This course material is now made available for public usage.
Special acknowledgement to School of Computing, National University of Singapore
for allowing Steven to prepare and distribute these teaching materials.

CS3233
Competitive Programming

Dr. Steven Halim

Week 09 – Mathematics in Programming Contests

Outline

- Mini Contest #7 + Discussion + Break + Admins
- Mathematics-Related Problems & Algorithms
  - Ad Hoc Mathematics Problems (quick overview)
    - Those that do not need specific algorithm, just basic coding/math skill
  - Java BigInteger Class
  - Number Theory, especially Prime Factors and Modulo Arithmetic
  - Many other topics are for self-reading at home (CP2.9)

Mathematics, CS, and ICPC/IOI (1)

- Computer Science is deeply rooted in Maths
  - Compute = Math
- It is not a surprise to see many Maths problems in ICPC (PS: IOI tasks are usually *not* Maths-specific)
  - Many of which, I do not have time to teach you…
  - Few others, I cannot teach you as I do not know them yet…
  - CS3233 is NOT a pure Mathematics module
    - Only 1 week (1.5 hours) is devoted for Mathematics-related topic
  - It is nice if we can improve our ranks by solving some mathematics problems in programming contest

Mathematics, CS, and ICPC/IOI (2)

- Tips:
  - Revise your high school mathematics
  - (In NUS): Take MAXXXX modules as CFM :D
  - Read more references about powerful math algorithms and/or interesting number theories, etc
  - Study C++ <cmath> & Java.Util.Math/Java.Math Library
  - Try maths problems in UVa/other OJ and at projecteuler

The Lecture Plan (more at home)

- Today, we will discuss a small subset of this big domain
- Plan:
  - We will skip/fast forward the "not so interesting" stuffs
  - I will give several Maths-related pop-quizzes using clicker system to see how far you know these tricks…
  - We will focus on **several related subjects**:
    Big Integer, Prime Factors, and Modulo Arithmetic
    - All involve "Big (Large) Integers"...
  - You will then have to read Chapter 5 of CP2.9 **on your own** (it is a huge chapter btw…)

Mathematics-Related Problems
(as currently listed in CP2.9)

1. **Ad Hoc Mathematics**
   1. The Simple Ones
   2. Mathematical Simulation (Brute Force)
   3. Finding Pattern or Formula
   4. Grid
   5. Number Systems or Sequences
   6. Logarithm, Exponentiation, Power
   7. Polynomial
   8. Base Number Variant
   9. Just Ad Hoc
2. **Java BigInteger**
   1. Basic Features
   2. Bonus Features
3. **Combinatorics**
   1. Fibonacci Numbers
   2. Binomial Coefficients
   3. Catalan Numbers
   4. Others
4. **Number Theory**
   1. Prime Numbers: Sieve of Eratosthenes
   2. GCD & LCM
   3. Factorial
   4. Prime Factors
   5. Working with Prime Factors
   6. Functions involving Prime Factors
   7. Modified Sieve
   8. Modulo Arithmetic
   9. Extended Euclid/Linear Diophantine Equation
   10. Others
5. **Probability Theory**
6. **Cycle-Finding**
   1. Floyd's Tortoise-Hare Algorithm
7. **Game Theory**
   1. Two Players Game, Minimax
   2. Nim Game (Sprague Grundy Theorem)
8. **Also see Chapter 9**

Programming problems that are from the domain of mathematics,
but we do not need specialized data structure(s) or algorithm(s) to solve them

We will do A QUICK SPLASH AND DASH… Learn the details at home ☺

Section 5.2

AD HOC MATHEMATICS

The Simpler Ones

- Nothing to teach ☹
- They are too simple, really…
- You can get ~10 ACs in < 1 hour if you solve all problems listed in this category in CP2.9 ☺

Mathematical Simulation (Brute Force)

