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++"
```

1.2 Template

```
#include<bits/stdc++.h>
#include<ext/pb_ds/assoc_container.hpp>
#include<ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
using namespace std;
template <typename T> using o_set =
    tree<T, null_type, less<T>,
    rb_tree_tag,
    tree_order_statistics_node_update>;
typedef long long ll;
int32_t main() {
  ios_base::sync_with_stdio(0);
  cin.tie(0);
}
```

1.3 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);
}
```

1.4 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
```

2 Number Theory

2.1 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;
}
```

2.2 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;
    }
  }
}
```

2.3 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;
}
```

2.4 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;
}
```

2.5 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;
}
```

2.6 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;
}
```

2.7 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** $(1 <= i, j <= n)$**:** We know, $\text{lcm}(a,b) = \frac{a*b}{\text{gcd}(a,b)}$.
- 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 $= (a_1 + a_2 + \cdots + a_n) * (a_1 + a_2 + \cdots + a_n)$

2.8 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;
}
```

2.9 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;
}
```

2.10 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;
}
```

2.11 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;
}
}
```

2.12 [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 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;
}
```

2.13 [Problem] Power Tower - CF

```
    // 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);
}
```

2.14 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.

3 Combinatorics and Probability

3.1 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;
}
```

3.2 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;
  }
}
```

3.3 nCr 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);
}
```

3.4 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;
}
```

3.5 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;
      }
    }
  }
}
```

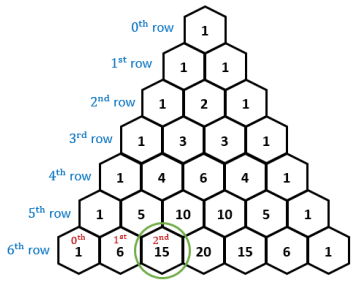
3.6 Derangement

```
    // number of combinations such that
    a_i != i of a permuation 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;
  }
}
```

3.7 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 $\leq n$. $\binom{n+k}{k}$
- 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:



3.8 Properties of Pascal's Triangle

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

3.9 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 — $\prod_{i=1}^{n} \left(a_i^{2\times n}\right)$.
- 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 $cnt0! = 0$ $\sum_{k=0}^{30} (2^{cnt_k[1]+cnt_k[0]-1} \times 2^k)$.
- Sum of Subset ANDs — $\sum_{k=0}^{30} ((2^{cnt_k[1]} - 1) \times 2^k)$.
- Sum of Subset ORs — $\sum_{k=0}^{30} ((2^n - 2^{cnt_k[0]}) \times 2^k)$.
- 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 $a + b + (a \cdot b)$

3.10 Probability and Expected Value

- Expected Value: $E = \frac{\text{Sum of all possible values}}{\text{Total number of outcomes}}$
- Expected Value: $E = \sum_{i=1}^{n} P_i \cdot i$
- Variance: $V(x) = E(x^2) - \{E(x)\}^2$
- Expected Value with DP: $E(i) = \sum P(i \to j) \times (R(i \to j) + E(j))$ Where $R()$ is Immediate Reward(cost/count)
- Linearity of Expectation:
  - $E[X + Y] = E[X] + E[Y]$
  - $E[\text{Total}] = E[I_1] + E[I_2] + \cdots = \sum P(I_i = 1)$

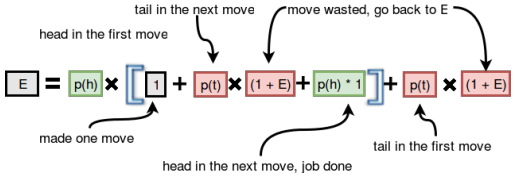
  We need to define the Indicator Random Variable (I) correctly. Some Examples below —

  Problem-1: Find E[correct hats] in a random permutation. Indicator I_i: "Does person i get their own hat?"

  Problem-2: Find E[total inversions] in a random permutation. Indicator I_{ij}: "Is pair (i, j) an inversion?"

  Problem-3: Given a string S, delete a random index until it becomes empty. Find the expected count of palindromes seen (exclude S, include empty string). Indicator I_L: "Is the string of length L a palindrome?"

  Problem-4: Pick N random integers from $[1, k]$. Merge consecutive equal values (e.g., $1, 1, 2, 2, 1, 1 \to 1, 2, 1$). Find the expected final length. Indicator I_i: "Is item i different from item $i - 1$?"
- 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 $\frac{\frac{x}{n}}{\frac{x}{n}}$. In that case, the expected number of moves is $\frac{\frac{x}{n}}{\frac{x}{n}} \times (1 + e(x))$.

  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:

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

4 Data Structure

4.1 Next Greater using Stack

```
stack<pair<int, int>> st;
   int right[n + 1];     // a[i] is maximum
    from index i to right[i]
for (int i = 1; i <= n; i++) {
  while (!st.empty() and st.top().first
      < a[i]) {
    right[st.top().second] = i - 1;
    st.pop();
  }
  st.push({a[i], i});
}
while (!st.empty()) {
  right[st.top().second] = n;
  st.pop();
}
```

4.2 Segment Tree Lazy

```
struct ST {
  int tree[4 * N], lazy[4 * N];
  void push(int n, int b, int e) {
    if(lazy[n] == 0) return;
    tree[n] += lazy[n] * (e - b + 1); //
        change here
    if(b != e) {
      int l = n << 1, r = l + 1;
      lazy[l] += lazy[n]; // change here
      lazy[r] += lazy[n]; // change here
    }
    lazy[n] = 0;
  }
  void build(int n, int  b, int e) {
    lazy[n] = 0;
    if(b == e) {
      tree[n] = a[b]; // change here
      return;
    }
    int mid = (b + e) >> 1, l = n << 1,
        r = l + 1;
    build(l, b, mid);
    build(r, mid + 1, e);
    tree[n] = tree[l] + tree[r]; //
        change here
  }
  void upd(int n, int b, int e, int i,
      int j, int x) {
    push(n, b, e);
    if(b > j || e < i) return;
    if(b >= i && e <= j) {
      lazy[n] += x; // change here
      push(n, b, e);
      return;
    }
    int mid = (b + e) >> 1, l = n << 1,
        r = l + 1;
    upd(l, b, mid, i, j, x);
    upd(r, mid + 1, e, i, j, x);
    tree[n] = tree[l] + tree[r]; //
        change here
  }
  int query(int n, int b, int e, int i,
      int j) {
    push(n, b, e);
    if(b > j || e < i) return 0; //
        return appropriate value
    if(b >= i && e <= j) return tree[n];
    int mid = (b + e) >> 1, l = n << 1,
        r = l + 1;
    int L = query(l, b, mid, i, j);
    int R = query(r, mid + 1, e, i, j);
    return L + R; // change here
  }
} st;
```

4.3 First Element Greater than K in a Range

```
// tree[n] = max in range
int get_first(int n, int b, int e, int
    i, int j, int x) { // return index
  push(n, b, e);
  if (b > j || e < i) return -1;
  if (tree[n] <= x) return -1;
  if (b == e) return b;
  int mid = (b + e) >> 1, l = n << 1, r
      = l + 1;
  int left = get_first(l, b, mid, i, j,
      x);
  if (left != -1) return left;
  return get_first(r, mid + 1, e, i , j,
      x);
}
```

4.4 Number of Unique Element in a Range

```
vector<pair<int, int>> Q[QQ];
int main() {
  int n; cin >> n;
  for(int i = 1; i <= n; i++) cin >> a[i];
  int q; cin >> q;
  for(int i = 1; i <= q; i++) {
    int l, r; cin >> l >> r;
    Q[r].push_back({l, i});
  }
  st.build(1, 1, n); // range sum and
      point add
  map<int, int> last_idx;
  for(int r = 1; r <= n; r++) {
    if(last_idx.find(a[r]) !=
        last_idx.end()) {
      st.upd(1, 1, n, last_idx[a[r]], -1);
    }
    last_idx[a[r]] = r;
    st.upd(1, 1, n, r, 1);
    for(auto i: Q[r]) {
      int l = i.first, id = i.second;
      ans[id] = st.query(1, 1, n, l, r);
    }
  }
}
```

4.5 Max Subarray Sum in a Range

```
struct node {
  ll tot_sum;
  ll prefix_max, suffix_max;
  ll max_subarray_sum;
};
node merge(node l, node r) {
  if (l.tot_sum == -inf) return r;
  if (r.tot_sum == -inf) return l;
  node ans;
  ans.max_subarray_sum =
      max(l.max_subarray_sum,
      r.max_subarray_sum);
  ans.max_subarray_sum =
      max(ans.max_subarray_sum,
      l.suffix_max + r.prefix_max);
  ans.tot_sum = l.tot_sum + r.tot_sum;
  ans.prefix_max = l.prefix_max;
  if (l.tot_sum + r.prefix_max >=
      l.prefix_max) {
    ans.prefix_max = l.tot_sum +
        r.prefix_max;
  }
  ans.suffix_max = r.suffix_max;
  if (r.tot_sum + l.suffix_max >=
      r.suffix_max) {
    ans.suffix_max = r.tot_sum +
        l.suffix_max;
  }
  return ans;
}
```

4.6 Count of subarrays such that their XOR is 0 in a range with upd

