

# 1 CSES

## 1.1 CountingTilings

```

1 //Your task is to count the number of ways you can fill an n×m
  grid using 1×2 and 2×1 tiles.
2 int dp[1005][1<<10] = {};
3 vector<pii>v;
4 void solve(){
5     int n,m;
6     cin>>n>>m;
7     for(int a = 0;a<(1<<n);++a){
8         for(int b = 0;b<(1<<n);++b){
9             bool flag = 1;
10            for(int i = 0;i<n;++i){
11                if(a&(1<<i) and b&(1<<i)){
12                    if(i==n-1 or !(a&(1<<(i+1))) or !(b&(1<<(i+1))))flag
                      = 0;
13                else{
14                    i++;
15                    continue;
16                }
17            }
18            if(!(a&(1<<i)) and !(b&(1<<i)))flag = 0;
19        }
20        if(flag)v.pb({a,b});
21    }
22    dp[0][(1<<n)-1] = 1;
23    for(int i = 1;i<=m;++i){
24        for(auto j:v)dp[i][j.S] = (1ll*dp[i-1][j.F]+dp[i][j.S])%mod
25        ;
26    }
27    cout<<dp[m][(1<<n)-1]<<endl;
28 }
29 signed main(){
30     IOS;
31     solve();
32 }

```

## 1.2 Sequence1

```

1 //0, 1, 2, 9, 44, 265, 1854, 14833, 133496, 1334961
2 #include <bits/stdc++.h>
3
4 #pragma GCC optimize("Ofast,unroll-loops,no-stack-protector,
  fast-math")
5
6 using namespace std;
7
8 #define fastio ios::sync_with_stdio(false), cin.tie(NULL), cout
  .tie(NULL)
9
10 typedef uint64_t ull;
11
12 const int mod = 1e9 + 7, mxN = 1e6 + 1;
13
14 int n;
15 ull dp[mxN];
16
17 int main() {
18     fastio;
19     dp[1] = 0;
20     dp[2] = 1;
21     for(int i = 3; i < mxN; ++i)
22         dp[i] = (i - 1) * (dp[i - 1] + dp[i - 2]) % mod;
23     cin >> n;
24     cout << dp[n] << "\n";
25
26     return 0;
27 }

```

## 1.3 JosephusQueries

```

1 /*
2 Consider a game where there are n children (numbered 1,2,...,n)
  in a circle. During the game, every second child is removed
  from the circle, until there are no children left.
3
4 Your task is to process q queries of the form: "when there are
  n children, who is the kth child that will be removed?"
5
6 */
7 #include <bits/stdc++.h>
8 using namespace std;
9
10 int f(int n, int k) {
11     if(n == 1) {
12         return 0;

```

```

12     }
13     if(k * 2 <= n) {
14         return k * 2 - 1;
15     }
16     int pos = f(n - n / 2, k - n / 2);
17     if(pos == 0) {
18         return (n % 2 == 1 ? n - 1 : 0);
19     }
20     return (pos - n % 2) * 2;
21 }
22
23 int main() {
24     ios::sync_with_stdio(false);
25     cin.tie(0);
26     int tt;
27     cin >> tt;
28     while(tt--) {
29         int n, k;
30         cin >> n >> k;
31         cout << f(n, k) + 1 << "\n";
32     }
33     return 0;
34 }

```

## 1.4 AnotherGame

```

1 /*
2 There are n heaps of coins and two players who move alternately
  . On each move, a player selects some of the nonempty heaps
  and removes one coin from each heap. The player who
  removes the last coin wins the game.
3
4 Your task is to find out who wins if both players play
  optimally.
5
6 */
7 #include <bits/stdc++.h>
8 using namespace std;
9
10 int main() {
11     ios::sync_with_stdio(false);
12     cin.tie(0);
13     int tt;
14     cin >> tt;
15     while(tt--) {
16         int n;
17         cin >> n;
18         bool b = false;
19         for(int i = 0; i < n; ++i) {
20             int x;
21             cin >> x;
22             b = (b || x % 2);
23         }
24         cout << (b ? "first" : "second") << "\n";
25     }
26     return 0;
27 }

