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

```
// 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

```
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

```
// 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 e_1, e_2 -> p_1^{e_1}, p_2^{e_2}
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

```
  // A sequence w_1, w_2, ..., w_n 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.

4 Combinatorics and Probability

4.1 Combinations

```
// Prime Mod in O(n)
void prec() {
  fact[0] = 1;
  for (int i = 1; i < N; i++) {
    fact[i] = 1ll * fact[i - 1] * i % mod;
  }
  ifact[N - 1] = inverse(fact[N - 1]);
  for (int i = N - 2; i >= 0; i--) {
    ifact[i] = 1ll * ifact[i + 1] * (i +
        1) % mod;
  }
}
int nCr(int n, int r) {
  if (r > n) return 0;
```

```
    return 1ll * fact[n] * ifact[r] % mod
        * ifact[n - r] % mod;
}
int nPr(int n, int r) {
    if (r > n) return 0;
    return 1ll * fact[n] * ifact[n - r] %
        mod;
}
```

4.2 nCr for any mod

```
// Time: O(n^2)
// nCr = (n-1)C(r-1) + (n-1)Cr
for (int i = 0; i < N; i++) {
  C[i][i] = 1;
  for (int j = 0; j < i; j++) {
    C[i][j] = (C[i - 1][j] + C[i - 1][j
        - 1]) % mod;
  }
}
```

4.3 nCk without mod in O(r)

```
ll nCk(ll n, ll k) {
  double res = 1;
  for (ll i = 1; i <= k; ++i)
    res = res * (n - k + i) / i;
  return (ll)(res + 0.01);
}
```

4.4 Lucas Theorem

```
// returns nCr modulo mod where mod is a
    prime
// Complexity: ?
ll Lucas(ll n, ll r) {
  if (r < 0 || r > n) return 0;
  if (r == 0 || r == n) return 1;
  if (n >= MOD) {
    return (Lucas(n / MOD, r / MOD) %
        MOD * Lucas(n % MOD, r % MOD) %
        MOD) % MOD;
  }
  return (((fact[n] * invFact[r]) % MOD)
      * invFact[n - r]) % MOD;
}
```

4.5 Catalan Number

```
const int MOD = 1e9 + 7, int MAX = 1e7;
int catalan[MAX];
void init(ll n) {
  catalan[0] = catalan[1] = 1;
  for ( ll i = 2; i <= n; i++ ) {
    catalan[i] = 0;
    for ( ll j = 0; j < i; j++ ) {
      catalan[i] += ( catalan[j] *
          catalan[i - j - 1] ) % MOD;
      if ( catalan[i] >= MOD ) {
        catalan[i] -= MOD;
      }
    }
  }
}
```

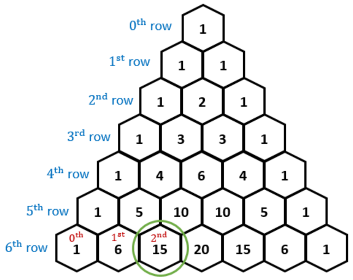
4.6 Derangement

```
  // number of combinations such that
    a_i != i of a permutation a
const int N = 1e6 + 100, int p = 1e9 + 7;
ll der[N];
void countDer() {
  der[1] = 0;
  der[2] = 1;
  for (ll i = 3; i <= N; ++i) {
    der[i] = (i - 1) % p * (der[i - 1] %
        p + der[i - 2] % p);
    der[i] %= p;
  }
}
```

4.7 Stars and Bars Theorem

- Find the number of k-tuples of non-negative integers whose sum is n. $\binom{n+k-1}{n}$
- Find the number of k-tuples of non-negative integers whose sum is $\le n$. $\binom{n+k}{k}$
- Combination with Repetition (choose k elements from n objects, same element can be chosen multiple times). $\binom{n+k-1}{k}$
- How many ways to go from $(0,0)$ to (n,m). $\binom{n+m}{m}$

Pascals Triangle is equivalent to nCr:



4.8 Properties of Pascal's Triangle

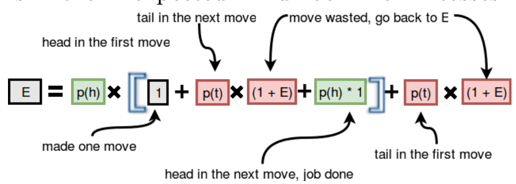
- $(a+b)^n = \sum_{k=0}^{n} \binom{n}{k} a^k b^{n-k}$
- $(k+1)^n = \sum_{i=0}^{n} k^i \cdot \binom{n}{i}$
- $\sum_{i=0}^{n} \binom{n}{i} = 2^n$
- $\binom{k}{n} = \frac{k}{n} \binom{k-1}{n-1}$
- $\sum_{k=0}^{m} \binom{n+k}{k} = \binom{n+m+1}{m}$
- $\binom{n}{0}^2 + \binom{n}{1}^2 + \cdots + \binom{n}{n}^2 = \binom{2n}{n}$
- $1\binom{n}{1} + 2\binom{n}{2} + \cdots + n\binom{n}{n} = n\,2^{n-1}$

4.9 Contribution Technique

- Sum of all pair sums: $\sum_{i=1}^{n}\sum_{j=1}^{n}(a_i + a_j)$ Every element will be added $2n$ times. $\sum_{i=1}^{n}(2 \times n \times a_i) = 2 \times n \times \sum_{i=1}^{n} a_i$.
- Sum of all subarray sums — $\sum_{i=1}^{n}(a_i \times i \times (n-i+1))$.
- Sum of all Subsets sums — $\sum_{i=1}^{n}(2^{n-1} \times a_i)$.
- Product of all pair product — $\prod_{i=1}^{n}(a_i^{2\times n})$.
- XOR of subarray XORS — How many subarrays does an element have? $(i \cdot (n-i+1)$ times). If subarray count is odd then this element can contribute in total XORs.
- Sum of max minus min over all subset — Sort the array. $Min = 2^{n-i}$, $Max = 2^{i-1}$. $\sum_{i=1}^{n}(a_i \cdot 2^{i-1} - a_i \cdot 2^{n-i})$
- Sum using bits — $\sum_{k=0}^{30}(cnt_k[1] \times 2^k)$.
- Sum of Pair XORs — XOR will 1 if two bits are different $\sum_{k=0}^{30}(cnt_k[0] \times cnt_k[1] \times 2^k)$.
- Sum of Pair ANDs — $\sum_{k=0}^{30}(cnt_k[1]^2 \times 2^k)$.
- Sum of Pair ORs — $\sum_{k=0}^{30}((cnt_k[1]^2 + 2 \times cnt_k[1] \times cnt_k[0]) \times 2^k)$.
- Sum of Subset XORs — where $cnt0! = 0$ $\sum_{k=0}^{30}\left(2^{cnt_k[1]+cnt_k[0]-1} \times 2^k\right)$.
- Sum of Subset ANDs — $\sum_{k=0}^{30}\left(\left(2^{cnt_k[1]} - 1\right) \times 2^k\right)$.
- Sum of Subset ORs — $\sum_{k=0}^{30}\left(\left(2^n - 2^{cnt_k[0]}\right) \times 2^k\right)$.
- Sum of subarray XORs — Convert to prefix xor, then solve for pairs.
- Sum of product of all subsequence — $\prod_{i=1}^{n}(a_i + 1) - 1$. Example array — $[a,b]$ the subsequences are $\{a\}, \{b\}, \{a,b\}$ so ans is $a + b + (a \cdot b)$

4.10 Probability and Expected Value

- Expected value: $E = \sum_{i=1}^{n} P_i \cdot i$
- Variance: $V(x) = E(x^2) - \{E(x)\}^2$
- To get two consecutive heads, what is the expected number of tosses?



- To get n heads, what is the expected number of tosses? Let's define: to get n heads, we need to toss $E(n)$ times. Now — I can get a head; I need to toss $E(n-1)$ more times, or if I get a tail; I need to toss $E(n)$ times. So, the recurrence is: $E(n) = 0.5 \cdot (1 + E(n-1)) + 0.5 \cdot (1 + E(n))$
- You have n bulbs, all of which are initially off. In each move, you randomly select one bulb. If the selected bulb is **off**, you toss a coin:
  - If you get head, you turn it on.
  - If you get tail, you do nothing.

  If the bulb is already **on**, you skip that move (nothing happens).

  Now, what is the expected number of moves required to turn all bulbs on?

  The coin is not fair — the probability of getting tail is p. This problem can also be solved recursively.

  Let's assume at some moment, x bulbs are already on, and the expected number of moves needed from here is $e(x)$.

  The probability of picking an already on bulb is $\frac{x}{n}$. In that case, the expected number of moves is $\frac{x}{n} \times (1 + e(x))$.

  The probability of picking an off bulb is $\frac{n-x}{n}$.