```
struct node {
  int cnt[64];
  node() {
    memset(cnt, 0, sizeof cnt);
  }
};
struct ST {
  node tree[4 * N]; int lazy[4 * N];
  void push(int n, int b, int e) {
    if (lazy[n] == 0) return;
    for (int k = 0; k < 64; k++) {
      if ((k ^ lazy[n]) > k) {
        swap(tree[n].cnt[k],
            tree[n].cnt[k ^ lazy[n]]);
      }
    }
    if (b != e) {
      int l = n << 1, r = l + 1;
      lazy[l] ^= lazy[n];
      lazy[r] ^= lazy[n];
    }
```

```
    lazy[n] = 0;
  }
  node merge(node a, node b) {
    for (int k = 0; k < 64; k++)
        a.cnt[k] += b.cnt[k];
    return a;
  }
  void build(int n, int  b, int e) {
    lazy[n] = 0;
    if (b == e) {
      tree[n].cnt[0] = 1;
      return;
    }
    int mid = (b + e) >> 1, l = n << 1,
        r = l + 1;
    build(l, b, mid);
    build(r, mid + 1, e);
    tree[n] = merge(tree[l], tree[r]);
  }
  void upd(int n, int b, int e, int i,
      int j, int x) {
    push(n, b, e);
    if (b > j || e < i) return;
    if (b >= i && e <= j) {
      lazy[n] ^= x; push(n, b, e);
      return;
    }
    int mid = (b + e) >> 1, l = n << 1,
        r = l + 1;
    upd(l, b, mid, i, j, x);
    upd(r, mid + 1, e, i, j, x);
    tree[n] = merge(tree[l], tree[r]);
  }
  node query(int n, int b, int e, int i,
      int j) {
    push(n, b, e);
    if (b > j || e < i) {
      node tmp;return tmp;
    }
    if (b >= i && e <= j) return tree[n];
    int mid = (b + e) >> 1, l = n << 1,
        r = l + 1;
    node L = query(l, b, mid, i, j);
    node R = query(r, mid + 1, e, i, j);
    return merge(L, R);
  }
} st;
int32_t main() {
  string s; cin >> s;
  int n = s.size();
  s = '.' + s;
  st.build(1, 1, n);
  for (int i = 1; i <= n; i++) {
    int x = (1 << (s[i] - 'a'));
    st.upd(1, 1, n, i, n, x); // segtree
        contains prefix xor
  }
  int q; cin >> q;
  while (q--) {
    int type; cin >> type;
    if (type == 1) {
      int l, r; cin >> l >> r;
      node tmp = st.query(1, 1, n, l -
          1, r);
      ll ans = 0;
      for (int k = 0; k < 64; k++) {
        int x = tmp.cnt[k];
        if (l == 1 and k == 0) x++;
        ans += (1ll * x * (x - 1)) / 2;
      }
      cout << ans << '\n';
    }
    else {
      int i; cin >> i;
      char c; cin >> c;
      int mask = (1 << (s[i] - 'a'));
      mask ^= (1 << (c - 'a'));
      st.upd(1, 1, n, i, n, mask);
      s[i] = c;
    }
  }
}
```

4.7 Range Add, Set and Sum

```
// Problem: Range Updates and Sums - CSES
struct ST {
  int tree[4 * N];
  pair<int, int> lazy[4 * N]; // val,
      types
  // lazy[n].first = 0; default for all n
  // lazy[n].second = -1; default for all
      n
  void push(int n, int b, int e) {
    if (lazy[n].first == 0) return;
    if (lazy[n].second == 1) { // set
      tree[n] = lazy[n].first * (e - b +
          1);
    }
    else { // add
      tree[n] += lazy[n].first * (e - b
          + 1);
    }
    if (b != e) {
      int l = n << 1, r = l + 1;
      if (lazy[l].second == 1) {
        if (lazy[n].second == 1)
            lazy[l].first =
            lazy[n].first;
        else lazy[l].first +=
            lazy[n].first;
      }
      else {
        if (lazy[n].second == 1) {
          lazy[l].first = lazy[n].first;
          lazy[l].second = 1;
        }
        else lazy[l].first +=
            lazy[n].first;
      }
      if (lazy[r].second == 1) {
        if (lazy[n].second == 1)
            lazy[r].first =
            lazy[n].first;
        else lazy[r].first +=
            lazy[n].first;
      }
      else {
        if (lazy[n].second == 1) {
          lazy[r].first = lazy[n].first;
          lazy[r].second = 1;
        }
        else lazy[r].first +=
            lazy[n].first;
      }
    }
    lazy[n].first = 0; lazy[n].second =
        -1;
  }
} st;
```

4.8 a[i] = a[i] * b + c in a Range

```
// Problem: Range Affine Range Sum
// lazy1[n] = 1; lazy2[n] = 0; // init in
    build
// lazy1[n] *= x; lazy2[n] += y; // when
    upd
mint tree[4 * N], lazy1[4 * N], lazy2[4
    * N];
void push(int n, int b, int e) {
  if (lazy1[n] == 1 or lazy2[n] == 0)
      return;
  tree[n] = tree[n] * lazy1[n] +
      lazy2[n] * (e - b + 1);
  if (b != e) {
    int mid = (b + e) >> 1, l = n << 1,
        r = l + 1;
    lazy1[l] *= lazy1[n];
    lazy2[l] = lazy2[l] * lazy1[n] +
        lazy2[n];
    lazy1[r] *= lazy1[n];
    lazy2[r] = lazy2[r] * lazy1[n] +
        lazy2[n];
  }
  lazy1[n] = 1; lazy2[n] = 0;
}
```

4.9 [Trick] Historic Information - GSS2

```
// Statement: Max Subarray Sum in a Range
    but only Unique Value can contrib.
int ans[QQ];
vector<pair<int, int>> Q[QQ];
struct ST {
  int tree[4 * N], lazy[4 * N];
  int tree2[4 * N], lazy2[4 * N];
  void push(int n, int b, int e) {
    tree2[n] = max(tree2[n], tree[n] +
        lazy2[n]);
    tree[n] += lazy[n];
    if (b != e) {
      int l = n << 1, r = l + 1;
      lazy2[l] = max(lazy2[l], lazy[l] +
          lazy2[n]);
      lazy2[r] = max(lazy2[r], lazy[r] +
          lazy2[n]);
      lazy[l] += lazy[n];
      lazy[r] += lazy[n];
    }
    lazy[n] = 0; lazy2[n] = 0;
  }
  void upd(int n, int b, int e, int i,
      int j, int x) {
    push(n, b, e);
    if (b > j || e < i) return;
    if (b >= i && e <= j) {
      lazy[n] += x;
      lazy2[n] = max(lazy2[n], lazy[n]);
      push(n, b, e);
      return;
    }
    int mid = (b + e) >> 1, l = n << 1,
        r = l + 1;
    upd(l, b, mid, i, j, x);
    upd(r, mid + 1, e, i, j, x);
    tree[n] = max(tree[l], tree[r]);
    tree2[n] = max(tree2[l], tree2[r]);
  }
  int query(int n, int b, int e, int i,
      int j) {
    push(n, b, e);
    if (b > j || e < i) return 0;
    if (b >= i && e <= j) return tree2[n];
    int mid = (b + e) >> 1, l = n << 1,
        r = l + 1;
    int L = query(l, b, mid, i, j);
    int R = query(r, mid + 1, e, i, j);
    return max(L, R);
  }
} st;
int32_t main() {
  int n; cin >> n; int a[n + 1];
  for (int i = 1; i <= n; i++) cin >>
      a[i];
  int q; cin >> q;
  for (int i = 1; i <= q; i++) {
    int l, r; cin >> l >> r;
    Q[r].push_back({l, i});
  }
  map<int, int> last;
  for (int r = 1; r <= n; r++) {
    // for (int i = last[a[r]] + 1; i <=
        r; i++) {
    //   sum[i] += a[r];
    //   p[i] = max(p[i], sum[i]);
    // }
    st.upd(1, 1, n, last[a[r]] + 1, r,
        a[r]);
    last[a[r]] = r;
    for (auto i : Q[r]) {
      int l = i.first, id = i.second;
      ans[id] = st.query(1, 1, n, l, r);
    }
  }
}
```

4.10 [Problem] Strongest Community - LOJ

```
    // Given an Array, if any a_i = x
    exist all x are consecutive. Given
    [L, R] Q times, Print Max Freq.
struct node {
  int first_element, first_element_cnt;
  int last_element, last_element_cnt;
  int max_cnt;
};
node merge(node l, node r) {
  if(l.first_element == -1) return r;
  if(r.first_element == -1) return l;
```

```
  node ans;
  ans.max_cnt = max(l.max_cnt, r.max_cnt);
  if(l.last_element == r.first_element) {
    ans.max_cnt = max(ans.max_cnt,
        l.last_element_cnt +
        r.first_element_cnt);
  }
  ans.first_element = l.first_element;
  ans.first_element_cnt =
      l.first_element_cnt;
  if(l.first_element == r.first_element) {
    ans.first_element_cnt +=
        r.first_element_cnt;
  }
  ans.last_element = r.last_element;
  ans.last_element_cnt =
      r.last_element_cnt;
  if(r.last_element == l.last_element) {
    ans.last_element_cnt +=
        l.last_element_cnt;
  }
  return ans;
}
```

4.11 [Problem] Diablo - LOJ

