CS35X: Competitive Programming

Lecture 7: Math

Joshua Brody

Warmup Kattis Problem: Missing number

Problem debrief: orphan backups

Math Problems in Competitive Programming

Many problems in competitive programming are mathematical. Some involve making clever observations, some involve recalling little-used mathematical algorithms. Some are common though.

CS35x goals for mathematical problems:

- Handle large integers
- Compute prime numbers
- generate factors of a number

C++ large integers, modular arithmetic

- Some problems ask you to work with numbers with thousands of digits
  - Often you'll need to print the number modulo some large prime.
  - e.g. Print the number of solutions to [this problem] modulo `998244353`.
- In C++ always use `long long` for these problems.
  - Always take intermediate results mod the large prime.
  - **Note:** in C++ the mod operator (`%`) doesn't work the way you'd expect on negative numbers: `-5 % 3 == -2`

Example: read array of N elements, compute product

```
const int MOD = 1000000007;


int N; cin >> N;
long long prod = 1;
for(int i=0; i<N; i++) {
  long long X; cin >> X;
  prod = (prod*X) % MOD;
}
cout << prod << endl;
```

Prime Factorization

The most straightforward way to factor a number n is to perform trial divisions up to sqrt(N):

```
vector<int> factor(int N) {
  vector<int> result;
  for(int i=2; i<sqrt(N); +1 && i<N; i++) {
    while(N%i ==0) {
       result.push_back(i);
     }
  }
   if(N>1) {
     result.push_back(N);
   }
  return result;
}
```

Generate all primes

The classic way: **Sieve of Eratosthenes**

This runs in `O(nlog(log(n)))` time

Faster Prime Generation: Euler Sieve

```
vector<int> lpf;          // lpf[i] stores least prime factor of i
vector<int> primes;

void computePrimes(int N) {
  lpf = vector<int>(N+1);
  for(int i=2; i<=N; i++) {
    if(lpf[i]==0) {
      lpf[i] = i;  primes.push_back(i);    // found a new prime!
    }
    for(int j=0; i*primes[j]<=N; j++) {
      lpf[i*primes[j]] = primes[j];
      if(primes[j] == lpf[I]) {
        break;
} } } }
```

This runs in `O(n)` time

Faster Prime Factorization

With Euler Sieve, it is easy to prime factorize a number

```
Vector<int> factor(int N) {
  Vector<int> result;
  while(N!=1) {
    Result.push_back(lpf[N]);
    N = N/lpf[N];
  }
  Return N;
}
```

Compact Prime Factorization

- For some problems, you'll need to iterate over all factors of a number, not just all prime factors.
  - One solution: enumerate over subsets of prime factorization
  - This can be inefficient if primes are repeated
- Compact prime factorization:
  - List of `[p,e]` pairs, where `p^e` divides `N`.

Exercise:
Read in a list of primes from cin,
Produce compact prime list.
Ex input: `[2,2,2,3,5,5,19]`
Output: `[[2,3],[3,1],[5,2],[19,1]]`

Kattis Problem: popkorn