```

## 1.5 CountingCoprimePairs

```

1 /*
2 Given a list of n positive integers, your task is to count the
  number of pairs of integers that are coprime (i.e., their
  greatest common divisor is one).
3
4 */
5 #include <bits/stdc++.h>
6 using namespace std;
7
8 const int N = 1e6 + 5;
9
10 int main(int argc, char* argv[]) {
11     ios::sync_with_stdio(false);
12     cin.tie(0);
13     vector<bool> isprime(N + 1, true);
14     isprime[0] = isprime[1] = false;
15     vector<int> prime;
16     vector<int> mu(N + 1);
17     mu[1] = 1;
18     for(int i = 2; i <= N; ++i) {
19         if(isprime[i]) {
20             mu[i] = -1;
21             prime.push_back(i);
22         }
23         for(int j = 0; j < (int) prime.size() && i * prime[j] <= N;
24             ++j) {
25             isprime[i * prime[j]] = false;
26             mu[i * prime[j]] = mu[i] * mu[prime[j]];
27             if(i % prime[j] == 0) {
28                 mu[i * prime[j]] = 0;
29                 break;
30             }
31         }
32     }
33 }

```

```

30 }
31 int n;
32 cin >> n;
33 vector<int> cnt(N + 1);
34 for(int i = 0; i < n; ++i) {
35     int x;
36     cin >> x;
37     cnt[x] += 1;
38 }
39 long long ans = 0;
40 for(int i = 1; i <= N; ++i) {
41     long long s = 0;
42     for(int j = i; j <= N; j += i) {
43         s += cnt[j];
44     }
45     ans += 1LL * mu[i] * s * (s - 1) / 2;
46 }
47 cout << ans << "\n";
48 return 0;
49 }

```

## 1.6 Sequence2

```

1 //1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796
2 #include <bits/stdc++.h>
3
4 #pragma GCC optimize("Ofast,unroll-loops,no-stack-protector,
5     fast-math")
6 using namespace std;
7
8 #define fastio ios::sync_with_stdio(false), cin.tie(NULL), cout
9     .tie(NULL)
10 typedef uint64_t ull;
11
12 ull FastPower(ull a, ull b, ull m) {
13     a %= m;
14     ull ans = 1;
15     while(b) {
16         if(b & 1)
17             ans = ans * a % m;
18         a = a * a % m;
19         b >>= 1;
20     }
21     return ans;
22 }
23
24 const int mod = 1e9 + 7, mxN = 2e6 + 1;
25
26 ull n, f[mxN];
27
28 int main() {
29     fastio;
30     f[0] = 1;
31     for(int i = 1; i < mxN; ++i)
32         f[i] = f[i - 1] * i % mod;
33     cin >> n;
34     if(n & 1) {
35         cout << "0\n";
36         return 0;
37     }
38     n >>= 1;
39     ull temp = FastPower(f[n], mod - 2, mod);
40     cout << f[n << 1] * temp % mod * temp % mod * FastPower(n +
41         1, mod - 2, mod) % mod << "\n";
42
43     return 0;
44 }

```

## 1.7 LongestPalindrome

```

1 /*
2 Problem Name: Longest Palindrome
3 Problem Link: https://cses.fi/problemset/task/1111
4 Author: Sachin Srivastava (mrsac7)
5 */
6 #include<bits/stdc++.h>
7 using namespace std;
8
9 template<typename... T>
10 #define error(args...) { string _s = #args; replace(_s.begin(),
11     _s.end(), ',', ' '); stringstream _ss(_s);
12     istream_iterator<string> _it(_ss); err(_it, args); }
13 void err(istream_iterator<string> it) {}
14 template<typename T, typename... Args>
15 void err(istream_iterator<string> it, T a, Args... args) {cerr
16     << *it << "==" << a << ", "; err(++it, args...);}
17
18 #define int long long
19 #define F first