```
// Add element at the End, and Remove Kth
    Index (also print it's value)
int deleted_idx;
struct ST {
  pair<int, int> tree[4 * (N + Q)]; //
      sum, alive_cnt
  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) {
      deleted_idx = b;
      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;
void solve() {
  int id = n, end = n + q + 5;
  while (q--) {
    char c; cin >> c;
    if (c == 'a') {
      int p; cin >> p;
      st.upd(1, 1, end, ++id, p, 1);
    }
    else {
      int k; cin >> k;
      pair<int, int> ans = st.query(1,
          1, end, k);
      if (ans.second == -1) cout <<
          "none\n";
      else {
        cout << ans.first << '\n';
        st.upd(1, 1, end, deleted_idx,
            0, 0);
      }
    }
  }
}
```

4.12 Merge Sort Tree with Point Upd

```
struct MST { // merge sort tree
  o_set<array<int, 2>> tree[4 * N]; //
      greater<T>
  o_set<array<int, 2>>
      merge(o_set<array<int, 2>> &a,
      o_set<array<int, 2>> &b) {
    int i = 0, j = 0;
    int n = a.size(), m = b.size();
    o_set<array<int, 2>> ans;
    for (auto x : a) ans.insert(x);
    for (auto x : b) ans.insert(x);
    return ans;
  }
  void build(int n, int b, int e) {
    if (b == e) {
      tree[n].insert({a[b], b});
      return;
    }
    int mid = (b + e) >> 1, l = n << 1,
        r = l + 1;
    build(l, b, mid);
    build(r, mid + 1, e);
    tree[n] = merge(tree[l], tree[r]);
  }
  void upd(int n, int b, int e, int i,
      int x) {
    if (b == e) {
      tree[n].erase({a[b], b});
      tree[n].insert({x, b});
      a[b] = x;
      return;
    }
    int mid = (b + e) >> 1, l = n << 1,
        r = l + 1;
    tree[n].erase({a[i], i});
    tree[n].insert({x, i});
    if (i <= mid) upd(l, b, mid, i, x);
    else upd(r, mid + 1, e, i, x);
  }
  int query(int n, int b, int e, int i,
      int j, int k) {
    if (b > j || e < i) return 0;
    if (b >= i && e <= j) {
      int idx = tree[n].order_of_key({k,
          inf});
      return idx;
    }
    int mid = (b + e) >> 1, l = n << 1,
        r = l + 1;
    int L = query(l, b, mid, i, j, k);
    int R = query(r, mid + 1, e, i, j, k);
    return (L + R);
  }
} mst;
```

4.13 Sparse Table

```
const int N = 2e5 + 9, K = 20; // change
    here
int a[N], tree[N][K];
int log2_floor(unsigned long long i) {
  return i ? __builtin_clzll(1) -
      __builtin_clzll(i) : -1;
}
void build(int n) {
  for (int i = 1; i <= n; i++) {
    tree[i][0] = a[i];
  }
  for (int k = 1; k < K; k++) {
    for (int i = 1; i + (1 << k) - 1 <=
        n; i++) {
      tree[i][k] = min(tree[i][k - 1],
          tree[i + (1 << (k - 1))][k -
          1]); // change here
    }
  }
}
int query(int l, int r) {
  int k = log2_floor(r - l + 1);
  return min(tree[l][k], tree[r - (1 <<
      k) + 1][k]); // change here
}
build(n);
```

4.14 DSU

```
struct DSU {
  vector<int> par, sz;
  int c;
  DSU(int n) {
    par.resize(n + 1), sz.resize(n + 1,
        1);
    for (int i = 1; i <= n; i++) {
      par[i] = i;
    }
    c = n;
  }
  int find(int i) {
    return (i == par[i] ? i : par[i] =
        find(par[i]));
  }
  bool same(int i, int j) {
    return find(i) == find(j);
  }
  int get_size(int i) {
    return sz[find(i)];
  }
  int count() { // number of connected
      components
    return c;
  }
  int merge(int i, int j) {
    if ((i = find(i)) == (j = find(j)))
        return -1;
    c--;
    if (sz[i] < sz[j]) swap(i, j);
    par[j] = i;
    sz[i] += sz[j];
    return i;
  }
};
DSU dsu(n);
```

4.15 MOs Algorithm

```
const int N = 1e6 + 9, B = 440;
struct query {
  int l, r, id;
  bool operator < (const query &x) const {
    if (l / B == x.l / B) return ((l /
        B) & 1) ? r > x.r : r < x.r;
    return l / B < x.l / B;
  }
} Q[N];
int cnt[N], a[N];
long long sum;
inline void add_left(int i) {
  int x = a[i];
  sum += 1LL * (cnt[x] + cnt[x] + 1) * x;
  ++cnt[x];
}
inline void add_right(int i) {
  int x = a[i];
  sum += 1LL * (cnt[x] + cnt[x] + 1) * x;
  ++cnt[x];
}
inline void rem_left(int i) {
  int x = a[i];
  sum -= 1LL * (cnt[x] + cnt[x] - 1) * x;
  --cnt[x];
}
inline void rem_right(int i) {
  int x = a[i];
  sum -= 1LL * (cnt[x] + cnt[x] - 1) * x;
  --cnt[x];
}
long long ans[N];
int32_t main() {
  int n, q; cin >> n >> q;
  for (int i = 1; i <= n; i++) cin >>
      a[i];
  for (int i = 1; i <= q; i++) {
    cin >> Q[i].l >> Q[i].r;
    Q[i].id = i;
  }
  sort(Q + 1, Q + q + 1);
  int l = 1, r = 0;
  for (int i = 1; i <= q; i++) {
    int L = Q[i].l, R = Q[i].r;
    if (R < l) {
      while (l > L) add_left(--l);
      while (l < L) rem_left(l++);
      while (r < R) add_right(++r);
      while (r > R) rem_right(r--);
    } else {
      while (r < R) add_right(++r);
      while (r > R) rem_right(r--);
      while (l > L) add_left(--l);
      while (l < L) rem_left(l++);
    }
    ans[Q[i].id] = sum;
  }
  for (int i = 1; i <= q; i++) cout <<
      ans[i] << '\n';
}
```

4.16 Wavelet Tree

```
struct wavelet_tree {
  int low, high;
  wavelet_tree *lft = NULL, *rgt = NULL;
  int *pref = NULL;
  long long *sum = NULL; // remove if no
      need
  wavelet_tree(int *l, int *r, int low,
      int high): low(low), high(high) {
    if (l >= r || low >= high) return;
    pref = new int[r - l + 2];
    sum = new long long[r - l + 2];
    pref[0] = 0;
    sum[0] = 0;
    int mid = (low + high) >> 1, cnt = 1;
    for (int *i = l; i != r; i++, cnt++) {
      pref[cnt] = pref[cnt - 1] + ((*i)
          <= mid);
      sum[cnt] = sum[cnt - 1] + (*i);
    }
    int *pivot = stable_partition(l, r,
        [&](int x) {return x <= mid;});
    lft = new wavelet_tree(l, pivot,
        low, mid);
    rgt = new wavelet_tree(pivot, r, mid
        + 1, high);
  }
  // swaps the elements at index 'idx'
      and 'idx+1'
  void swap_adjacent(int idx) {
    if (low == high)return;
    int firstBit = pref[idx] - pref[idx
        - 1];
    sum[idx] = sum[idx - 1] + sum[idx +
        1] - sum[idx];
    if (firstBit == (pref[idx + 1] -
        pref[idx])) {
      if (firstBit)
          lft->swap_adjacent(pref[idx]);
      else rgt->swap_adjacent(idx -
          pref[idx]);
    }
    else {
      if (firstBit) pref[idx]--;
      else pref[idx]++;
    }
  }
  // count occurences of 'k' in range
      [l,r]
  int count(int l, int r, int k) {
    if (l > r || high < k || low > k)
        return 0;
    if (low == high) return r - l + 1;
    int mid = (low + high) >> 1;
    if (k <= mid) return
        lft->count(pref[l - 1] + 1,
        pref[r], k);
    return rgt->count(l - pref[l - 1], r
        - pref[r], k);
  }
  // returns the k'th smallest element in
      range [l,r]  // act like multiset
  int kth(int l, int r, int k) {
    if (l > r) return 0;
    if (low == high) return low;
    int lftCount = pref[r] - pref[l - 1];
    if (lftCount >= k) return
        lft->kth(pref[l - 1] + 1,
        pref[r], k);
    return rgt->kth(l - pref[l - 1], r -
        pref[r], k - lftCount);
  }
  // returns the count of elements that
      are less than or equal to 'k' in
      range [l,r]
  int LTE(int l, int r, int k) {
    if (l > r || low > k) return 0;
    if (high <= k) return r - l + 1;
    return lft->LTE(pref[l - 1] + 1,
        pref[r], k) + rgt->LTE(l -
        pref[l - 1], r - pref[r], k);
  }
  // returns the count of elements that
      are greater than or equal to 'k' in
      range [l,r]
  int GTE(int l, int r, int k) {
    if (l > r || high < k) return 0;
    if (low >= k) return r - l + 1;
    return lft->GTE(pref[l - 1] + 1,
        pref[r], k) + rgt->GTE(l -
        pref[l - 1], r - pref[r], k);
  }
  // returns the sum of elements less
      than 'k' in range [l,r]
  long long sum_query(int l, int r, int
      k) {
    if (l > r || low >= k) return 0;
    if (low == high) return 1LL * (r - l
        + 1) * low;
    if (high < k) return sum[r] - sum[l
        - 1];
    return lft->sum_query(pref[l - 1] +
        1, pref[r], k) +
        rgt->sum_query(l - pref[l - 1],
        r - pref[r], k);
  }
  ~wavelet_tree() {
    if (pref != NULL) delete []pref;
    if (sum != NULL) delete []sum;
    if (lft != NULL) delete lft;
    if (rgt != NULL) delete rgt;
  }
};
int32_t main() {
  int n; cin >> n;
  for (int i = 1; i <= n; i++) cin >>
      a[i];
  wavelet_tree t = wavelet_tree(a + 1, a
      + n + 1, -MAXV, MAXV); // a[i] will
      change
}
```

4.17 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++;
    }
  }
  void erase( int val ) {
    node *cur = root;
    cur->sz--;
    for (nt i = B - 1; i >= 0; i--) {
      int x = val >> i & 1;
      if (cur->nxt[x] == NULL) return;
      cur = cur->nxt[x];
      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;
```

