American International University-Bangladesh

AIUB Eclipse

MD Siyam Talukder
Kazi Shoaib Ahmed Saad
Faysal Ahammed Chowdhury

Contents

1 Setup

1.1 Sublime Build

```
"shell_cmd": "g++ -std=c++17 -o
    \"$file_base_name\" \"$file\" &&
    timeout 2.5s ./\"$file_base_name\" <
    input.txt > output.txt",
"file_regex":
    "^(..[^:]*):([0-9]+):?([0-9]+)?:?
    (.*)$",
"working_dir": "${file_path}",
"selector": "source.c, source.c++"
```

2 Stress Testing

2.1 Input Gen

```
mt19937_64 rnd(chrono::steady_clock::now
    ().time_since_epoch().count());
ll get_rand(ll l, ll r) {
  assert(l <= r);
  return l + rnd() % (r - l + 1);
}
```

2.2 Bash Script

```
// run -> bash script.sh

set -e
g++ code.cpp -o code
g++ gen.cpp -o gen
g++ brute.cpp -o brute
for((i = 1; ; ++i)); do
    ./gen $i > input_file
    ./code < input_file > myAnswer
    ./brute < input_file > correctAnswer
    diff -Z myAnswer correctAnswer >
        /dev/null || break
    echo "Passed test: "  $i
done
echo "WA on the following test:"
cat input_file
echo "Your answer is:"
cat myAnswer
echo "Correct answer is:"
cat correctAnswer
```

3 Number Theory

3.1 Euler Totient Function

```
// Time: O(√N)
map<int, int> dp; // memo
int phi(int n) {
  if (dp.count(n)) return dp[n];
  int ans = n, m = n;
  for (int i = 2; i * i <= m; i++) {
    if (m % i == 0) {
      while (m % i == 0) m /= i;
      ans = ans / i * (i - 1);
    }
  }
  if (m > 1) ans = ans / m * (m - 1);
  return dp[n] = ans;
}
```

3.2 Phi 1 to N

```
void phi_1_to_n(int n) {
  vector<int> phi(n + 1);
  for (int i = 0; i <= n; i++)
    phi[i] = i;
  for (int i = 2; i <= n; i++) {
    if (phi[i] == i) {
      for (int j = i; j <= n; j += i)
        phi[j] -= phi[j] / i;
    }
  }
}
```

3.3 Segmented Sieve

```
vector<char> segmentedSieve(ll L, ll R) {
  // generate all primes up to √R
  ll lim = sqrt(R);
  vector<char> mark(lim + 1, false);
  vector<ll> primes;
  for (ll i = 2; i <= lim; ++i) {
    if (!mark[i]) {
      primes.emplace_back(i);
      for (ll j = i * i; j <= lim; j +=
          i) mark[j] = true;
    }
  }
  vector<char> isPrime(R - L + 1, true);
  for (ll i : primes)
    for (ll j = max(i * i, (L + i - 1) /
        i * i); j <= R; j += i)
      isPrime[j - L] = false;
  if (L == 1) isPrime[0] = false;
  return isPrime;
}
```

3.4 Extended GCD

```
// ax + by = gcd(a, b)
int egcd(int a, int b, int& x, int& y) {
  if (b == 0) {
    x = 1, y = 0;
    return a;
  }
  int x1, y1;
  int d = egcd(b, a % b, x1, y1);
  x = y1;
  y = x1 - y1 * (a / b);
  return d;
}
```

3.5 Linear Diophantine Equation

```
// ax + by = c, find any x and y
bool find_any_solution(int a, int b, int
    c, int &x0, int &y0, int &g) {
  g = egcd(abs(a), abs(b), x0, y0);
  if (c % g) return false;
  x0 *= c / g;
  y0 *= c / g;
  if (a < 0) x0 = -x0;
  if (b < 0) y0 = -y0;
  return true;
}
void shift_solution(int & x, int & y,
    int a, int b, int cnt) {
  x += cnt * b;
  y -= cnt * a;
}
int find_all_solutions(int a, int b, int
    c, int minx, int maxx, int miny, int
    maxy) {
  int x, y, g;
  if (!find_any_solution(a, b, c, x, y,
      g)) return 0;
  a /= g, b /= g;
  int sign_a = a > 0 ? +1 : -1;
  int sign_b = b > 0 ? +1 : -1;
  shift_solution(x, y, a, b, (minx - x)
      / b);
  if (x < minx) shift_solution(x, y, a,
      b, sign_b);
  if (x > maxx) return 0;
  int lx1 = x;
  shift_solution(x, y, a, b, (maxx - x)
      / b);
  if (x > maxx) shift_solution(x, y, a,
      b, -sign_b);
  int rx1 = x;
  shift_solution(x, y, a, b, -(miny - y)
      / a);
  if (y < miny) shift_solution(x, y, a,
      b, -sign_a);
  if (y > maxy) return 0;
  int lx2 = x;
  shift_solution(x, y, a, b, -(maxy - y)
      / a);
  if (y > maxy) shift_solution(x, y, a,
      b, sign_a);
  int rx2 = x;
  if (lx2 > rx2) swap(lx2, rx2);
  int lx = max(lx1, lx2);
  int rx = min(rx1, rx2);
  if (lx > rx) return 0;
  return (rx - lx) / abs(b) + 1;
}
```