```

```

17 #define S second
18
19 const long long inf = 1LL<<62;
20 const int md = 1000000007;
21
22 void solve(){
23     string s; cin>>s;
24     int n = s.size();
25     int dp[n][2] = {0};
26     int x1 = 0, y1 = -1;
27     int x2 = 0, y2 = -1;
28     int mx = 0, ans = 0;
29     for (int i = 0; i < n; i++) {
30         int k = 0;
31         if (i>y1) k = 1;
32         else k = min(dp[x1+y1-i][0], y1-i+1);
33         while (0<=i-k && i+k<n && s[i-k] == s[i+k]) k++;
34         dp[i][0] = k--;
35         if (i+k>y1) x1 = i-k, y1 = i+k;
36         if (2*dp[i][0] - 1 > mx) ans = i-k, mx = 2*dp[i][0] -
37             1;
38         k = 0;
39         if (i<=y2) k = min(dp[x2+y2-i+1][1],y2-i+1);
40         while(0<=i-k-1 && i+k<n && s[i-k-1] == s[i+k]) k++;
41         dp[i][1] = k--;
42         if (i+k>y2) x2 = i-k-1, y2 = i+k;
43         if (2*dp[i][1] > mx) ans = i-k-1, mx = 2*dp[i][1];
44     }
45     cout<<s.substr(ans,mx);
46 }
47 signed main(){
48     ios_base::sync_with_stdio(false);cin.tie(0);cout.tie(0);
49     #ifdef LOCAL
50     freopen("input.txt", "r", stdin);
51     freopen("output.txt", "w", stdout);
52     #endif
53     int t=1;
54     //cin>>t;
55     for (int i = 1; i <= t; i++) {
56         solve();
57         cout<<'\n';
58     }
59 }

```

## 1.8 DistinctSubstrings

```

1 //Count the number of distinct substrings that appear in a
2 //string.
3 //abaa => 8 : Explanation: the substrings are a, b, aa, ab, ba,
4 //aba, baa and abaa.
5 #include <bits/extc++.h>
6 #include <bits/stdc++.h>
7 #pragma gcc optimize("ofast, unroll-loops, no-stack-protector,
8     fast-math")
9 #define IOS ios::sync_with_stdio(0),cin.tie(0),cout.tie(0)
10 #define int long long
11 #define double long double
12 #define pb push_back
13 #define sz(x) (int)(x).size()
14 #define all(v) begin(v),end(v)
15 #define debug(x) cerr<<#x<<" = "<<x<<'\n'
16 #define LINE cout<<"\n-----\n"
17 #define endl '\n'
18 #define VI vector<int>
19 #define F first
20 #define S second
21 #define MP(a,b) make_pair(a,b)
22 #define rep(i,m,n) for(int i = m;i<=n;++i)
23 #define res(i,m,n) for(int i = m;i>n;--i)
24 #define gcd(a,b) __gcd(a,b)
25 #define lcm(a,b) a*b/gcd(a,b)
26 #define Case() int _;cin>>_;for(int Case = 1;Case<=_;++Case)
27 #define pii pair<int,int>
28 using namespace __gnu_cxx;
29 using namespace __gnu_pbds;
30 using namespace std;
31 template <typename K, typename cmp = less<K>, typename T =
32     thin_heap_tag> using _heap = __gnu_pbds::priority_queue<K,
33     cmp, T>;
34 template <typename K, typename M = null_type> using _hash =
35     gp_hash_table<K, M>;
36 const int N = 1e6+5,L = 20,mod = 1e9+7;
37 const long long inf = 2e18+5;
38 const double eps = 1e-7,pi = acos(-1);
39 mt19937 mt(std::chrono::system_clock::now().time_since_epoch().
40     count());
41 struct suffix_array{
42     int n;
43     vector<int>SA,Rank,LCP;
44     void counting_sort(vector<int>&v,auto getkey){
45         int n = 0;
46         for(auto i:v)n = max(n,getkey(i)+1);
47         vector<int>bucket(n),ans(v.size());
48         for(auto i:v)++bucket[getkey(i)];