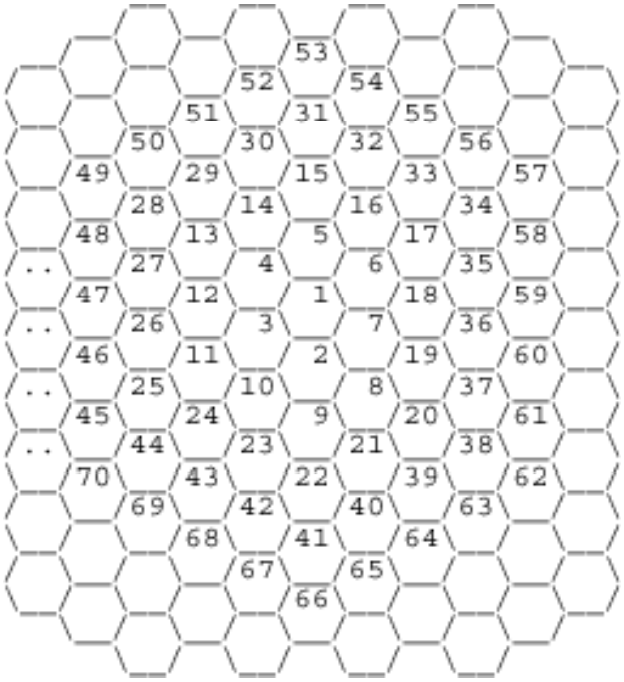
- Nothing to teach other than the ones already presented during iterative/recursive "Complete Search" topic
  - Just remember to prune the search space whenever possible…
- Note: Problems that require other technique (like number theory knowledge) and cannot be solved with brute-force are NOT classified in this category

Finding Pattern or Formula

- This requires your mathematical insights to obtain those patterns/formulas **as soon as possible** to reduce the time penalty (in ICPC setting)
- Useful trick:
  - Solve some small instances by hand
  - List the solutions and see if there is/are any pattern(s)?
- Let's do a quick exercise ☺

Grid

- Also about finding pattern
- It requires ***creativity*** on manipulating the grid or converting it to simpler ones
- Example:



Number Systems or Sequences

- Nothing much to teach :O
- Most of the time, carefully following the problem description is sufficient

Logarithm, Exponentiation, Power

- In C/C++ `<cmath>`, we have `log` (base *e*) and `log10` (base 10)
- In Java.lang.Math, we only have `log` (base *e*)
- To do $\log_b(a)$ (base *b*), we can use: `log(a) / log(b)`
- Btw, what does this code snippet do? `(int)floor(1 + log10((double)a))`
- And how to compute the n-th root of a?

Polynomial

- Representation: Usually the coefficients of the terms in some sorted order (based on power)
- Polynomial formatting, evaluation, derivation (Horner's rule), division, remainder, roots (Ruffini's rule)…
- The solution usually requires careful loops…

Base Number Variants

- Do you know that base number conversion is now *super easy* with Java BigInteger?
- However, for some variants, we still have to go to the basics method…
  - The solution usually use base 10 (decimal) as an intermediate step

And A Few Others…

- Best way to learn these: Via practice…

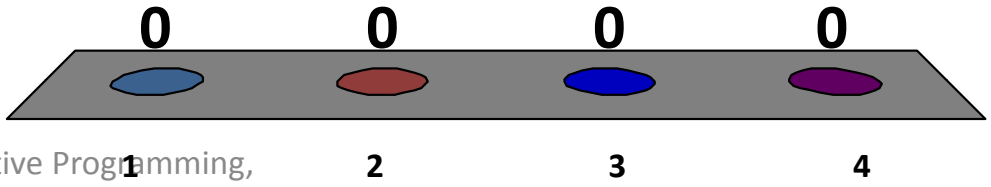
Section 5.3

A Powerful API for Programming Contests

JAVA BIGINTEGER CLASS

Java BigInteger…

1. I am a Java user but I have **never** used it before
2. I am a (pure) C++ user so I **never** used it before
3. I am a Java user and I have used it before ☺
4. I am bilingual (Java/C++) and I have used it before ☺



Big Integer (1)

