**Greatest Common Divisor (GCD)**

gcd(a, b): greatest integer divides both a and b

If b|a then gcd(a,b) = b

Otherwise a = bt+r for some t and r

gcd(a,b) = gcd(b,r)

gcd(a,b) = gcd(b,a%b)

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

Recursive:

int gcd(int a, int b){

if (b==0)

return a;

else

return gcd(b,a%b);

}

Iterative:

int gcd(int a, int b) {

while(b){

int r = a % b;

a = b;

b = r;

}

return a;

}

Running time: O (log (a + b))

**Modular Exponentiation**

Compute a^n in O(log n) time

a^n = 1, if n=0

= a if n=1

= a^{n/2}^2, if n =even

= a^{(n−1)/2}^ 2, if n = odd

Implementation:

double pow(double a, int n) {

if(n == 0) return 1;

if(n == 1) return a;

double t = pow(a, n/2);

return t \* t \* pow(a, n%2);

}

Iterative solution:

a = a0 + a\_1\*2 + a\_2\*2^2+...+a\_k\*2^k

int result=1,power=a

while(!n){

if(n&1)

result\*=power

power\*=power

n>>=1;

}

**Efficiently Computing F\_n**

F\_n = F\_{n-1} + F\_{n-2}

F\_n = [ 1 1] [F\_{n-1}] //make them a single line

F\_{n-1}= [1 0] [F\_{n-2}]

F\_n = [ 1 1] ….. [F\_{1}] //make them a single line

F\_{n-1}= [1 0] ….... n times[F\_{0}]

Compute the product in O(lg n) time

\alert{Can be extended to support any linear recurrence with constant coefficients}

**Binomial Coefficients**

nCk = is the number of ways to choose k objects out of n distinguishable objects

Computing nCk

Solution 1: Compute using the following formula

nCk = n(n − 1) ⋯ (n − k + 1)/ k!

Solution 2: Use Pascal’s triangle

Case 1: Both n and k are small

Use either solution

Case 2: n is big, but k or n − k is small

Use Solution 1 (carefully)

Lucas Theorem:

The Lucas' theorem expresses the remainder of division of the binomial coefficient mCn by a prime number p in terms of the base p expansions of the integers m and n.

For non-negative integers m and n and a prime p, the following congruence relation holds:



where



and



are the base p expansions of m and n respectively.

Source: Wikipedia

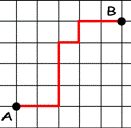
**Problem 1:**

Find the number of strings of length “N” made up of only 3 characters – a, b, c such that “a” occurs at least “min\_a” times and at most “max\_a” times, “b” occurs at least “min\_b” times and at most “max\_b” times and “c” occurs at least “min\_c” times and at most “max\_c” times. Note that all permutations of same string count as 1, so “abc” is same as “bac”.

\url{<http://www.spoj.pl/problems/DCEPC702/>}

**Problem 2: Method of paths (or trajectories).**

The main idea is to find a geometrical interpretation for the problem in which we should calculate the number of paths of a special type. More precisely, if we have two points A, B on a plane with integer coordinates, then we will operate only with the shortest paths between A and B that pass only through the lines of the integer grid and that can be done only in horizontal or vertical movements with length equal to 1

 Source: <http://community.topcoder.com/tc?module=Static&d1=tutorials&d2=combinatorics>

Solution: All paths between A and B have the same length equal to n+m (where n is the difference between x-coordinates and m is the difference between y-coordinates). We can easily calculate the number of all the paths between A and B:

Ans: (m+n)Cn or (m+n)Cm

**Problem 3:**

Let’s solve a famous problem using this method. The goal is to find the number of Dyck words with a length of 2n. What is a Dyck word? It's a string consisting only of n X’s and n Y’s, and matching this criteria: each prefix of this string has more X’s than Y’s. For example, “XXYY” and “XYXY” are Dyck words, but “XYYX” and “YYXX” are not.

Or

Find the number of ways to arrange n ‘(‘ and n’)’ brackets such that at each index, the number of ‘(‘ is never less than the number of ‘)’

Solution: Total ways: 2nCn

Incorrect ways: 2nC(n-1) How?? Think!

Ans: Catalan number 2nCn /(n+1)

**Modular Arithmetic**

(x+y) mod n = ((x mod n) + (y mod n))mod n

(x-y) mod n = ((x mod n) – (y mod n))mod n

(x\*y) mod n = (x mod n)(y mod n)mod n

\alert{But, (x/y) mod n = ((x mod n)/(y mod n))mod n, not always true}

(x^y) mod n = (x mod n)^y mod n

**Multiplicative inverse**

x^{−1} is the inverse of x modulo n if xx^{−1}≡ 1⁡(mod⁡n)

5^{−1} ≡ 3⁡(mod⁡7) because 5 ⋅ 3 ≡ 15 ≡ 1⁡(mod⁡7)

May not exist (e.g. Inverse of 2 mod 4)

Exists iff gcd( x, n) = 1

gcd (a, b )= ax + by for some integers x, y

If gcd (a, n )= 1, then ax + ny = 1 for some x, y

Taking modulo n gives ax ≡ 1⁡(mod⁡n)

Given a,b, Finding x and y, such that ax+by = d is done by **Extended Euclid's algorithm**

**Prime Sieve**

Idea: every composite number n has a prime factor p ≤ √n. So let us assume that all numbers are prime. But if we come across a prime factor of a number, we immediately know that it is not a prime. If there is no prime factor of a number n in the range [2..n-1] then it must be prime. This given rise to the following sieve:

Generate all primes in range [1..n]

For i=1 to n

prime[i]=1

Prime[1]=0

For i=2 to √n

if(prime[i])

for j = i to n/i

prime[i\*j]=0

At the end of this step, all numbers i which are

prime have prime[i]=1. Others have prime[i]=0.