3.6 Modular Inverse using EGCD

```
// finding inverse(a) modulo m
int x, y;
int g = extended_euclidean(a, m, x, y);
if (g != 1) cout << "No solution!";
else {
  x = (x % m + m) % m;
  cout << x << endl;
}
```

3.7 Exclusion DP

```
ll f[N], g[N];
for (int i = N - 1; i >= 1; i--) {
  f[i] = nC4(div_cnt[i]);
  g[i] = f[i];
  for (int j = i + i; j < N; j += i) {
    g[i] -= g[j];
  }
}
```

Here, $f[i]$ = how many pairs/k-tuple such that their gcd is i or it's multiple (count of pairs those are divisible by i).

$g[i]$ = how many pairs/k-tuple such that their gcd is i.

$g[i] = f[i] - \sum_{i|j} g[j]$.

Sum of all pair gcd:

We know, how many pairs are there such that their gcd is i for every i (1 to n). So now, $\sum_{i=1}^{n} g[i] * i$.

Sum of all pair lcm (i = 1, j = 1):

We know, $\text{lcm}(a, b) = \frac{a*b}{\gcd(a,b)}$.

Now, $f[i]$ = All pair product sum of those, whose gcd is i or it's multiple.

$g[i]$ = All pair product sum of those, whose gcd is i.

Ans $= \sum_{i=1}^{n} \frac{g[i]}{i}$.

All pair product sum $= (a_1 + a_2 + \cdots + a_n) * (a_1 + a_2 + \cdots + a_n)$

3.8 Legendres Formula

```
// n!/p^x - you will get the largest x
int legendre(int n, int p) {
  int ex = 0;
  while(n) {
    ex += (n / p);
    n /= p;
  }
  return ex;
}
```

3.9 Binary Expo

```
int power(int x, long long n, int mod) {
  int ans = 1 % mod;
  while (n > 0) {
    if (n & 1) {
      ans = 1LL * ans * x % mod;
    }
    x = 1LL * x * x % mod;
    n >>= 1;
  }
  return ans;
}
```

3.10 Digit Sum of 1 to N

```cpp
// for n=10, ans = 1+2+...+9+1+0
ll solve(ll n) {
  ll res = 0, p = 1;
  while (n / p > 0) {
    ll left = n / (p * 10);
    ll cur  = (n / p) % 10;
    ll right = n % p;
    res += left * 45 * p;
    res += (cur * (cur - 1) / 2) * p;
    res += cur * (right + 1);
    p *= 10;
  } return res;
}
```

3.11 Pollard Rho

```cpp
namespace PollardRho {
mt19937 rnd(chrono::steady_clock::now().time_since_epoch().count());
const int P = 1e6 + 9;
ll seq[P];
int primes[P], spf[P];
inline ll add_mod(ll x, ll y, ll m) {
  return (x += y) < m ? x : x - m;
}
inline ll mul_mod(ll x, ll y, ll m) {
  ll res = __int128(x) * y % m;
  return res;
  // ll res = x * y - (ll)((long double)x * y / m + 0.5) * m;
  // return res < 0 ? res + m : res;
}
inline ll pow_mod(ll x, ll n, ll m) {
  ll res = 1 % m;
  for (; n; n >>= 1) {
    if (n & 1) res = mul_mod(res, x, m);
    x = mul_mod(x, x, m);
  }
  return res;
}
// O(it * (logn)^3), it = number of rounds performed
inline bool miller_rabin(ll n) {
  if (n <= 2 || (n & 1 ^ 1)) return (n == 2);
  if (n < P) return spf[n] == n;
  ll c, d, s = 0, r = n - 1;
  for (; !(r & 1); r >>= 1, s++) {}
  // each iteration is a round
  for (int i = 0; primes[i] < n && primes[i] < 32; i++) {
    c = pow_mod(primes[i], r, n);
    for (int j = 0; j < s; j++) {
      d = mul_mod(c, c, n);
      if (d == 1 && c != 1 && c != (n - 1)) return false;
      c = d;
    }
    if (c != 1) return false;
  }
  return true;
}
void init() {
  int cnt = 0;
  for (int i = 2; i < P; i++) {
    if (!spf[i]) primes[cnt++] = spf[i] = i;
    for (int j = 0, k; (k = i * primes[j]) < P; j++) {
      spf[k] = primes[j];
      if (spf[i] == spf[k]) break;
    }
  }
}
// returns O(n^(1/4))
ll pollard_rho(ll n) {
  while (1) {
    ll x = rnd() % n, y = x, c = rnd() % n, u = 1, v, t = 0;
    ll *px = seq, *py = seq;
    while (1) {
      *py++ = y = add_mod(mul_mod(y, y, n), c, n);
      *py++ = y = add_mod(mul_mod(y, y, n), c, n);
      if ((x = *px++) == y) break;
      v = u;
      u = mul_mod(u, abs(y - x), n);
      if (!u) return __gcd(v, n);
      if (++t == 32) {
        t = 0;
        if ((u = __gcd(u, n)) > 1 && u < n) return u;
      }
    }
    if (t && (u = __gcd(u, n)) > 1 && u < n) return u;
  }
}
vector<ll> factorize(ll n) {
  if (n == 1) return vector <ll>();
  if (miller_rabin(n)) return vector<ll> {n};
  vector <ll> v, w;
  while (n > 1 && n < P) {
    v.push_back(spf[n]);
    n /= spf[n];
  }
  if (n >= P) {
    ll x = pollard_rho(n);
    v = factorize(x);
    w = factorize(n / x);
    v.insert(v.end(), w.begin(), w.end());
  }
  return v;
}
}
```

