Combinatorics

- Concerned with counting various objects, configurations, etc.
- Often solved by recurrences.
- Make sure the data type is large enough to store the result (e.g. use BigInteger).

Fibonacci Numbers

- Fibonacci numbers grow very quickly.
- $f(0) = 0$, $f(1) = 1$, $f(n) = f(n-1) + f(n-2)$
- An alternate way to compute Fibonacci numbers (or other recurrences):

$$\begin{bmatrix} f(n) \\ f(n+1) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} f(n-1) \\ f(n) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}^n \cdot \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

- Use fast exponentiation to compute matrix power in $O(\log n)$ steps.
- See `exp.cc` or `expmod.cc` for fast exponentiation of integers.
- Useful for computing $f(n)$ for large n.

Binomial Coefficients

- Compute with dynamic programming (Pascal's triangle).
- If answer does not overflow, no overflow can occur during the calculation using DP.

Catalan Numbers

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

- Number of binary trees with n nodes.
- Number of ways to bracket an expression of n operators.
- Number of ways to write n pairs of parenthesis which are correctly matched.
- Number of ways to triangulate a convex polygon with $n+2$ sides.
- Recurrence:

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

Prime Numbers

- A positive integer ≥ 2 is prime if its only divisors are 1 and itself.
- To determine if a number n is prime, use trial division up to $\sqrt{n}$. This is good enough for most 32-bit integers.

Sieve of Eratosthenes

- If we want to test for primality in a range, it is inefficient to test each one by trial division.
- Given a prime p, it is easier to mark off the multiples of p.
- Set up a boolean array for the range, and initialize all of them to be true.
- For each relevant prime p, mark off the entries for p, $2p$, $3p$, $\ldots$
- Some optimization: consider primes in increasing order, and start from $p \cdot p$, $(p+1) \cdot p$, $\ldots$
- Complexity to check the first n integers: $O(n \log \log n)$.

GCD

- $\gcd(a, b)$ is the largest (positive) integer dividing both a and b.
- Computed by the Euclidean algorithm: `euclid.cc`. Very fast.
- To compute the GCD of three or more integers, use:

$$\gcd(a, b, c) = \gcd(\gcd(a, b), c)$$

LCM

- LCM is related to the GCD (for two integers):

$$\operatorname{lcm}(a,b) = \frac{a}{\gcd(a,b)} \cdot b$$

  It is important to divide by GCD first to avoid overflow.

- For three or more integers:

$$\operatorname{lcm}(a,b,c) = \operatorname{lcm}(\operatorname{lcm}(a,b),c)$$

  This is **NOT** the same as:

$$\frac{abc}{\gcd(a,b,c)}$$

Prime Factorization

- Many problems can be solved easier if we factored the integers into prime factors.
- Relatively easy for "small" (32-bit) integers.
- See `factor.cc`.
- For large integers this is very hard. See `factor_large.cc` if you want to use one possible advanced algorithm.
- See 5.5.5 and 5.5.6 for problems that can be solved easily once factorization is obtained.

Linear Diophantine Equations

- Given a, b, and c, you want to find integer solutions s and t such that
$$a \cdot s + b \cdot t = c.$$
- Solution exists iff $\gcd(a, b)$ is a divisor of c.
- Use the extended Euclidean algorithm (`exteuclid.cc`) to find s' and t' such that
$$a \cdot s' + b \cdot t' = \gcd(a, b)$$
- Multiply solutions by $c/\gcd(a, b)$ to get one solution.
- If (s, t) is a solution, the rest of the solutions can be obtained from
$$(s + k \cdot b/\gcd(a, b), t - k \cdot a/\gcd(a, b))$$
where k is any integer.
- Sometimes we are only interested in solutions in a certain range (e.g.

both are positive).