Prime number Theorem

Number of primes till n ~ n/logn

Maximum number of prime factors of n

log n

**Euler’s totient function \phi(n)**

\phi(n) = Number of positive integers less than or equal to n which are coprime to n

Φ(ab) = φ(a) φ(b)

• For a prime p φ(p) = p-1

• For a prime p φ(p^k) = p^k-p^{k-1} = p^k(1-1/p)

N = (p\_1^{k\_1})\*(p\_2^{k\_2})\*...\*(p\_t^{k\_t})

• \phi(n)=p\_1^{a\_1}p\_2^{a\_2}.. p\_t^{a\_t} ) = n\*(p\_1-1)\*(p2\_-1)..(p\_t-1)/(p\_1 p\_2..p\_t)

Sieve for \phi(n)

One approach:

run prime sieve once and store result in primes[1..n]

For (i=1 to n) phi[i]=i

For(i=2 to n)

if(prime[i])

for(j=1 to n/i)

phi[i\*j]=phi[i\*j]\*(i-1)/i

This algo runs in O(n log log n) time, but we will make improvements to show how you can at times optimize your code.

Question: do we really need to generate list of all primes?

Answer: no

For (i=1 to n) phi[i]=i

For(i=2 to n)

if(phi[i]==i)

for(j=i to n;j+=i)

phi[j] = (phi[j]/i)\*(i-1);

**Fermat’s little theorem:**

If p is a prime number, then for any integer a that is coprime to n, we have

a^p ≡ a (mod p)

This theorem can also be stated as: If p is a prime number and a is coprime to p, then

a^{p -1} ≡ 1 (mod p)

**Euler’s Theorem**

• Euler’s Theorem is a genaralization for Fermat's little theorem

• When a and n are co-prime

• if x ≡ y (mod φ(n)), then ax ≡ ay (mod n).

• a^{φ(n)} ≡ 1 (mod n) (actual theorem the above is a generalization)

The number of divisors:

If n = p\_1^{a\_1})\*(p\_2^{a\_2})\*...\*(p\_t^{a\_t}, then the number of its positive divisors equals to

(a\_1 + 1) \* (a\_2 + 1) \* … \* (a\_t + 1)

sum of the divisors of n equals

\prod\_{i=1}^k {p\_i^{m\_i + 1} - 1 \over p\_i - 1}

**Miller Rabin Test (Primality testing)**

Input: n > 1, an odd integer to test for primality.

write n−1 as 2^s·d by factoring powers of 2 from n−1

repeat for all :a ∈ [2, min n − 1, 2 ln n 2) )]

If a^d ≠ 1 mod n and a^2^r.d ≠ −1 mod n for all r ∈ [0, s − 1]

then return composite

Return prime

if n < 9,080,191, it is enough to test a = 31 and 73;

if n < 4,759,123,141, it is enough to test a = 2, 7, and 61;

if n < 2,152,302,898,747, it is enough to test a = 2, 3, 5, 7, and 11;

if n < 3,474,749,660,383, it is enough to test a = 2, 3, 5, 7, 11, and 13;

if n < 341,550,071,728,321, it is enough to test a = 2, 3, 5, 7, 11, 13, and 17.

**\section{Probability}**

Mathematical Expectation:

For a discrete variable X with probability function P(X), the expected value E[X] is given by ΣxiP(xi) the summation runs over all the distinct values xi that the variable can take.

The rule of "linearity of of the expectation" says that E[x1+x2] = E[x1] + E[x2].

It is important to understand that "expected value" is not same as "most probable value" - rather, it need not even be one of the probable values.

Example:

For a six-sided die, the expected number of throws to get each face at least once is (6/6)+(6/5)+(6/4)+(6/3)+(6/2)+(6/1) = 14.7.

Logic: The chance of rolling a number you haven't yet rolled when you start off is 1, as any number would work. Once you've rolled this number, your chance of rolling a number you haven't yet rolled is 5/6. Continuing in this manner, after you've rolled n different numbers the chance of rolling one you haven't yet rolled is (6−n)/6.

For an n-sided die the expected throws is (n/n) + (n/(n-1)) + (n/(n-2)) + … + n.

Other things to read:

Extended Euclid’s algo

Chinese remaindering

Farey’s sequence

Optimised Sieve, Sieve of Atkins

How to solve Diophantine Equation

Pollard Rho factorization

Stirling numbers

Inclusion-exclusion

Gaussean Elimination (Find the determinant of a matrix)

Group Theory