  Now two things can happen:
  - With probability p, you get tail, so you stay at the same state ($e(x)$ more moves).
  - With probability $(1-p)$, you get head, so one more bulb turns on ($e(x+1)$ moves from there).

  So, the recurrence relation is:

$$e(x) = \frac{x}{n}(1 + e(x)) + \frac{n-x}{n}\left(p(1 + e(x)) + (1-p)(1 + e(x+1))\right)$$

5 Data Structure

5.1 Trie

```
const int N = 10; // change here
const char base_char = '0';  // change
    here
struct TrieNode {
  int cnt;
  TrieNode * nxt[N];
  TrieNode() {
    cnt = 0;
    for (int i = 0; i < N; i++) nxt[i] =
        NULL;
  }
} *root;
void insert(const string &s) {
  TrieNode *cur = root;
  int n = (int)s.size();
  for (int i = 0; i < n; i++) {
    int idx = s[i] - base_char;
    if (cur -> nxt[idx] == NULL) cur ->
        nxt[idx] = new TrieNode();
    cur = cur -> nxt[idx];
    cur -> cnt++;
  }
}
```

```
void rem(TrieNode *cur, string &s, int
    pos) { // free :: De Alloactes Memory
  if (pos == s.size()) return;
  int idx = s[pos] - base_char;
  rem(cur -> nxt[idx], s, pos + 1);
  cur -> nxt[idx] -> cnt--;
  if (cur -> nxt[idx] -> cnt == 0) {
    free(cur -> nxt[idx]);
    cur -> nxt[idx] = NULL;
  }
}
int query(const string &s) { // "s" is a
    prefix of some element or not
  int n = (int)s.size();
  TrieNode *cur = root;
  for (int i = 0; i < n; i++) {
    int idx = s[i] - base_char;
    if (cur -> nxt[idx] == NULL) return 0;
    cur = cur -> nxt[idx];
  }
  return cur -> cnt;
}
void del(TrieNode *cur) {
  for (int i = 0; i < N; i++) if (cur ->
      nxt[i]) del(cur -> nxt[i]);
  delete(cur);
}
int32_t main() {
  root = new TrieNode(); // init new trie
  del(root); // clear trie
}
```

5.2 Trie for bit

```
struct Trie {
  static const int B = 31;
  struct node {
    node* nxt[2];
    int sz;
    node() {
      nxt[0] = nxt[1] = NULL;
      sz = 0;
    }
  }*root;
  Trie() {
    root = new node();
  }
  void insert(int val) {
    node* cur = root;
    cur -> sz++;
    for (int i = B - 1; i >= 0; i--) {
      int b = val >> i & 1;
      if (cur -> nxt[b] == NULL) cur ->
          nxt[b] = new node();
      cur = cur -> nxt[b];
      cur -> sz++;
    }
  }
  int query(int x, int k) { // number of
      values s.t. val ^ x < k
    node* cur = root;
    int ans = 0;
    for (int i = B - 1; i >= 0; i--) {
      if (cur == NULL) break;
      int b1 = x >> i & 1, b2 = k >> i &
          1;
      if (b2 == 1) {
        if (cur -> nxt[b1]) ans += cur
            -> nxt[b1] -> sz;
        cur = cur -> nxt[!b1];
      } else cur = cur -> nxt[b1];
    }
    return ans;
  }
  int get_max(int x) { // returns maximum
      of val ^ x
    node* cur = root;
    int ans = 0;
    for (int i = B - 1; i >= 0; i--) {
      int k = x >> i & 1;
      if (cur -> nxt[!k]) cur = cur ->
          nxt[!k], ans <<= 1, ans++;
      else cur = cur -> nxt[k], ans <<= 1;
    }
    return ans;
  }
  int get_min(int x) { // returns minimum
      of val ^ x
    node* cur = root;
    int ans = 0;
    for (int i = B - 1; i >= 0; i--) {
      int k = x >> i & 1;
      if (cur -> nxt[k]) cur = cur ->
          nxt[k], ans <<= 1;
      else cur = cur -> nxt[!k], ans <<=
          1, ans++;
    }
    return ans;
  }
  void del(node* cur) {
    for (int i = 0; i < 2; i++) if (cur
        -> nxt[i]) del(cur -> nxt[i]);
    delete(cur);
  }
} t;
```