4.18 2D BIT

```
const int N = 2500 + 10;
int a[N][N], tree[N][N];
void update(int x, int y, int val) { // x
    and y is the coordinate from range
    update
  for (int i = x; i < N; i += i & (-i)) {
    for (int j = y; j < N; j += j &
        (-j)) {
      tree[i][j] += val;
    }
  }
}
int query(int x, int y) {
  int res = 0;
  for (int i = x; i < N; i += i & (-i)) {
    for (int j = y; j < N; j += j &
        (-j)) {
      res += tree[i][j];
    }
  }
  return res;
}
void range_add(int a, int b, int c, int
    d, int val) {
  update(a, b, val);
  update(a, d + 1, -val);
  update(c + 1, b, -val);
  update(c + 1, d + 1, val);
}
```

5 Dynamic Programming

5.1 Knapsack 2

```
Constraints: N <= 100, W <= 1e9, val[i]
    <= 1000
dp[i][cur_val] = min weight needed to
    achieve cur_val from i to n. if
    dp[1][val] <= W then val is a
    candidate ans. where val is
    all_possible_val (1 to 100*1000)
```

5.2 LIS using Segment Tree

```
int32_t main() {
  int n; cin >> n;
  for (int i = 1; i <= n; i++) {
    cin >> a[i]; // a[i] must be >= 2
  }
  // dp[i] = LIS ending at pos i
  st.build(1, 1, M); // range max query,
      and upd idx with max(cur_val,
      new_val)
```

```
  for (int i = 1; i <= n; i++) {
    dp[i] = 1;
    if (a[i] != 1) {
      int mx = st.query(1, 1, M, 1, a[i]
          - 1);
      mx++;
      dp[i] = max(dp[i], mx);
    }
    st.upd(1, 1, M, a[i], dp[i]);
  }
  int ans = 0;
  for (int i = 1; i <= n; i++) ans =
      max(ans, dp[i]);
  cout << ans << '\n';
}
```

5.3 Digit DP

```
// Problem: How Many Zeroes? - LightOJ
  // How many zeroes between n to m
     (n <= m). 100 to 102 ans 4
ll fun2(int i, bool is_small) {
  if (i == sz) return 1;
  int l = 0, r = s[i] - '0';
  if (is_small) r = 9;
  ll &ans = dp2[i][is_small];
  if (ans != -1) return ans;
  ans = 0;
  for (int x = l; x <= r; x++) {
    ans += fun2(i + 1, (is_small | (x <
        r)));
  }
  return ans;
}
ll fun(int i, bool is_small, bool
    has_started) {
  if (i == sz) return 0;
  int l = 0, r = s[i] - '0';
  if (is_small) r = 9;
  ll &ans = dp1[i][is_small][has_started];
  if (ans != -1) return ans;
  ans = 0;
  for (int x = l; x <= r; x++) {
    bool new_has_started = has_started |
        (x != 0);
    ans += fun(i + 1, (is_small | (x <
        r)), new_has_started);
    if (x == 0 and has_started) {
      ans += fun2(i + 1, (is_small | (x
          < r)));
    }
  }
  return ans;
}
void get(long long x) {
  if (x < 0) return; s = "";
  while (x > 0) {
    char c = (x % 10) + '0';
    s += c; x /= 10;
  }
  reverse(s.begin(), s.end());
  sz = s.size();
  memset(dp1, -1, sizeof(dp1));
  memset(dp2, -1, sizeof(dp2));
}
void solve() {
  ll n, m; cin >> n >> m;
  get(n - 1);
  memset(dp1, -1, sizeof(dp1));
  memset(dp2, -1, sizeof(dp2));
  ll ans1 = (n == 0) ? 0 : fun(0, false,
      false);
  get(m);
  ll ans2 = fun(0, false, false);
  cout << ans2 - ans1 + (n == 0) << '\n';
}
```

5.4 Bitmask DP

```
// Problem: DNA Sequence - LOJ
// Given n strings, find the shortest
    string that contains all the given
    strings as substrings and
    lexicographically smallest.
const int N = 17;
int n, tail[N][N], dp[(1 << N) + 2][N +
    2];
string s[N + 2];
bool cmp(string a, string b) {
  if (a.size() < b.size()) {
    return true;
  }
  return false;
}
int fun(int mask, int last) {
  if (__builtin_popcount(mask) >= n)
      return 0;
  int &ans = dp[mask][last];
  if (ans != -1) return ans;
  ans = 1e9;
  for (int j = 1; j <= n; j++) {
    if (!(mask & (1 << j))) {
      if (last == n + 1) {
        int x = s[j].length() + fun(mask
            | (1 << j), j);
        ans = min(ans, x);
      }
      else {
        int x = (s[j].length() -
            tail[last][j]) + fun(mask |
            (1 << j), j);
        ans = min(ans, x);
      }
    }
  }
  return ans;
}
void print(int mask, int last) {
  if (__builtin_popcount(mask) >= n)
      return;
  int ans = fun(mask, last), idx = -1;
  string str = "{";
  for (int j = 1; j <= n; j++) {
    if (!(mask & (1 << j))) {
      if (last == n + 1) {
        int x = s[j].length() + fun(mask
            | (1 << j), j);
        if (x == ans) {
          string d = s[j];
          if (d <= str) str = d; idx = j;
        }
      }
      else {
        int x = (s[j].length() -
            tail[last][j]) + fun(mask |
            (1 << j), j);
        if (x == ans) {
          string d =
              s[j].substr(tail[last][j]);
          if (d <= str) str = d; idx = j;
        }
      }
    }
  }
  cout << s[idx].substr(tail[last][idx]);
  print(mask | (1 << idx), idx);
}
void solve() {
  cin >> n;
  string str[n + 1];
  for (int i = 1; i <= n; i++) {
    cin >> str[i]; // len <= 100
  }
  // remove duplicates and strings which
      is a subarray of others
  sort(str + 1, str + n + 1, cmp);
  vector<string> vec;
  for (int i = 1; i <= n; i++) {
    bool ok = true;
    for (int j = i + 1; j <= n; j++) {
      if (str[j].find(str[i]) !=
          string::npos) {
        ok = false;
        break;
      }
    }
    if (ok) vec.push_back(str[i]);
  }
  n = vec.size();
  int idx = 1;
  for (auto x : vec) s[idx++] = x;
  // maximum length that suffix of a[i] =
      prefix of a[j]
  memset(tail, 0, sizeof(tail));
  for (int i = 1; i <= n; i++) {
    string x = s[i];
    for (int j = 1; j <= n; j++) {
      string y = s[j];
      int cnt = 0;
      for (int k = min(x.length(),
          y.length()); k >= 1; k--) {
        if (x.substr(x.length() - k) ==
            y.substr(0, k)) {
          cnt = k;
          break;
        }
      }
      tail[i][j] = cnt;
    }
  }
  memset(dp, -1, sizeof(dp));
  fun(0, n + 1);
  print(0, n + 1);
}
```

5.5 MCM DP

```
// Problem: Slimes - Atcoder DP Contest
// Given n slimes. Choose two adjacent
    slimes, and combine them into a new
    slime. The new slime has a size of
    x+y, where x and y are the sizes of
    the slimes before combining them.
    Here, a cost of x+y is incurred.
    Example-
// (10, 20, 30, 40) -> (30, 30, 40)
// (30, 30, 40) -> (60, 40)
// (60, 40) -> (100) ans = 190
      // Solution: Think reverse. We
    are given the final sum, from i to j.
    Now we will cut any point between i to
    j and calculate the cost
// Time: O(n^3)
ll fun(int i, int j) {
  if (i == j) return 0;
  ll &ans = dp[i][j];
  if (ans != -1) return ans;
  ll cur = 0;
  for (int x = i; x <= j; x++) {
    cur += a[x];
  }
  ans = inf;
  for (int x = i; x < j; x++) {
    ans = min(ans, cur + fun(i, x) +
        fun(x + 1, j));
  }
  return ans;
}
cout << fun(1, n) << '\n';
```

5.6 Number of Unique Subsequence

```
int nxt[N][26], dp[N];
int f(int i) {
  if (i > n) return 0;
  int &ans = dp[i];
  if (ans != -1) return ans;
  ans = 1;
  for (int c = 0; c < 26; c++) {
    ans = (ans + f(nxt[i][c])) % mod;
  }
  return ans;
}
// nxt[i][c] = nxt c strictly after i
void solve() {
  cin >> s; s = '.' + s;
  cout << f(0) << '\n';
}
```