- Range of default integer data types (C++)
  - unsigned int = unsigned long: 2^{32} (9-10 digits)
  - unsigned long long: 2^{64} (19-20 digits)
- Question:
  - What is “777!”, i.e. factorial of 777?
- Solution?
  - Big Integer: Use **string** to represent number
    - ~ number can be as long as computer memory permits
    - FYI, this is similar to how basic data types are stored in computer memory. Just that this time we do not have limitation of the number of bits (digits) used…

Big Integer (2)

- Operations on Big Integer
  - Basic: add, subtract, multiply, divide, etc
  - Use “high school method”
    - Some examples below:

```
 1  ← carry
218
 45
--- +
263
```

```
 218
  45
 --- x
1090   (218*5)
 872   (218*4)*10
----- +
9810
```

Big Integer (3)

- Note:
  - Writing these "high school methods" during stressful contest environment is **not a good strategy**!
- Fortunately, Java has BigInteger library
  - They are allowed to be used in contests (ICPC and CS3233)
    - So use it…
  - Note: IOI does not allow Java yet,
    and anyway, I have not see BigInteger-related tasks in IOI…
- Or, if you insist, build your own BigInt library and bring its hardcopy to future contests!

Java BigInteger Class

- This class is rather powerful
  - Not just it allows for basic mathematical operations involving big integers (addition, subtraction, multiplication, division, mod or remainder, and power)…
  - It also provides support for:
    - Finding GCD of big numbers
    - Finding the solution of x^y mod m (modulo arithmetic)
    - Very Easy Base Number Conversion, quite useful
    - NEW in CP2.9:
    - See various examples in the book ☺

Section 5.4

COMBINATORICS

Combinatorics

- Given problem description,
  find some nice formula to **count something**
  - Coding is (usually very) short
  - Finding the formula is not straightforward…
    - If formula has overlapping sub problems → use DP
    - If formula yield huge numbers → use Java BigInteger
- Memorize/study the basic ones: Fibonacci-based formulas, Binomial Coefficients, Catalan Numbers…
- PS: On-Line Encyclopedia of Integer Sequences
  - Can be a good reference: **http://oeis.org/**

Programming problems that requires the knowledge of number theory, otherwise you will likely get Time Limit Exceeded (TLE) response for solving them naively…

Section 5.5

NUMBER THEORY

Prime Numbers

- First prime and the only even prime: 2
- First 10 primes: {2, 3, 5, 7, 11, 13, 17, 19, 23, 29}
- Primes in range:
  - 1 to 100 : 25 primes
  - 1 to 7,919 : 1,000 primes
  - 1 to 1,000 : 168 primes
  - 1 to 10,000 : 1,229 primes
- Largest prime in signed 32-bit int = 2,147,483,647
- Used/appear in:
  - Factoring
  - Cryptography
  - Many other problems in ICPC, etc

Optimized Prime Testing

- Algorithms for testing if N is prime: isPrime(N)
  - First try: check if N is divisible by $i \in [2 .. N-1]$?
    - O(N)
  - Improved 1: Is N divisible by $i \in [2 .. sqrt(N)]$?
    - O(sqrt(N))
  - Improved 2: Is N divisible by $i \in [3, 5, .. sqrt(N)]$?
    - One test for i = 2, no need to test other even numbers!
    - O(sqrt(N)/2) = O(sqrt(N))
  - Improved 3: Is N divisible by $i \in$ primes $\leq$ sqrt(N)
    - $O(\pi(sqrt(N))) = O(sqrt(N)/log(sqrt(N)))$
      - $\pi(M)$ = num of primes up to M
      - For this, we need smaller primes beforehand

Prime Generation

- What if we want to generate a list of prime numbers between [0 … N]?
- Slow naïve algorithm:

```
Loop i from [0 … N]
  if (isPrime(i))
    print i
```