6 String

6.1 Hashing

```
const int N = 1e6 + 9; // change here
const int MOD1 = 127657753, MOD2 =
    987654319;
const int p1 = 137, p2 = 277; // change
    here
int ip1, ip2;
pair<int, int> pw[N], ipw[N];
void prec() {
  pw[0] = {1, 1};
  for (int i = 1; i < N; i++) {
    pw[i].first = 1ll * pw[i - 1].first
        * p1 % MOD1;
    pw[i].second = 1ll * pw[i -
        1].second * p2 % MOD2;
  }
  ip1 = power(p1, MOD1 - 2, MOD1);
  ip2 = power(p2, MOD2 - 2, MOD2);
  ipw[0] = {1, 1};
  for (int i = 1; i < N; i++) {
    ipw[i].first = 1ll * ipw[i -
        1].first * ip1 % MOD1;
    ipw[i].second = 1ll * ipw[i -
        1].second * ip2 % MOD2;
  }
}
struct Hashing {
  int n;
  string s;
  vector<pair<int, int>> hash_val;
  vector<pair<int, int>> rev_hash_val;
  Hashing() {}
  Hashing(string _s) {
    s = _s;
    n = s.size();
    hash_val.emplace_back(0, 0);
    for (int i = 0; i < n; i++) {
      pair<int, int> p;
      p.first = (hash_val[i].first + 1ll
          * s[i] * pw[i].first % MOD1) %
          MOD1;
      p.second = (hash_val[i].second +
          1ll * s[i] * pw[i].second %
          MOD2) % MOD2;
      hash_val.push_back(p);
    }
    rev_hash_val.emplace_back(0, 0);
    for (int i = 0, j = n - 1; i < n;
        i++, j--) {
      pair<int, int> p;
      p.first = (rev_hash_val[i].first +
          1ll * s[i] * pw[j].first %
          MOD1) % MOD1;
      p.second = (rev_hash_val[i].second
          + 1ll * s[i] * pw[j].second %
          MOD2) % MOD2;
      rev_hash_val.push_back(p);
    }
  }
  pair<int, int> get_hash(int l, int r)
      { // 1 indexed
    pair<int, int> ans;
    ans.first = (hash_val[r].first -
        hash_val[l - 1].first + MOD1) *
        1ll * ipw[l - 1].first % MOD1;
    ans.second = (hash_val[r].second -
        hash_val[l - 1].second + MOD2) *
        1ll * ipw[l - 1].second % MOD2;
    return ans;
  }
  pair<int, int> rev_hash(int l, int r)
      { // 1 indexed
    pair<int, int> ans;
    ans.first = (rev_hash_val[r].first -
        rev_hash_val[l - 1].first +
        MOD1) * 1ll * ipw[n - r].first %
        MOD1;
    ans.second = (rev_hash_val[r].second
        - rev_hash_val[l - 1].second +
        MOD2) * 1ll * ipw[n - r].second
        % MOD2;
    return ans;
  }
  pair<int, int> get_hash() { // 1
      indexed
    return get_hash(1, n);
  }
  bool is_palindrome(int l, int r) {
    return get_hash(l, r) == rev_hash(l,
        r);
  }
};
```