5.7 [Problem] Rose Land - SUST 24

```
      // Given a complete
    binary tree, i-th node grows a_i roses
    per day and deliver to u in dis(u,i)
    days. Alice at node u, How many roses
    he will get within t days? u and t for
    Q queries.
    // Idea: as it is a complete binary
    tree, max dis is 2*logn. We can do DP
    on Tree for ~40 days.
ll dp[N][45], dp2[N][45];
void dfs(int u, int p) {
  for (int i = 1; i <= 40; i++) {
```

```
    dp[u][i] = i * a[u];
  }
  for (auto v : g[u]) {
    if (v != p) {
      dfs(v, u);
      for (int i = 1; i <= 40; i++) {
        dp[u][i] += dp[v][i - 1];
      }
    }
  }
}
void dfs2(int u, int p) {
  for (int i = 2; u != 1 and i <= 40;
      i++) {
    dp2[u][i] = (dp[p][i - 1] + dp2[p][i
        - 1]) - dp[u][i - 2];
  }
  for (auto v : g[u]) {
    if (v != p) {
      dfs2(v, u);
    }
  }
}
int32_t main() {
  cin >> n; ll sum = 0;
  for (int i = 1; i <= n; i++) {
    cin >> a[i]; sum += a[i];
  }
  dfs(1, 0); // rose from subtree
  dfs2(1, 0); // rose from par
  int q; cin >> q;
  while (q--) {
    int u, t; cin >> u >> t;
    int extra = t - 40;
    t = min(40, t);
    ll ans = dp[u][t] + dp2[u][t];
    if (extra > 0) ans += 1ll * extra *
        sum;
    cout << ans << '\n';
  }
}
```

5.8 [Problem] Sub-Palindromic Tree

```
// Given a tree, each node has a char.
    Print max len subsequence any path
    which is a palindrome. (CF/1771/D)
// Time: O(N^2)
int nxt[N][N], dp[N][N];
vector<int> vec;
void dfs(int u, int p) {
  vec.push_back(u);
  for (auto v: g[u]) {
    if (v != p) dfs(v, u);
  }
}
int f(int u, int v) {
  if (v == u) return 1;
  int &ans = dp[u][v];
  if (ans != -1) return ans;
  ans = 0;
  if (s[u] == s[v]) ans = 2 + (nxt[u][v]
      == v ? 0 : f(nxt[u][v],
      nxt[v][u]));
  else ans = max(f(nxt[u][v], v), f(u,
      nxt[v][u]));
  return ans;
}
void solve() {
  cin >> n >> s; s = '.' + s;
  for (int i = 1; i < n; i++) {
    int u, v; cin >> u >> v;
    g[u].push_back(v);
    g[v].push_back(u);
  }
          // nxt[u][v] = next node after u
      if I want to go from u to v
  for (int u = 1; u <= n; u++) {
    for (auto x : g[u]) {
      vec.clear();
      dfs(x, u);
      for (auto v : vec) nxt[u][v] = x;
    }
  }
  memset(dp, -1, sizeof dp);
  int ans = 0;
  for (int u = 1; u <= n; u++) {
    for (int v = 1; v <= n; v++) {
      ans = max(ans, f(u, v));
    }
  }
  cout << ans << '\n';
}
```

5.9 SOS DP

```
void AmeDoko() {
  fill(f, f + (1 << N), INT_MAX);
  int n; cin >> n;
  for (int i = 0; i < n; i++) {
    cin >> a[i];
    f[a[i]] = a[i];
  }
  for (int mask = 0; mask < (1 << N);
      mask++) {
    for ( int i = mask; i > 0; i -= (i &
        -i)) {
      int temp = (mask ^ ( i & -i ));
      if ( f[temp] != INT_MAX ) {
        f[mask] = f[temp];
        break;
      }
    }
  }
  for (int i = 0; i < n; i++) {
    if (f[ ~a[i] & ((1 << N) - 1) ] !=
        INT_MAX) cout << f[ ~a[i] & ((1
        << N) - 1) ] << ' ';
    else cout << -1 << ' ';
  }
}
```

6 Graph Theory

6.1 Binary Lifting and LCA

```
const int N = 2e5 + 9, LOG = 20;
vector<int> g[N];
int par[N][LOG], depth[N];
void dfs(int u, int p) {
  par[u][0] = p;
  depth[u] = depth[p] + 1;
  for (int i = 1; i < LOG; i++) {
    par[u][i] = par[par[u][i - 1]][i - 1];
  }
  for (auto v : g[u]) {
    if (v != p) {
      dfs(v, u);
    }
  }
}
int lca(int u, int v) {
  if (depth[u] < depth[v]) {
    swap(u, v);
  }
  int k = depth[u] - depth[v];
  for (int i = 0; i < LOG; i++) {
    if (CHECK(k, i)) u = par[u][i];
  }
  if (u == v) return u;
  for (int i = LOG - 1; i >= 0; i--) {
    if (par[u][i] != par[v][i]) {
      u = par[u][i];
      v = par[v][i];
    }
  }
  return par[u][0];
}
int kth(int u, int k) { // kth parent of
    u
  assert(k >= 0);
  for (int i = 0; i < LOG; i++) {
    if (CHECK(k, i)) u = par[u][i];
  }
  return u;
}
int dist(int u, int v) { // distance from
    u to v
  int l = lca(u, v);
  return (depth[u] - depth[l]) +
      (depth[v] - depth[l]);
}
// kth node from u to v, 0th node is u
int kth(int u, int v, int k) {
  int l = lca(u, v);
  int d = dist(u, v);
  assert(k <= d);
  if (depth[l] + k <= depth[u]) {
    return kth(u, k);
  }
  k -= depth[u] - depth[l];
  return kth(v, depth[v] - depth[l] - k);
}
```

6.2 LCA and Sparse Table on Tree

```
// max and min weights of a path
const int N = 1e5 + 9, LOG = 20, inf =
    1e9; // change here
vector<array<int, 2>> g[N];
int par[N][LOG], tree_mx[N][LOG],
    depth[N];
void dfs(int u, int p, int dis) {
  par[u][0] = p;
  tree_mx[u][0] = dis;
  depth[u] = depth[p] + 1;
  for (int i = 1; i < LOG; i++) {
    par[u][i] = par[par[u][i - 1]][i - 1];
    tree_mx[u][i] = max(tree_mx[u][i -
        1], tree_mx[par[u][i - 1]][i -
        1]);
  }
  for (auto [v, w] : g[u]) {
    if (v != p) {
      dfs(v, u, w);
    }
  }
}
int query_max(int u, int v) { // max
    weight on path u to v
  int l = lca(u, v);
  int d = dist(l, u);
  int ans = 0;
  for (int i = 0; i < LOG; i++) {
    if (CHECK(d, i)) {
      ans = max(ans, tree_mx[u][i]);
      u = par[u][i];
    }
  }
  d = dist(l, v);
  for (int i = 0; i < LOG; i++) {
    if (CHECK(d, i)) {
      ans = max(ans, tree_mx[v][i]);
      v = par[v][i];
    }
  }
  return ans;
}
```

6.3 Dijkstra

```
vector<int> dijkstra(int s) {
  vector<int> dis(n + 1, inf);
  vector<bool> vis(n + 1, false);
  dis[s] = 0;
  priority_queue<array<int, 2>,
      vector<array<int, 2>>,
      greater<array<int, 2>>> pq;
  pq.push({0, s});
  while (!pq.empty()) {
    auto [d, u] = pq.top(); pq.pop();
    if (vis[u]) continue;
    vis[u] = true;
    for (auto [v, w] : g[u]) {
      if (dis[v] > d + w) {
        dis[v] = d + w;
        pq.push({dis[v], v});
      }
    }
  }
  return dis;
}
```

6.4 Bellman Ford

```
// works for neg edge, can detect neg
    cycle
// Time: O(n^2)
const ll inf = 1e18;
vector<ll> dis(N, inf);
bool bellman_ford(int s) {
  dis[s] = 0;
  bool has_cycle = false;
  for (int i = 1; i <= n; i++) {
    for (int u = 1; u <= n; u++) {
```

```
      for (auto [v, w] : g[u]) {
        if (dis[v] > dis[u] + w) {
          if (i == n) has_cycle = true;
          dis[v] = dis[u] + w;
        }
      }
    }
  }
  return has_cycle;
}
```

6.5 Floyd Warshall

```
// dis[i][j] = min distance to reach i to
    j, works for neg edge (no neg cycle)
// Time: O(n^3)
vector<int> construct_path(int u, int v)
    { // O(n)
  if (nxt[u][v] == -1) return {};
  vector<int> path = { u };
  while (u != v) {
    u = nxt[u][v];
    path.push_back(u);
  }
  return path;
}
void floyd_warshall() {
  for (int i = 1; i <= n; i++) {
    for (int j = 1; j <= n; j++) {
      if (i == j) dis[i][j] = 0;
      else if (g[i][j] == 0) dis[i][j] =
          inf;
      else dis[i][j] = g[i][j];
    }
  }
  for (int k = 1; k <= n; ++k) {
    for (int i = 1; i <= n; ++i) {
      for (int j = 1; j <= n; ++j) {
        if (dis[i][k] < inf and
            dis[k][j] < inf)
          dis[i][j] = min(dis[i][j],
              dis[i][k] + dis[k][j]);
          nxt[i][j] = nxt[i][k];
      }
    }
  }
}
int32_t main() {
  int q; cin >> n >> m >> q;
  memset(nxt, -1, sizeof nxt);
  while (m--) {
    int u, v, w; cin >> u >> v >> w;
    g[u][v] = (g[u][v] != 0 ?
        min(g[u][v], w) : w);
    g[v][u] = (g[v][u] != 0 ?
        min(g[v][u], w) : w);
    nxt[u][v] = v;
    nxt[v][u] = u;
  }
  floyd_warshall();
  while (q--) {
    int u, v; cin >> u >> v;
    cout << (dis[u][v] == inf ? -1 :
        dis[u][v]) << '\n';
  }
  return 0;
}
```