- Can we do better?
  - Yes: Sieve of Eratosthenes

Sieve of Eratosthenes Algorithm

- Generate primes between [0 … N]:
  - Use bitset of size N, set all true except index 0 & 1
  - Start from i = 2 until k*i > N
    - If bitset at index i is on, cross all multiple of I (i.e. turn off bit at index i) starting from i*i
  - Finally, whatever not crossed are primes
- Example:
  - 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, …, 51, 52, 53, 54, 55, …, 75, 76, 77, …
  - 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, …, 51, 52, 53, 54, 55, …, 75, 76, 77, …
  - 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, …, 51, 52, 53, 54, 55, …, 75, 76, 77, …
  - 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, …, 51, 52, 53, 54, 55, …, 75, 76, 77, …
  - 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, …, 51, 52, 53, 54, 55, …, 75, 76, 77, …

Code: sieve & isPrime

```
#include <bitset>       // compact STL for Sieve, better than vector<bool>!
ll _sieve_size;                  // ll is defined as: typedef long long ll;
bitset<10000010> bs;                // 10^7 should be enough for most cases
vi primes;                 // compact list of primes in form of vector<int>

void sieve(ll upperbound) {     // create list of primes in [0..upperbound]
  _sieve_size = upperbound + 1;              // add 1 to include upperbound
  bs.set();                                          // set all bits to 1
  bs[0] = bs[1] = 0;                                // except index 0 and 1
  for (ll i = 2; i <= _sieve_size; i++) if (bs[i]) {
    // cross out multiples of i starting from i * i!
    for (ll j = i * i; j <= _sieve_size; j += i) bs[j] = 0;
    primes.push_back((int)i);       // add this prime to the list of primes
} }                                      // call this method in main method

bool isPrime(ll N) {            // a good enough deterministic prime tester
  if (N <= _sieve_size) return bs[N];              // O(1) for small primes
  for (int i = 0; i < (int)primes.size(); i++)
    if (N % primes[i] == 0) return false;
  return true;               // it takes longer time if N is a large prime!
}                // note: only work for N <= (last prime in vi "primes")^2
```

Greatest Common Divisor (GCD)

- Naïve Algorithm:
  - Find all divisors of a and b (slow)
  - Find those that are common
  - Pick the greatest one
- Better & Famous algorithm: D & C Euclid algorithm
  - GCD(a, 0) = a
  - GCD(a, b) = GCD(b, a % b) // problem size decreases a lot!!
- Its recursive code is easy to write:

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

Lowest Common Multiple (LCM)

- lcm(a, b) = a*b / gcd(a, b)

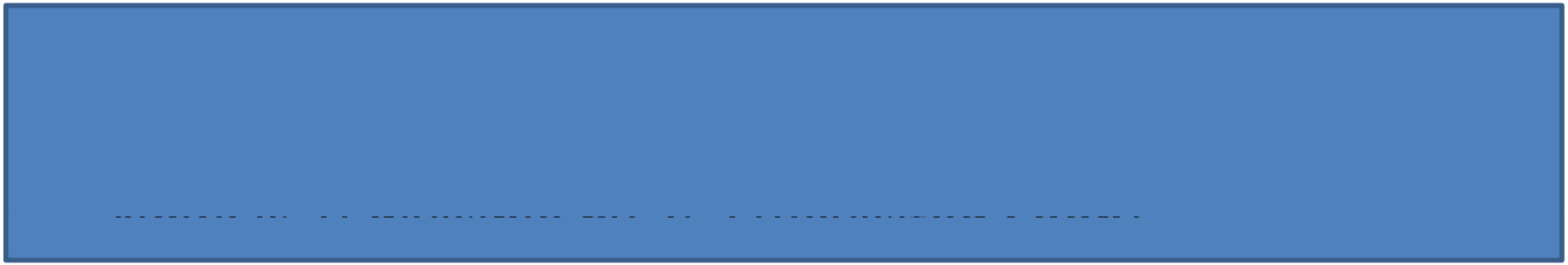
```
int lcm(int a, int b) { return (a / gcd(a, b) * b); }
// Q: why we write the lcm code this way?
```

- Note for gcd/lcm of more than 2 numbers:
  - gcd(a, b, c) = gcd(a, gcd(b, c));

- Both gcd and lcm runs in $O(\log_{10} n)$ where n = max(a, b)

Factorial

- What is the highest **n** so that **factorial(n)** still fits in 64-bits unsigned long long?
  - Answer: n = 20
    - `20! = 2432902008176640000`
    - `ull = 18446744073709551615`
    - `21! = 51090942171709440000`
- Hm... so almost all factorial related questions require Java BigInteger?



