding of number theory is important to avoid brute force attacks to certain problems, or to pre-process data in large problems.

Number Theory: Primality

Prime numbers are numbers >= 1 that are only divisible by 1 and themselves. There is a huge use for prime numbers, including cryptography.

- Naive calculation of prime number: For each $i \in 1..N$, test i|N
- Better calculation of prime number: For each $i \in 1..\sqrt{N}$, test i|N
- Even better calculation: For each i is prime $\in 1..\sqrt{N}$, test i|N

  number of primes up to x: $\pi(x) =$ x/logx (prime number theorem)

How can we calculate all the primes between 1 and $\sqrt{N}$ fast?

https://primes.utm.edu/howmany.html

Sieve of Eratosthenes

Main Idea

Start with a set of numbers (possible primes) between 2 and $\sqrt{N}$.
For each confirmed prime, remove all multiples of that number from the list.

```
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
```

Complexity the sieve: $O(N \log \log N)$ (Also, this is a kind of DP!)

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 such as $i|N$.

Factorization is a key issue in cryptography

Very Naive approach

For every $i \in 1..\sqrt{N}$, test $i|N$ and isPrime(i).

Very Costly!

Naive approach

Calculate a list of primes, for each prime i, test if $i|N$.

Will not find all non-prime factors!

Prime factorization: Divide and conquer approach

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 $n = 6$
  $F_m = \{5, 5\} F_{n!} = \{2, 3, 2, 2, 5, 2, 3\}$

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.

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 $k > n$

Problem Discussion (1)

Division (7)

For a, b, t, determine whether $\frac{t^a-1}{t^b-1}$ is integer with less than 100 digits.

- What are the special cases?
- What language did you use to solve it?

What base is this? (1)

Given two numbers a, b, determine what pair of bases (1..35) can be used to make $a == b$.

- Naive search: Try all bases (35x35)
- Are there any special cases?
- Can we reduce the number of bases?

Problem Discussion (2)

Divisibility of Factors (4)

Given N and D, how many factors of $N!$ are divisible by D?

- We already know how to calculate the set of prime factors.
- A factor f of $N!$ is divisible by D, if f contains all prime factos of D.

Triangle Counting (3)

If we have n sticks of size $1, 2, 3, \ldots, n$, how many different triangles can we create?

- Triangle inequality: $a + b > c$
- How many triangles can we create with $n = 3$? $n = 4$? $n = m + 1$?

Problem Discussion (3)

Help My Brother (0)

Calculate the median of the “line” composed of fibonacci numbers:

```
0 / 1 / 2 3 / 4 5 6 / 7 8 9 10 11
```

- What is the first number of line *n*? What is the last?
- How to you calculate the median of an unitary sequence?
- Note: Fibonacci above 60 are VERY BIG, use tables!

Marbles (0)

- Diaphantine equation, but now with a cost associated for *a* and *b*.
- Calculate the feasible range for *n*, then find *n* with minimal cost.

Problem Discussion (4)

Ocean Deep, Make it Shallow! (1)

Use modulo operation to break a binary number, and test it for divisibility with 131071

Winning Streak (0)

Calculate the probability for all the events, and use the formula to calculate the expected length of the winning streak.

Class Summary

- Math Problems
- Java's Big Integer class
- Primality
- Modulo arithmetic
- GCD and Diophantine Equations
- Combinatorics

Next week: Geometry problems!

This Week's Problems

- Division
- What Base Is This
- Divisibility of Factors
- Triangle Counting
- Help My Brother
- Marbles
- Ocean Deep!
- Winning Streak

To Learn More

Euler Project: Mathematical questions using computers:
`http://projecteuler.net`