6.6 Strongly Connected Components

```
// Time: O(n + m)
const int N = 1e5 + 9;
vector<int> g[N], gT[N], G[N];
vector<bool> vis(N, false);
vector<vector<int>> components;
vector<int> order;
int n, roots[N], sz[N];
void dfs(int u) {
  vis[u] = true;
  for (auto v : g[u]) {
    if (!vis[v]) dfs(v);
  }
  order.push_back(u);
}
void dfs2(int u, vector<int> &component) {
  vis[u] = true;
  component.push_back(u);
  for (auto v : gT[u]) {
    if (!vis[v]) dfs2(v, component);
  }
}
void scc() {
  // get order sorted by end time
  order.clear();
  for (int u = 1; u <= n; u++) {
    if (!vis[u]) dfs(u);
  }
  reverse(order.begin(), order.end());
  // transpose the graph
  for (int u = 1; u <= n; u++) {
    for (auto v : g[u]) {
      gT[v].push_back(u);
    }
  }
  // get all components
  components.clear();
  for (int i = 1; i <= n; i++) vis[i] =
      false;
  for (auto u : order) {
    if (!vis[u]) {
      vector<int> component;
      dfs2(u, component);
      sort(component.begin(),
          component.end());
      components.push_back(component);
      for (auto v : component) {
        roots[v] = component.front();
        sz[v] = component.size();
      }
    }
  }
  // add edges to condensation graph
  for (int u = 1; u <= n; u++) {
    for (auto v : g[u]) {
      if (roots[u] != roots[v]) {
        G[roots[u]].push_back(roots[v]);
      }
    }
  }
}
// when you need to use condensed graph,
    use it carefully (Specially g->G,
    i->roots[i])
```

6.7 Articulation Points

```
int disc[N], low[N], timer, n;
vector<bool> vis(N, false), is_ap(N,
    false);
void ap_dfs(int u, int p) {
  disc[u] = low[u] = ++timer;
  vis[u] = true;
  int children_cnt = 0;
  for (auto v : g[u]) {
    if (v == p) continue;
    if (vis[v]) low[u] = min(low[u],
        disc[v]);
    else {
      ap_dfs(v, u);
      low[u] = min(low[u], low[v]);
      if (disc[u] <= low[v] and p != -1)
          is_ap[u] = true;
      children_cnt++;
    }
  }
  if (p == -1 and children_cnt > 1)
      is_ap[u] = true;
}
void find_articulation_points() {
  for (int u = 1; u <= n; u++) {
    if (!vis[u]) {
      timer = 0;
      ap_dfs(u, -1);
    }
  }
}
```

6.8 Find Bridges

```
map<pair<int, int>, int> bridges;
void bridges_dfs(int u, int p) { // find
    bridges
  disc[u] = low[u] = ++timer;
  vis[u] = true;
  for (auto v : g[u]) {
    if (v == p) continue;
    if (vis[v]) low[u] = min(low[u],
        disc[v]);
    else {
      bridges_dfs(v, u);
      low[u] = min(low[u], low[v]);
      if (disc[u] < low[v]) {
        bridges[(make_pair(min(u, v),
            max(u, v)))]++;
      }
    }
  }
}
void find_bridges() {
  for (int u = 1; u <= n; u++) {
    if (!vis[u]) {
      timer = 0;
      bridges_dfs(u, -1);
    }
  }
}
```

6.9 DSU on Tree

```
// Problem: Distinct Colors CSES (Number
    of distinct color in a subtree)
// Complexity: O(n(logn)^2)
set<int> se[N];
 int col[N], ans[N], par[N];  // par[i] = i
    initially
int find(int i) {
  return (i == par[i] ? i : par[i] =
      find(par[i]));
}
void merge(int u, int v) {
  if ((u = find(u)) == (v = find(v)))
      return;
  if (se[u].size() > se[v].size())
      swap(u, v);
  for (auto x : se[u]) {
    se[v].insert(x);
  }
  se[u].clear();
  par[u] = v;
}
void dfs(int u, int p) {
  se[find(u)].insert(col[u]);
  for (auto v : g[u]) {
    if (v != p) {
      dfs(v, u);
      merge(u, v);
    }
  }
  ans[u] = se[find(u)].size();
}
```

6.10 Heavy-Light Decomposition

```
// Per Query Complexity: O(logn^2)
// Path and subtree updates and queries.
const int N = 2e5 + 9, LOG = 20, inf =
    1e9; // change here
vector<int> g[N];
int par[N][LOG], depth[N], sz[N];
int disc[N], finish[N], timer, head[N];
int n;
void dfs(int u, int p = 0) {
  par[u][0] = p;
  depth[u] = depth[p] + 1;
  sz[u] = 1;
  for (int i = 1; i < LOG; i++) {
    par[u][i] = par[par[u][i - 1]][i - 1];
  }
  if (p) g[u].erase(find(g[u].begin(),
      g[u].end(), p));
  for (auto &v : g[u]) {
    if (v != p) {
      dfs(v, u);
      sz[u] += sz[v];
      if (sz[v] > sz[g[u][0]]) swap(v,
          g[u][0]);
    }
  }
}
void dfs_hld(int u) {
  disc[u] = ++timer;
  for (auto v : g[u]) {
    head[v] = (v == g[u][0] ? head[u] :
        v);
```

```
    dfs_hld(v);
  }
  finish[u] = timer;
}
int kth(int u, int k) {
  assert(k >= 0);
  for (int i = 0; i < LOG; i++) {
    if (CHECK(k, i)) u = par[u][i];
  }
  return u;
}
int query_up(int u, int v) {
  int ans = -inf; // change here
  while (head[u] != head[v]) {
    ans = max(ans, st.query(1, 1, n,
        disc[head[u]], disc[u])); //
        change here
    u = par[head[u]][0];
  }
  ans = max(ans, st.query(1, 1, n,
      disc[v], disc[u])); // change here
  return ans;
}
int query(int u, int v) {
  int lc = lca(u, v);
  int ans = query_up(u, lc);
  if (v != lc) {
    ans = max(ans, query_up(v, kth(v,
        depth[v] - depth[lc] - 1))); //
        change here
  }
  return ans;
}
void solve() {
  cin >> 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);
  head[1] = 1;
  dfs_hld(1);
  st.build(1, 1, n);
}
```

6.11 Inverse Graph

```
// Problem: CF/0-1 MST
  // A Complete Graph. m edges have
     weight 1, rest have 0. MST?
const int N = 200000 + 9;
set<int> g[N], non_vis;
void dfs(int u) {
  non_vis.erase(u);
  for (int v = 1; v <= n; v++) {
    if (v == u) continue;
    auto it = non_vis.lower_bound(v);
    if (it == non_vis.end()) break;
    v = *it;
    if (g[u].find(v) != g[u].end())
        continue;
    dfs(v);
  }
}
int32_t main() {
  int m; cin >> n >> m;
  while (m--) {
    int u, v; cin >> u >> v;
    g[u].insert(v);
    g[v].insert(u);
  }
  for (int i = 1; i <= n; i++)
      non_vis.insert(i);
  int component_cnt = 0;
  for (int i = 1; i <= n; i++) {
    if (non_vis.find(i) !=
        non_vis.end()) {
      component_cnt++;
      dfs(i);
    }
  }
  cout << component_cnt - 1 << '\n';
}
```

6.12 [Problem] Rooted Tree - Hackerrank

Given a rooted tree, Upd: Add $d * k$ on v(subtree nodes of u), where k will be given and d is distance of v from u. Query: Node Value/Subtree Sum (using ETT) or Path Value Sum (using HLD). **Idea:** d_i = dis from root to i. Then, ans $= k \cdot \sum_{v \in \text{subtree}(u)} d_v$. It's hard to find d_v for each u, rather we can find d_v = dis of v from root. Although some extra value added on each v of subtree u because of d_v from root (not u), we can easily subtract them ($depth[u] * k$).

