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 9, 2016

Last Week Results

Week 6 - Graph II

- From Dusk Until Dawn – 1/31
- Wormholes – 11/31
- Mice and Maze – 6/31
- Degrees of Separation – 5/31
- Avoiding your Boss – 4/31
- Arbitrage – 0/31
- Software Allocation – 3/31
- Sabotage – 0/31
- Little Red Riding Hood – 6/31
- Gopher II – 3/31

- 19 people: 0 problems;
- 3 people: 1 problem;
- 3 people: 2 problems;
- 2 people: 3-4 problems;
- 2 person: 6-7 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 u

Some Programming Hints

Calculating Logb(a): import cmath, log(a)/log(b)
Counting Digits: (int) floor(1+log10((double)a))
Square root n of a: pow((double) a, 1/(double) n)

Example Problem: Probability

Throw x dice, what is the chance of result > m?
Chance = #desired outcomes / #all outcomes
This can be solved by DP!
(roundoff errors, simplifying fractions)

Dealing with big numbers

C++ unsigned int = unsigned long = 2^{32} (9-10 digits) C++ unsigned long long: 2^{64} (19-20 digits) Factorial of 21: $>$ 20 digits
For math problems with relatively small *n*, we need bignum;
The “DIY” approach is to transform the numbers in strings, and implement digit by digit operators. The programmer efficient approach is to use the JAVA BigInteger class

Java Bignum Class

BigInteger.add(BI) BigInteger.subtract(BI)
BigInteger.multiply(BI) BigInteger.divide(BI) BigInteger.pow(int)
BigInteger.mod(BI) BigInteger.remainder(BI)
BigInteger.divideAndRemainder(BI)

Problem Example - 10925 - Krakovia

Problem Description

Java BigInteger superpowers

.toString(int radix) – converts base
.isProbablePrime(int certainty) – probabilistic primality test
$1 - \frac{1}{2}^{\text{certainty}}$ (for “small” primes, we can use the Sieve of Erasthothenes which we will see later)
.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 i in 1:N, test i//N
Better calculation of prime number: For i in 1:sqrtN, test i//N
Even better calculation of prime number: For i in primes[1:sqrtN] test i//N
Can we calculate the primes[1:sqrtN] fast? pi(x) = x/logx (prime number theorem)

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

Sieve of Eratosthenes

Idea
Code
Complexity of primality testing: O(NloglogN)

Finding Prime Factors

Prime factors: is the list of all primes that divide N
Naive approach: If you have a list of primes, try to divide each prime in that list by N
Divide and conquer approach: Find the smallest prime factor i of N. Now find the smallest prime factor i' of N//i, recursively. This is slow if the number has few, big prime factors, but very fast if the number has many, small primes. This is the principle of our cryptography!

Working with Prime Factors

Prime factors are the building blocks of all integers. Therefore, sometimes we can work with prime factors instead of working with huge integers.

Example: Problem 10139: Factorvision

Problem description: Does *m* divides *n*!? (m,n $< 2^{31} - 1$).

Factor of 22 is already bignum, we cannot work with the factors directly!

Solution: Calculate the prime factors of *m* and *n*!, and see if *n*! contains *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. $(a^n)\%s = ((a^{n/2}\%s) * (a^{n/2}\%s) * (a^{n\%2}\%s))\%s$

UVA 10176 – Ocean Deep!

Test if a 100-digit binary is divisible by 131071.

We can calculate the module of each (binary) digit using the recurrence in (3), and module sum the result without ever touching the entire 2^{100} number.

Euclid Algorithm and Extended Euclid Algorithm

- The Euclid Algorithm gives us the maximum common divisor D of a and b;
- The 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

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

Combinatory problems usually involving finding out the recurrence of a set (recursive function that construct the set) or the closed form of a set (formula that calculates the nth element of the set)

Depending on the recurrence or formula, the need of bignums is not uncommon either. Also, often the recurrence can be sped-up by using DP.

Let's see some examples.

Fibonacci Numbers

fib(0) = 0, fib(1) = 1, fib(n>2) = fib(n-1)+fib(n-2)
Usually this is implemented based on a O(n) table, as a complete recursion is very slow.
However, there is an approximation that is O(1): calculate the closest integer to $\frac{(phi^n-(-phi)^{-n})}{\sqrt{5}}$ where phi is the golden ratio $\frac{1+\sqrt{5}}{2}$ (this is not accurate for very large fibonacci numbers).

Funny Fibonacci

Binomial Coefficients

^nC_k the number of ways that n items can be selected k at a time, also the coefficients of the $(x+y)^n$ polinomial.
A single value of the binomial can be calculated as $\frac{n!}{(n-k)!k!}$.
This formula is problematic, however. Not only factorials are very large for small n, multiplyign factorials have a good chance of exploding.
Some tricks to avoid using bignum * If k > n-k. then exchange k and n-k. * Try to divide before multiplyign * top down dynamic programming: C(n,0) = C(n,n) = 1 C(n,k) = C(n-1,k-1) + C(n-1,k).
In this case, the recursive formula is more useful than the closed form.

Catalan Numbers

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

They are calculated as: Cat(n) = $\frac{({}^{2n}C_n)}{(n+1)}$, Cat(0) = 1

Or

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

Catalan Numbers – Uses

- Number of ways that you can match n parenthesis.
- 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...

Class Summary

- Math Problems!

This Week's Problems

- UVA Problems

To Learn More

Euler Project