6.2 Hashing with Updates

```
using T = array<int, 2>;
const T MOD = {127657753, 987654319};
const T p = {137, 277}; // change here
T operator + (T a, int x) {return {(a[0]
    + x) % MOD[0], (a[1] + x) % MOD[1]};}
T operator - (T a, int x) {return {(a[0]
    - x + MOD[0]) % MOD[0], (a[1] - x +
    MOD[1]) % MOD[1]};}
T operator * (T a, int x) {return
    {(int)((long long) a[0] * x %
    MOD[0]), (int)((long long) a[1] * x
    % MOD[1])};}
T operator + (T a, T x) {return {(a[0] +
    x[0]) % MOD[0], (a[1] + x[1]) %
    MOD[1]};}
T operator - (T a, T x) {return {(a[0] -
    x[0] + MOD[0]) % MOD[0], (a[1] -
    x[1] + MOD[1]) % MOD[1]};}
T operator * (T a, T x) {return
    {(int)((long long) a[0] * x[0] %
    MOD[0]), (int)((long long) a[1] *
    x[1] % MOD[1])};}
ostream& operator << (ostream& os, T
    hash) {return os << "(" << hash[0]
    << ", " << hash[1] << ")";}
T pw[N], ipw[N];
void prec() {
  pw[0] =  {1, 1};
  for (int i = 1; i < N; i++) {
    pw[i] = pw[i - 1] * p;
  }
  ipw[0] =  {1, 1};
  T ip = {power(p[0], MOD[0] - 2,
      MOD[0]), power(p[1], MOD[1] - 2,
      MOD[1])};
  for (int i = 1; i < N; i++) {
    ipw[i] = ipw[i - 1] * ip;
  }
}
struct Hashing {
  int n;
  string s; // 1 - indexed
  vector<array<T, 2>> t; // (normal, rev)
      hash
  array<T, 2> merge(array<T, 2> l,
      array<T, 2> r) {
    l[0] = l[0] + r[0];
    l[1] = l[1] + r[1];
    return l;
  }
  void build(int node, int b, int e) {
    if (b == e) {
      t[node][0] = pw[b] * s[b];
      t[node][1] = pw[n - b + 1] * s[b];
      return;
    }
```

```cpp
    int mid = (b + e) >> 1, l = node << 1, r = l | 1;
    build(l, b, mid);
    build(r, mid + 1, e);
    t[node] = merge(t[l], t[r]);
  }
  void upd(int node, int b, int e, int i, char x) {
    if (b > i || e < i) return;
    if (b == e && b == i) {
      t[node][0] = pw[b] * x;
      t[node][1] = pw[n - b + 1] * x;
      return;
    }
    int mid = (b + e) >> 1, l = node << 1, r = l | 1;
    upd(l, b, mid, i, x);
    upd(r, mid + 1, e, i, x);
    t[node] = merge(t[l], t[r]);
  }
  array<T, 2> query(int node, int b, int e, int i, int j) {
    if (b > j || e < i) return {T({0, 0}), T({0, 0})};
    if (b >= i && e <= j) return t[node];
    int mid = (b + e) >> 1, l = node << 1, r = l | 1;
    return merge(query(l, b, mid, i, j), query(r, mid + 1, e, i, j));
  }
  Hashing() {}
  Hashing(string _s) {
    n = _s.size();
    s = "." + _s;
    t.resize(4 * n + 1);
    build(1, 1, n);
  }
  void upd(int i, char c) {
    upd(1, 1, n, i, c);
    s[i] = c;
  }
  T get_hash(int l, int r) { // 1 - indexed
    return query(1, 1, n, l, r)[0] * ipw[l - 1];
  }
  T rev_hash(int l, int r) { // 1 - indexed
    return query(1, 1, n, l, r)[1] * ipw[n - r];
  }
  T get_hash() {
    return get_hash(1, n);
  }
  bool is_palindrome(int l, int r) {
    return get_hash(l, r) == rev_hash(l, r);
  }
};
```