6.13 Dinic

```
const int N = 105, inf = 1e9;
int n, m, st, en;
// Edges should be added in both
    direction separately if the graph is
    undirected
const int INF = 2000000000;
struct Edge {
  int from, to, cap, flow, index;
  Edge(int from, int to, int cap, int
      flow, int index) :
    from(from), to(to), cap(cap),
        flow(flow), index(index) {}
};
struct Dinic {
  int N;
  vector<vector<Edge> > G;
  vector<Edge*> dad;
  vector<int> Q;
  Dinic(int N) : N(N), G(N), dad(N),
      Q(N) {}
  void AddEdge(int from, int to, int
      cap) {
    G[from].emplace_back(from, to, cap,
        0, G[to].size());
    if (from == to)
        G[from].back().index++;
    G[to].emplace_back(to, from, 0, 0,
        G[from].size() - 1);
  }
  long long BlockingFlow(int s, int t) {
    fill(dad.begin(), dad.end(), (Edge
        *) NULL);
    dad[s] = &G[0][0] - 1;
    int head = 0, tail = 0;
    Q[tail++] = s;
    while (head < tail) {
      int x = Q[head++];
      for (int i = 0; i < G[x].size();
          i++) {
        Edge &e = G[x][i];
        if (!dad[e.to] && e.cap - e.flow
            > 0) {
          dad[e.to] = &G[x][i];
          Q[tail++] = e.to;
        }
      }
    }
    if (!dad[t]) return 0;
    long long totflow = 0;
    for (int i = 0; i < G[t].size();
        i++) {
      Edge *start =
          &G[G[t][i].to][G[t][i].index];
      int amt = INF;
      for (Edge *e = start; amt && e !=
          dad[s]; e = dad[e->from]) {
        if (!e) { amt = 0; break; }
        amt = min(amt, e->cap - e->flow);
      }
      if (amt == 0) continue;
      for (Edge *e = start; amt && e !=
          dad[s]; e = dad[e->from]) {
        e->flow += amt;
        G[e->to][e->index].flow -= amt;
      }
      totflow += amt;
    }
    return totflow;
  }
  long long GetMaxFlow(int s, int t) {
    long long totflow = 0;
    while (long long flow =
        BlockingFlow(s, t))
      totflow += flow;
    return totflow;
  }
};
void solve() {
  cin >> n >> st >> en >> m;
  Dinic *dinic = new Dinic(n + 1);
  while (m--) {
    int u, v, w; cin >> u >> v >> w;
    dinic->AddEdge(u, v, w);
    dinic->AddEdge(v, u, w);
  }
  cout << dinic->GetMaxFlow(st, en) <<
      '\n';
  delete(dinic);
}
```

7 String

7.1 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);
  }
};
```

7.2 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;
    }
    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);
  }
};
```

7.3 Hashing with Upd and Deletes

```
// 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;
    }
  }
}
```

7.4 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";
}
```

7.5 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);
}
```

7.6 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;
}
```

7.7 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]);
    }
  }
}
```

7.8 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]++;
```

7.9 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.

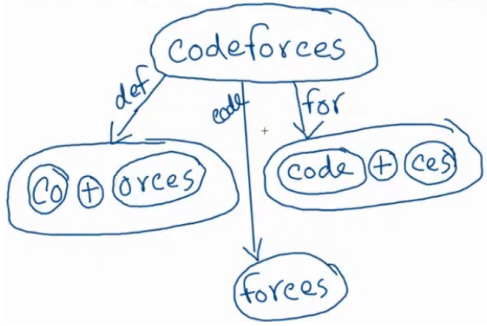
7.10 Trie

```
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
}
```

8 Game Theory

8.1 Notes

- First Write a Bruteforce Solution
- If game is not impartial. Greedy, DP may work
- Nim = All XOR
- Misere Nim (Last player who took a stone loses) = Nim + (Corner Case: if all (odd) piles are 1 you lose)
- Bogus Nim = Nim
- Staircase Nim (Given an array where $a[i]$ is the number of stones at index i. Move: Choose any stones (> 1) from index i ($i > 1$) and move them to index $i-1$.) = Odd indexed pile Nim (Even indexed pile doesnt matter, as one player can give bogus moves to drop all even piles to ground)
- **Sprague-Grundy:** Every impartial game under the normal play convention is equivalent to a ONE-HEAP GAME of NIM.
  Grundy value can be calculate by DP/Pattern.



9 Misc.

9.1 [Trick] Median of Medians

Problem: Median of All Sub-Array Medians
Idea: Total Subarray, $k = \frac{n(n+1)}{2}$. We need all subarray medians, sort them and get $\frac{k}{2} + 1$. Now, Binary Search on Answer. ok() will return true if $cnt >= \frac{k}{2} + 1$, where cnt = count of subarrays whose median is $<= mid$. If true, $r = mid - 1$

9.2 Ternary Search

```
// Problem: Pyramid-ICPC Dhaka 2023
// Given the surface area of a square
    base pyramid, Need to maximize
    volume.
double suface_area;
double fun(double square_area) {
  double base = sqrt(square_area);
  double triangle_area = suface_area -
      square_area;
  double per_triangle_area =
      triangle_area / 4;
  double triangle_height =
      (per_triangle_area * 2) / base;
  double x = base / 2;
  double height = sqrt((triangle_height
      * triangle_height) - (x * x));
  double volume = (base * base * height)
      / 3;
  if (x > triangle_height) volume = 0;
  return volume;
}
```

```
int32_t main() {
  cin >> suface_area;
  double l = 0, r = suface_area, ans = -1;
  int it = 100;
  while (it--) {
    double mid1 = l + (r - l) / 3;
    double mid2 = r - (r - l) / 3;
    double x = fun(mid1);
    double y = fun(mid2);
    if (x > y) {
      ans = x;
      r = mid2;
    }
    else l = mid1;
  }
}
```

9.3 Pigeonhole Principle

- At least 1 subarray of an array of length N must be divisible by N.
- Build all possible sequences of length 10 whose value is between 1 to 100. At least any two sequences will be same.
- For $N >= 100$, at least one 4-tuples XOR is 0.
- For $N > 20$ and $a_i <= 10^6$, at least two subsets has same XOR. Because distinct XOR count can be 10^6 and Total Subset $2^{20} > 10^6$. Means there is a subset whose XOR is 0.

9.4 Matrix Expo

```
struct Mat {
  int n, m;
  vector<vector<int>> a;
  Mat() { }
  Mat(int _n, int _m) {n = _n; m = _m;
      a.assign(n, vector<int>(m, 0)); }
  Mat(vector< vector<int> > v) { n =
      v.size(); m = n ? v[0].size() : 0;
      a = v; }
  inline void make_unit() {
    assert(n == m);
    for (int i = 0; i < n; i++)  {
      for (int j = 0; j < n; j++)
          a[i][j] = i == j;
    }
  }
  inline Mat operator + (const Mat &b) {
    assert(n == b.n && m == b.m);
    Mat ans = Mat(n, m);
    for(int i = 0; i < n; i++) {
      for(int j = 0; j < m; j++) {
        ans.a[i][j] = (a[i][j] +
            b.a[i][j]) % mod;
      }
    }
    return ans;
  }
  inline Mat operator - (const Mat &b) {
    assert(n == b.n && m == b.m);
    Mat ans = Mat(n, m);
    for(int i = 0; i < n; i++) {
      for(int j = 0; j < m; j++) {
        ans.a[i][j] = (a[i][j] -
            b.a[i][j] + mod) % mod;
      }
    }
    return ans;
  }
  inline Mat operator * (const Mat &b) {
    assert(m == b.n);
    Mat ans = Mat(n, b.m);
    for(int i = 0; i < n; i++) {
      for(int j = 0; j < b.m; j++) {
        for(int k = 0; k < m; k++) {
          ans.a[i][j] = (ans.a[i][j] +
              1LL * a[i][k] * b.a[k][j]
              % mod) % mod;
        }
      }
    }
    return ans;
  }
  inline Mat pow(long long k) {
    assert(n == m);
    Mat ans(n, n), t = a; ans.make_unit();
    while (k) {
      if (k & 1) ans = ans * t;
      t = t * t;
      k >>= 1;
    }
    return ans;
  }
  inline Mat& operator += (const Mat& b)
      { return *this = (*this) + b; }
  inline Mat& operator -= (const Mat& b)
      { return *this = (*this) - b; }
  inline Mat& operator *= (const Mat& b)
      { return *this = (*this) * b; }
  inline bool operator == (const Mat& b)
      { return a == b.a; }
  inline bool operator != (const Mat& b)
      { return a != b.a; }
};
int32_t main() {
  int n; long long k; cin >> n >> k;
  Mat a(n, n); // then assign value
  Mat ans = a.pow(k);
}
```

9.5 2D Prefix Sum

```
const int N = 1e3 + 9;
int a[N][N];
ll pref[N][N];
void query(int x1, int y1, int x2, int
    y2) {
    return pref[x2][y2] - pref[x1 -
        1][y2] - pref[x2][y1 - 1] +
        pref[x1 - 1][y1 - 1]
}
int32_t main() {
  int n, m; cin >> n >> m;
  for (int i = 1; i <= n; i++) {
    for (int j = 1; j <= m; j++) {
      pref[i][j] = pref[i - 1][j] +
          pref[i][j - 1] - pref[i - 1][j
          - 1] + a[i][j];
    }
  }
  int q; cin >> q;
  while (q--) {
    int x1, y1, x2, y2; cin >> x1 >> y1
        >> x2 >> y2;
    cout << query(x1, y1, x2, y2) << '\n';
  }
}
```

9.6 2D Static Range Update

```
    // Add x on a rectangle q times,
    Finally print the array
const int N = 1e3 + 9;
int a[N][N];
ll d[N][N]; // difference array
int32_t main() {
  int n, m; cin >> n >> m;
  int q; cin >> q;
  while (q--) {
    int x1, y1, x2, y2, x; cin >> x1 >>
        y1 >> x2 >> y2 >> x; // add x on
        this rectangle
    d[x1][y1] += x;
    d[x1][y2 + 1] -= x;
    d[x2 + 1][y1] -= x;
    d[x2 + 1][y2 + 1] += x;
  }
  for (int i = 1; i <= n; i++) {
    for (int j = 1; j <= m; j++) {
      d[i][j] += d[i - 1][j] + d[i][j -
          1] - d[i - 1][j - 1];
    }
  }
  // new updated array
  for (int i = 1; i <= n; i++) {
    for (int j = 1; j <= m; j++) {
      cout << d[i][j] + a[i][j] << ' ';
    }
    cout << '\n';
  }
}
```

9.7 CHECK SET CLR