Prime Factors

- Direct algorithm: Generate list of primes (use sieve), check how many of them can divide integer N
  - This can be improved!
- Better algorithm: Divide and Conquer!
  - An integer N can be expressed as:
    - N = PF * N'
      - PF = a **prime factor**
      - N' = another number which is N / PF
    - If N' = 1, stop; otherwise, repeat
    - N is reduced every time we find a divisor

But if integer I is a large prime, then this is still slow.

This fact is the basis for cryptography techniques

Code: Prime Factors

```
vi primeFactors(ll N) {     // remember: vi is vector<int>, ll is long long
  vi factors;
  ll PF_idx = 0, PF = primes[PF_idx];  // PF = 2, then 3,5,7,... is also ok
  while (N != 1 && (PF * PF <= N)) {  // stop at sqrt(N); N can get smaller
    while (N % PF == 0) { N /= PF; factors.push_back(PF); }    // remove PF
    PF = primes[++PF_idx];                                    // only consider primes!
  }
  if (N != 1) factors.push_back(N);          // special case if N is a prime
  return factors;     // if N does not fit in 32-bit integer and is a prime
}                   // then `factors' will have to be changed to vector<ll>
```

Ok… that's enough for this semester ☺

THE OTHER MATHS PROBLEMS IN PROGRAMMING CONTEST

Not Covered in Lecture this Sem

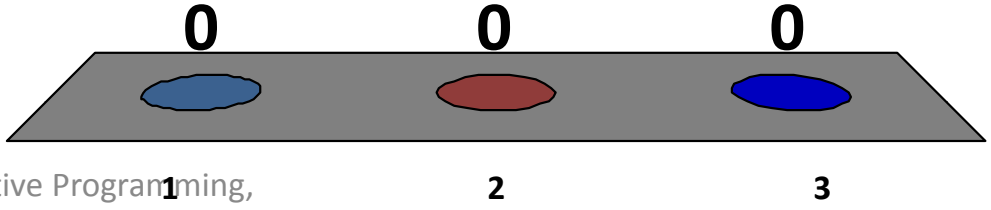
- Linear Diophantine Equation (Section 5.5)
- Probability Theory (Section 5.6)
- Cycle-Finding (Section 5.7)
- Game Theory (Section 5.8)
- Gaussian Elimination (Section 9.4)
- Matrix Power (Section 9.13)
- Roman Numerals (Section 9.20)
- They are already written in CP2.9
  - They are good read ☺
  - Read them on your own
  - These problems will not appear as problem A or B in mini contest 8

Many More Still Not in CP2.9 Yet…

- Mathematics is a large field
  - Pollard's Rho integer factoring algorithm
  - Many other Prime theorems, hypotheses, conjectures
  - Chinese Remainder Theorem
  - Lots of Divisibility Properties
  - Combinatorial Games, etc
- Again CS3233 != Math Module
- Chapter 5 of CP2.9 has a collection of ~**372** UVa programming exercises… the highest among the 9 chapters in CP2.9!

The Pace of this Lecture

1. Too slow, I already know all these…
   I want to know *more*
2. Fine; we are used to it now ☺
3. Crazy as always…
   and we still have lots of reading material at home



Summary

- We have seen *some* mathematics-related problems and algorithms
  - Too many to be learned in one night…
  - Even so, many others are left uncovered
    (some are inside CP2.9 for you to read on your own pace)
  - Best way to learn: Lots of practice
- In the next two weeks, two more ***new topics***:
  - Week 10: String Processing **(Focus on SA)**
  - Week 11: (Computational) Geometry **(Focus on Polygons)**

References

- CP2.9, Chapter 5 ☺ and parts of Chapter 9
- Introduction to Algorithms, Ch 31, Appendix A/B/C
- Project Euler, http://projecteuler.net/