6.3 Hashing with Upd and Deletes

```cpp
// update or delete a char in the string or check whether a range [l,r] is a palindrome or not (Palindromic Query I - Toph)
#define int long long
const int N = 1e5 + 9;
int en;
struct ST {
  pair<int, int> tree[4 * (N + N)];
  void build(int n, int b, int e) {
    if (b == e) {
      tree[n].first = b;
      tree[n].second = 1;
      return;
    }
    int mid = (b + e) >> 1, l = n << 1, r = l + 1;
    build(l, b, mid);
    build(r, mid + 1, e);
    tree[n].second = tree[l].second + tree[r].second;
  }
  void upd(int n, int b, int e, int i, int x1, int x2) {
    if (b > i || e < i) return;
    if (b == e && b == i) {
      tree[n].first = x1;
      tree[n].second = x2;
      return;
    }
    int mid = (b + e) >> 1, l = n << 1, r = l + 1;
    upd(l, b, mid, i, x1, x2);
    upd(r, mid + 1, e, i, x1, x2);
    tree[n].second = tree[l].second + tree[r].second;
  }
  pair<int, int> query(int n, int b, int e, int x) {
    if (b > e) return { -1, -1};
    if (tree[n].second < x) return {tree[n].second, -1};
    if (b == e) return tree[n];
    int mid = (b + e) >> 1, l = n << 1, r = l + 1;
    pair<int, int> L = query(l, b, mid, x);
    if (L.second != -1) return L;
    pair<int, int> R = query(r, mid + 1, e, x - L.first);
    return R;
  }
} st, st2;
using T = array<int, 2>;
const T MOD = {127657753, 987654319};
const T p = {137, 277};
// add operators overloading of T (from only upd) + prec()
int get(int i, int n) {
  return n - i + 1;
}
struct Hashing {
  int n; string s;
  vector<T> tree, lazy;
  void push(int node, int b, int e) {
    if (lazy[node][0] == 1) return;
    tree[node] = tree[node] * lazy[node];
    if (b != e) {
      int l = node << 1, r = l + 1;
      lazy[l] = lazy[l] * lazy[node];
      lazy[r] = lazy[r] * lazy[node];
    }
    lazy[node] = T{1, 1};
  }
  void build(int node, int b, int e) {
    lazy[node] = T{1, 1};
    if (b == e) {
      tree[node] = pw[b] * s[b];
      return;
    }
    int mid = (b + e) >> 1, l = node << 1, r = l | 1;
    build(l, b, mid);
    build(r, mid + 1, e);
    tree[node] = tree[l] + tree[r];
  }
  void upd(int node, int b, int e, int i, T x) {
    push(node, b, e);
    if (b > i || e < i) return;
    if (b == e && b == i) {
      tree[node] = x;
      return;
    }
    int mid = (b + e) >> 1, l = node << 1, r = l + 1;
    upd(l, b, mid, i, x);
    upd(r, mid + 1, e, i, x);
    tree[node] = tree[l] + tree[r];
  }
  void del(int node, int b, int e, int i, int j) {
    push(node, b, e);
    if (b > j || e < i) return;
    if (b >= i && e <= j) {
      lazy[node] = lazy[node] * ipw[1];
      push(node, b, e);
      return;
    }
    int mid = (b + e) >> 1, l = node << 1, r = l + 1;
    del(l, b, mid, i, j);
    del(r, mid + 1, e, i, j);
    tree[node] = tree[l] + tree[r];
  }
  T query(int node, int b, int e, int i, int j) {
    push(node, b, e);
    if (b > j || e < i) return {0, 0};
    if (b >= i && e <= j) return tree[node];
    int mid = (b + e) >> 1, l = node << 1, r = l + 1;
    T L = query(l, b, mid, i, j);
    T R = query(r, mid + 1, e, i, j);
    return L + R;
  }
  Hashing() {}
  Hashing(string _s) {
    s = _s;
    n = s.size();
    s = '.' + s;
    tree.resize(4 * n + 1);
    lazy.resize(4 * n + 1);
    build(1, 1, n);
  }
  void upd(int i, char c, int cur) {
    T x = pw[i] * c;
    if (cur == 1) i = st.query(1, 1, en, i).first;
    else i = st2.query(1, 1, en, i).first;
    upd(1, 1, n, i, x);
  }
  void del(int i, int cur) {
    int orgi = i;
    T x = pw[i] * 0ll;
    if (cur == 1) i = st.query(1, 1, en, i).first;
    else i = st2.query(1, 1, en, i).first;
    upd(1, 1, n, i, x);
    del(1, 1, n, i + 1, n);
    if (cur == 1) st.upd(1, 1, en, i, i, 0);
    else st2.upd(1, 1, en, i, i, 0);
  }
  T get_hash(int l, int r, int cur) { // 1 - indexed
    int ll = st.query(1, 1, en, l).first;
    int rr = st.query(1, 1, en, r).first;
    if (cur == 2) {
      ll = st2.query(1, 1, en, l).first;
      rr = st2.query(1, 1, en, r).first;
    }
    return query(1, 1, n, ll, rr) * ipw[l - 1];
  }
};
int32_t main() {
  prec(); // must include
  string s; cin >> s;
  int n = s.size();
  int q; cin >> q;
  string t = s;
  reverse(t.begin(), t.end());
  Hashing hs(s), hs2(t);
  en = n + q + 5;
  st.build(1, 1, en);
  st2.build(1, 1, en);
  while (q--) {
    char c; cin >> c;
    if (c == 'C') {
      int l, r; cin >> l >> r;
      int l2 = get(l, n);
      int r2 = get(r, n);
      if (hs.get_hash(l, r, 1) == hs2.get_hash(r2, l2, 2)) cout << "Yes!\n";
      else cout << "No!\n";
    }
    else if (c == 'U') {
      int i; char x; cin >> i >> x;
      int i2 = get(i, n);
      hs.upd(i, x, 1);
      hs2.upd(i2, x, 2);
    }
    else {
```

```
      int i; cin >> i;
      int i2 = get(i, n);
      hs.del(i, 1);
      hs2.del(i2, 2);
      --n;
    }
  }
}
```