3.12 [Problem] How Many Bases - UVa

```cpp
  // Given a number N^M, find out the number of integer bases in which it has exactly T trailing zeroes.
int solve_greater_or_equal(vector<int> e, int t) {
  int ans = 1;
  for (auto i : e) {
    ans = 1LL * ans * (i / t + 1) % mod;
  }
  return ans;
}
// e contains e1, e2 -> p1^e1, p2^e2
int solve_equal(vector<int> e, int t) {
  return (solve_greater_or_equal(e, t) - solve_greater_or_equal(e, t + 1) + mod) % mod;
}
```

3.13 [Problem] Power Tower - CF

```cpp
  // A sequence w1, w2, ..., wn and Q queries, l and r will be given.
  // Calculate w_l^(w_{l+1}^(...^(w_r)))
// n^x mod m = n^(phi(m)+x mod phi(m)) mod m
inline int MOD(int x, int m) {
  if (x < m) return x;
  return x % m + m;
}
int power(int n, int k, int mod) {
  int ans = MOD(1, mod);
  while (k) {
    if (k & 1) ans = MOD(ans * n, mod);
    n = MOD(n * n, mod);
    k >>= 1;
  }
  return ans;
}
int f(int l, int r, int m) {
  if (l == r) return MOD(a[l], m);
  if (m == 1) return 1;
  return power(a[l], f(l + 1, r, phi(m)), m);
}
```

3.14 Formula and Properties

- $\phi(n) = n \cdot \frac{p_1-1}{p_1} \cdot \frac{p_2-1}{p_2} \cdots$
- $\phi(p^e) = p^e - \frac{p^e}{p} = p^e \cdot \frac{p-1}{p}$
- For $n > 2$, $\phi(n)$ is always even.
- $\sum_{d|n} \phi(d) = n$
- NOD: $(e_1+1) \cdot (e_2+1) \cdots$
- SOD: $\frac{p_1^{e_1+1}-1}{p_1-1} \cdot \frac{p_2^{e_2+1}-1}{p_2-1} \cdots$
- $\log(a \cdot b) = \log(a) + \log(b)$
- $\log(a^x) = x \cdot \log(a)$
- $\log_a(x) = \frac{\log_b(x)}{\log_b(a)}$
- **Digit Count of n:** $\lfloor \log_{10}(n) \rfloor + 1$
- **Arithmetic Progression Sum:** $\frac{n}{2} \cdot (a + p)$, $\frac{n}{2} \cdot (2a + (n-1)d)$
- **Geometric Sum:** $S_n = a \cdot \frac{r^n - 1}{r - 1}$
- $(1^2 + 2^2 + \cdots + n^2) = \frac{n(n+1)(2n+1)}{6}$
- $(1^3 + 2^3 + \cdots + n^3) = \frac{n^2(n+1)^2}{4}$
- $(2^2 + 4^2 + \cdots + (2n)^2) = \frac{2n(n+1)(2n+1)}{3}$
- $(1^2 + 3^2 + \cdots + (2n-1)^2) = \frac{n(2n-1)(2n+1)}{3}$
- $(2^3 + 4^3 + \cdots + (2n)^3) = 2n^2(n+1)^2$
- $(1^3 + 3^3 + \cdots + (2n-1)^3) = n^2(2n^2-1)$
- For any number n and bases $> \sqrt{n}$, there will be no representation where the number contains 0 at its second least significant digit. So it is enough to check for bases $\leq \sqrt{n}$.
- For some x and y, let's try to find all m such that $x \bmod m \equiv y \bmod m$. We can rearrange the equation into $(x - y) \equiv 0 \pmod m$. Thus, if m is a factor of $|x - y|$, then x and y will be equal modulo m.