```
bool CHECK(int N, int pos) { return
    (bool)(N & (1ll << pos)); }
void SET(int &N, int pos) { (N |= (1ll
    << pos)); }
void CLR(int &N, int pos) { (N &= ~(1ll
    << pos)); }
```

9.8 Graph Directions

```
int dx[] = {+0, +0, +1, -1, -1, +1, -1,
    +1};
int dy[] = {-1, +1, +0, +0, +1, +1, -1,
    -1};
```

9.9 Custom Hash

```
struct custom_hash {
  static uint64_t splitmix64(uint64_t x) {
    x += 0x9e3779b97f4a7c15;
    x = (x ^ (x >> 30)) *
        0xbf58476d1ce4e5b9;
    x = (x ^ (x >> 27)) *
        0x94d049bb133111eb;
    return x ^ (x >> 31);
  }
  size_t operator()(uint64_t x) const {
    static const uint64_t FIXED_RANDOM =
        chrono::steady_clock::now().time
        _since_epoch().count();
    return splitmix64(x + FIXED_RANDOM);
  }
};
```

9.10 Coordinate Compression

```
int32_t main() {
  vector<int> a({100, 9, 10, 10, 9});
  vector<int> v = a;
  sort(v.begin(), v.end());
  v.resize(unique(v.begin(), v.end()) -
      v.begin());
  for (int i = 0; i < a.size(); i++) {
    a[i] = lower_bound(v.begin(),
        v.end(), a[i]) - v.begin() + 1;
  }
}
```

9.11 Submask Generator

```
int mask = 0; // any value
for (int submask = mask; submask;
    submask = (submask - 1) & mask) {}
```

9.12 Custom Comparators

```
bool cmp(pair<int, int> a, pair<int,
    int> b) { // for vector, arr,..
  if (a.first != b.first) return a.first
      > b.first;
  return a.second < b.second; // must
      return false for equal elements
}
struct cmp { // for set, map, pq,..
  bool operator()(const int& a, const
      int&b) const {
    return a > b;
  }
};
```

9.13 MOD Operations

```
inline ll modAdd(ll a, ll b, ll MOD) {ll
    res = a + b; res += ((res >> 63) &
    MOD); if (res >= MOD) res -= MOD;
    return res;}
inline ll modSub(ll a, ll b, ll MOD) {ll
    res = a - b; res += ((res >> 63) &
    MOD); if (res >= MOD) res -= MOD;
    return res;}
inline ll modMul(ll a, ll b, ll MOD) {a
    %= MOD; b %= MOD; a += ((a >> 63) &
    MOD); b += ((b >> 63) & MOD); return
    (a * b) % MOD; }
inline ll modInverse(ll a, ll MOD)
    {return modPow(a, MOD - 2, MOD);}
inline ll modDiv(ll a, ll b, ll MOD)
    {return modMul(a, modInverse(b,
    MOD), MOD);}
inline ll modMulBigMod(ll a, ll b, ll
    MOD) {a %= MOD; b %= MOD; a += ((a
    >> 63) & MOD); b += ((b >> 63) &
    MOD); return (ll)((__int128)a * b %
    MOD);}
```

9.14 Mex with Array Updates

```
int32_t main() {
  int n, q; cin >> n >> q;
  set<int> missing_numbers;
  for (int i = 0; i <= n + 200; i++) {
    missing_numbers.insert(i);
  }
  int a[n + 1];
  map<int, int> freq;
  for (int i = 1; i <= n; i++) {
    cin >> a[i];
    freq[a[i]]++;
    missing_numbers.erase(a[i]);
  }
  while (q--) {
    int i, x; cin >> i >> x;  // Set
      a[i] = x
    int y = a[i]; a[i] = x;
    missing_numbers.erase(x);
    freq[y]--;
    freq[x]++;
    if (freq[y] == 0) {
      freq.erase(y);
      missing_numbers.insert(y);
    }
    cout << *missing_numbers.begin() <<
        '\n'; // mex after upd
  }
}
```

10 Geometry

10.1 Convex Hull

```
typedef vector<long long> vll;
typedef long double ld;
#define int long long
struct point {
  int x, y;
  point() {}
  point( int x, int y ) { this->x = x;
      this->y = y; }
  bool operator<( const point &other ) {
    if ( x == other.x ) return y <
        other.y;
    return x < other.x;
  }
};
int crossProduct( point &a, point &b,
    point &c ) {
  return (b.x - a.x) * (c.y - a.y) -
      (b.y - a.y) * (c.x - a.x);
}
vector<point> hull( vector<point> a ) {
  vector<point> v = a;
  sort(v.begin(), v.end());
  vector<point> lower, upper;
  for ( auto& p : v ) {
    while ( lower.size() >= 2 &&
        crossProduct(lower[lower.size()
        - 2], lower.back(), p) < 0 ) //
        <= for coleniar
      lower.pop_back();
    lower.push_back(p);
  }
  for ( int i = v.size() - 1; i >= 0;
      i-- ) {
    point p = v[i];
    while ( upper.size() >= 2 &&
        crossProduct(upper[upper.size()
        - 2], upper.back(), p) < 0 ) //
        <= for coleniar
      upper.pop_back();
    upper.push_back(p);
  }
  lower.pop_back();
  upper.pop_back();
  lower.insert(lower.end(),
      upper.begin(), upper.end());
  return lower;
}
void senritsu() {
  int n; cin >> n;
  vector<point> a(n); for ( int i = 0; i
      < n; i++ ) cin >> a[i].x >> a[i].y;
  a = hull(a);
  n = a.size();
  cout << n << endl;
  for ( auto c : a ) cout << c.x << ' '
      << c.y << endl;
}
```

10.2 Area of 2 polygon intersection

```
typedef vector<long long> vll;
typedef long double ld;
#define int long long
struct point {
  ld x, y;
  point() {}
  point( ld x, ld y ) { this->x = x;
      this->y = y; }
  bool operator<( const point &other ) {
    if ( x == other.x ) return y <
        other.y;
    return x < other.x;
  }
  bool operator==( const point &other ) {
    return ( x == other.x && y ==
        other.y );
  }
  bool operator!=( const point &other ) {
    return ( x != other.x || y !=
        other.y );
  }
  point operator-( const point &other ) {
    return {x - other.x, y - other.y};
  }
};
ld crossProduct( point &a, point &b,
    point &c ) {
  return (b.x - a.x) * (c.y - a.y) -
      (b.y - a.y) * (c.x - a.x);
}
ld crossProduct( point &a, point &b ) {
  return a.x * b.y - a.y * b.x;
}
bool inside( point &a, point &b, point
    &c ) {
  return crossProduct(a, b, c) >= 0;
}
bool intersection( point &a, point &b,
    point &c, point &d, point &ans ) { //
    2 line intersection point
  point ab = b - a;
  point cd = d - c;
  ld det = crossProduct( ab, cd );
  if ( det == 0 ) return false; //
      parallel or collinear
  ld z1 = crossProduct( a, b );
  ld z2 = crossProduct( c, d );
  ans.x = (ld)(z1 * (c.x - d.x) - z2 *
      (a.x - b.x)) / det;
  ans.y = (ld)(z1 * (c.y - d.y) - z2 *
      (a.y - b.y)) / det;
  return true;
}
vector<point> innerhull( vector<point>
    a, vector<point> b ) {
  int n = a.size();
  vector<point> ans = b;
  for ( int i = 0; i < n; i++ ) {
    point x = a[i], y = a[(i + 1) % n];
    vector<point> temp = ans; ans.clear();
    int m = temp.size();
    if ( m == 0 ) break;
    for ( int j = 0; j < m; j++ ) {
      point p = temp[j], q = temp[(j +
          1) % m];
      int pIn = inside(x, y, p);
      int qIn = inside(x, y, q);
      if ( pIn ) ans.push_back(p);
      if ( pIn != qIn ) {
        point cur;
        if ( intersection(x, y, p, q,
            cur) ) {
          ans.push_back(cur);
        }
      }
    }
    if ( !ans.empty() ) {
      vector<point> temp;
      temp.push_back(ans[0]);
      for ( int j = 1; j < ans.size();
          j++ ) {
        if ( ans[j] != ans[j - 1]  ) {
          temp.push_back(ans[j]);
        }
      }
      ans = temp;
    }
  }
  return ans;
}
ld areaOfPolygon( vector<point> a ) {
  if ( a.size() < 3 ) return 0.0;
  a.push_back(a[0]);
  int n = a.size();
  ld ans = 0;
  for ( int i = 0; i < n - 1; i++ ) {
    ans += a[i].x * a[i + 1].y;
    ans -= a[i].y * a[i + 1].x;
  }
  if ( ans < 0 ) ans = -ans;
  return ans / 2.0;
}
}
void senritsu() {
  int n, m; cin >> n >> m;
  vector<point> a(n), b(m);
  for ( int i = 0; i < n; i++ ) {
    cin >> a[i].x >> a[i].y;
  }
  for ( int i = 0; i < m; i++ ) {
    cin >> b[i].x >> b[i].y;
  }
  vector<point> in = innerhull(a, b);
  cout << fixed << setprecision(4) <<
      areaOfPolygon(in) << endl;
}
```