6.4 Hashing on Tree

```
// Given a tree, Check whether it is
    symmetrical or not. Problem - CF G.
    Symmetree
// The value for each node is it's
    subtree size and position is the
    level (ordered). But the order of
    childs doesn't matter (unordered)
const int N = 2e5 + 9;
vector<int> g[N];
vector<array<int, 3>> hassh[N]; // hash1,
    hash2, node
int n, sz[N];
const int MOD1 = 1e9 + 9, MOD2 = 1e9 + 21;
const int p1 = 1e5 + 19, p2 = 1e5 + 43;
void dfs2(int u, int p, int lvl) {
  array<int, 3> my_hash;
  my_hash[0] = 1ll * sz[u] *
      pw[lvl].first % MOD1;
  my_hash[1] = 1ll * sz[u] *
      pw[lvl].second % MOD2;
  my_hash[2] = u;
  bool leaf = true;
  for (auto v : g[u]) {
    if (v != p) {
      dfs2(v, u, lvl + 1);
      leaf = false;
    }
  }
  if (!leaf) {
    int sum1 = 1, sum2 = 1;
    for (auto here : hassh[u]) {
      auto [x, y, _] = here;
      sum1 = (sum1 * x) % MOD1;
      sum2 = (sum2 * y) % MOD2;
    }
    my_hash[0] = power(my_hash[0], sum1,
        MOD1);
    my_hash[1] = power(my_hash[1], sum2,
        MOD2);
  }
  hassh[p].push_back(my_hash);
}
bool ok(int u) {
  map<pair<int, int>, int> mp;
  for (auto [x, y, who] : hassh[u]) {
    mp[{x, y}]++;
  }
  int odd = 0;
  pair<int, int> val;
  for (auto [here, cnt] : mp) {
    odd += cnt & 1;
    if (cnt & 1) val = here;
  }
  if (odd == 0) return true;
  if (odd > 1) return false;
  int node;
  for (auto [x, y, who] : hassh[u]) {
    pair<int, int> here = {x, y};
    if (here == val) node = who;
  }
  return ok(node);
}
void solve() {
  cin >> n; clr(n);
  for (int i = 2; i <= n; i++) {
    int u, v; cin >> u >> v;
    g[u].push_back(v);
    g[v].push_back(u);
  }
  dfs(1, 0); // calc. subtree size
  dfs2(1, 0, 1);
  if (ok(0)) cout << "YES\n";
  else cout << "NO\n";
}
```

6.5 Compare 2 strings Lexicographically

```
// Time: O(logn)
string s;
Hashing hs;
// return 0 if both equal
// return 1 if first substring greater
// return -1 if second substring greater
// here lcp() provides the len of longest
    common prefix
int compare(int i, int j, int x, int y) {
  int common_prefix = lcp(i, j, x, y);
  int len1 = j - i + 1, len2 = y - x + 1;
  if (common_prefix == len1 and len1 ==
      len2) return 0;
  else if (common_prefix == len1) return
      -1;
  else if (common_prefix == len2) return
      1;
  else return (s[i + common_prefix - 1]
      < s[x + common_prefix - 1] ? -1 :
      1);
}
```

6.6 KMP

```
vector<int> build_lps(string &pat) {
  int n = pat.size();
  vector<int> lps(n, 0);
  for (int i = 1; i < n; i++) {
    int j = lps[i - 1];
    while (j > 0 and pat[i] != pat[j]) {
      j = lps[j - 1];
    }
    if (pat[i] == pat[j]) j++;
    lps[i] = j;
  }
  return lps;
}
int kmp(string &txt, string &pat) {
  string s = pat + '#' + txt;
  vector<int> lps = build_lps(s);
  int ans = 0;
  for (auto x : lps) {
    if (x == pat.size()) ans++;
  }
  return ans;
}
int kmp(string &txt, string &pat) {
  vector<int> lps = build_lps(pat);
  int n = txt.size(), m = pat.size();
  int ans = 0;
  int j = 0;
  for (int i = 0; i < n; i++) {
    while (j > 0 and txt[i] != pat[j]) {
      j = lps[j - 1];
    }
    if (txt[i] == pat[j]) j++;
    if (j == m) {
      ans++;
      j = lps[j - 1];
    }
  }
  return ans;
}
```

6.7 KMP Automata

```
// like DFA. if string is "abcdeabg",
    aut[7]['c'] = 3. Means 7th index e
    'c' bosaile LPS koto, aut[7]['g'] = 8
void compute_automaton(string s,
    vector<vector<int>>& aut) {
  s += '#';
  int n = s.size();
  vector<int> pi = build_lps(s);
  aut.assign(n, vector<int>(26));
  for (int i = 0; i < n; i++) {
    for (int c = 0; c < 26; c++) {
      if (i > 0 && 'a' + c != s[i])
        aut[i][c] = aut[pi[i - 1]][c];
      else
        aut[i][c] = i + ('a' + c == s[i]);
    }
  }
}
```

6.8 Prefix Occurance Count

```
// Count the number of occurances of each
    prefix
vector<int> ans(n + 1);
for (int i = 0; i < n; i++) ans[lps[i]]++;
for (int i = n - 1; i > 0; i--)
    ans[lps[i - 1]] += ans[i];
for (int i = 0; i <= n; i++) ans[i]++;
```

6.9 Number of palindormic substring in L to R using Wavelet Tree

```
// Problem - Kattis palindromes
ll f(int x) {
  return (1ll * x * (x + 1)) / 2;
}
ll f(int l, int r) {
  if (l > r) return 0;
  return f(r) - f(l - 1);
}
bool ok(int l, int r) {
  return hash_s.is_palindrome(l, r);
}
int32_t main() {
  cin >> s;
  n = s.size();
  hash_s = Hashing(s);
  for (int i = 1; i <= n; i++) {
    int l = 0, r = min(n - i, i - 1),
        cnt = 1;
    while (l <= r) {
      int mid = (l + r) >> 1;
      if (ok(i - mid, i + mid)) {
        cnt = mid;
        l = mid + 1;
      }
      else r = mid - 1;
    }
    pi1[i] = cnt + 1;
    pi1_left[i] = pi1[i] - i;
    pi1_right[i] = i + pi1[i];
  }
  for (int i = 2; i <= n; i++) {
    if (s[i - 1] == s[i - 2]) {
      int l = 0, r = min(n - i, i - 1),
          cnt = 2;
      while (l <= r) {
        int mid = (l + r) >> 1;
        if (ok(i - 1 - mid, i + mid)) {
          cnt = mid;
          l = mid + 1;
        }
        else r = mid - 1;
      }
      pi2[i] = cnt + 1;
    }
    else pi2[i] = 0;

    pi2_left[i] = pi2[i] - i;
    pi2_right[i] = i + pi2[i];
  }
  // wavelet trees (odd_len_left,
      odd_len_right, even_len_left,
      even_len_right)
  t1.init(pi1_left + 1, pi1_left + n +
      1, -N, N);
  t2.init(pi1_right + 1, pi1_right + n +
      1, -N, N);
  t3.init(pi2_left + 1, pi2_left + n +
      1, -N, N);
  t4.init(pi2_right + 1, pi2_right + n +
      1, -N, N);

  int q; cin >> q;
  while (q--) {
    int l, r; cin >> l >> r;
    // define k, find cnt > k and
        summation whose are <= k;
    int m = (l + r) / 2;
    int k = 1 - l;
    ll ans = f(l, m);
    ans += t1.sum(l, m, k);
    int cnt = t1.GT(l, m, k);
    ans += 1ll * k * cnt;
    k = 1 + r;
    ans += -f(m + 1, r);
    ans += t2.sum(m + 1, r, k);
    cnt = t2.GT(m + 1, r, k);
    ans += 1ll * k * cnt;
```

```
    if (l + 1 <= m) { // a bit different
        than others
      k = -l;
      ans += f(l + 1, m);
      ans += t3.sum(l + 1, m, k);
      cnt = t3.GT(l + 1, m, k);
      ans += 1ll * k * cnt;
    }
    k = 1 + r;
    ans += -f(m + 1, r);
    ans += t4.sum(m + 1, r, k);
    cnt = t4.GT(m + 1, r, k);
    ans += 1ll * k * cnt;
    cout << ans << '\n';
  }
}
```

It is easier to explain by considering only palindromes centered at indicies (so, odd length), the idea is the same anyway. For each index i, r_i will be the longest radius of a palindrome centered there (in other words, the amount of palindromes centered at index i). Directly from manacher, this takes $\mathcal{O}(n)$ to calculate.

For a query $[l, r]$, we first compute $m = \frac{l+r}{2}$. Now we want to calculate

$$\sum_{i=l}^{m} \min(i - l + 1, r_i) + \sum_{i=m+1}^{r} \min(r - i + 1, r_i)$$

$$\sum_{i=l}^{m} \min(i - l + 1, r_i) = \sum_{i=l}^{m} i + \min(1 - l, r_i - i).$$

The sum over i can be found in constant time. As for the other term, if we create some array $r'_i = r_i - i$ during the preprocessing, then the queries are asking for some over range of $\min(C, r'_i)$ where C is constant. You can solve this in $\mathcal{O}(\log n)$ per query using wavelet tree.