```

```

42 partial_sum(begin(bucket),end(bucket),begin(bucket));
43 for(auto ite = v.rbegin();ite!=v.rend();++ite)ans[--bucket[
    getkey(*ite)]] = move(*ite);
44 v.swap(ans);
45 return;
46 }
47 suffix_array(string s):n(s.size()){
48 SA.resize(n),Rank.resize(n),LCP.resize(n);
49 for(int i = 0;i<n;++i)SA[i] = i;
50 sort(SA.begin(),SA.end(),[&](int a,int b){
51     return s[a]<s[b];
52 });
53 for(int i = 0;i<n;++i){
54     Rank[SA[i]] = (i?Rank[SA[i-1]]+(s[SA[i]]!=s[SA[i-1]]):SA
        [0]);
55 }
56 for(int k = 0;(1<k)<=n;++k){
57     vector<int>idx;
58     for(int i = n-(1<k);i<n;++i)idx.push_back(i);
59     for(auto i:SA)if(i>=(1<k))idx.push_back(i-(1<k));
60     counting_sort(idx,[&](int a){return Rank[a]});
61     SA.swap(idx);
62     vector<int>new_rank(n);
63     new_rank[SA[0]] = 0;
64     for(int i = 1;i<n;++i){
65         auto cmp = [&](int a,int b){
66             return Rank[a]!=Rank[b] or a+(1<k)>=n or Rank[a+(1<
                k)]!=Rank[b+(1<k)];
67         };
68         new_rank[SA[i]] = new_rank[SA[i-1]]+cmp(SA[i-1],SA[i]);
69     }
70     Rank.swap(new_rank);
71 }
72 for(int i = 0,k = 0;i<n;++i){
73     if(Rank[i]==0)continue;
74     if(k)--k;
75     while(i+k<n and SA[Rank[i]-1]+k<n and s[i+k]==s[SA[Rank[i]
        ]-1]+k])++k;
76     LCP[Rank[i]] = k;
77 }
78 };
79 void solve(){
80     string s;
81     getline(cin,s);
82     suffix_array sa(s);
83     int n = s.size();
84     int ans = n*(n+1)/2;
85     for(int i = 1;i<n;++i)ans-=sa.LCP[i];
86     cout<<ans<<endl;
87 }
88 signed main(){
89     IOS;
90     solve();
91 }
92 }

```

## 1.9 BracketSequencesII

```

1 //Your task is to calculate the number of valid bracket
    sequences of length n when a prefix of the sequence is
    given.
2 int main() {
3     ios::sync_with_stdio(false);
4     cin.tie(0);
5     int n;
6     cin >> n;
7     if(n % 2 == 1) {
8         cout << "0\n";
9         return 0;
10    }
11    string s;
12    cin >> s;
13    int m = (int) s.size();
14    int delta = 0;
15    int left = 0;
16    for(char& c : s) {
17        delta += (c == '(' ? +1 : -1);
18        left += (c == '(');
19        if(delta < 0) {
20            cout << "0\n";
21            return 0;
22        }
23    }
24    left = n / 2 - left;
25    mint ans = C(n - m, left) - C(n - m, left + delta + 1);
26    cout << ans << "\n";
27    return 0;
28 }

```

## 1.10 CountingNumbers

```

1 //Your task is to count the number of integers between a and b
    where no two adjacent digits are the same.
2 #include <bits/extc++.h>
3 #include <bits/stdc++.h>
4 #pragma gcc optimize("ofast, unroll-loops, no-stack-protector,
    fast-math")
5 #define IOS ios::sync_with_stdio(0),cin.tie(0),cout.tie(0)
6 #define int long long
7 #define double long double
8 #define pb push_back
9 #define sz(x) (int)(x).size()
10 #define all(v) begin(v),end(v)
11 #define debug(x) cerr<<#x<<" = "<<x<<'\n'
12 #define LINE cout<<"\n-----\n"
13 #define endl '\n'
14 #define VI vector<int>
15 #define F first
16 #define S second
17 #define MP(a,b) make_pair(a,b)
18 #define rep(i,m,n) for(int i = m;i<=n;++i)
19 #define res(i,m,n) for(int i = m;i>=n;--i)
20 #define gcd(a,b) __gcd(a,b)
21 #define lcm(a,b) a*b/gcd(a,b)
22 #define Case() int _;cin>>_;for(int Case = 1;Case<=_;++Case)
23 #define pii pair<int,int>
24 #define lowbit(x) (x&(-x))
25 using namespace __gnu_cxx;
26 using namespace __gnu_pbds;
27 using namespace std;
28 template <typename K, typename cmp = less<K>, typename T =
    thin_heap_tag> using _heap = __gnu_pbds::priority_queue<K,
    cmp, T>;
29 template <typename K, typename M = null_type> using _hash =
    gp_hash_table<K, M>;
30 const int N = 1e6+5,L = 20,mod = 1e9+7,inf = 2e9+5;
31 const double eps = 1e-7,pi = acos(-1);
32 mt19937 mt(std::chrono::system_clock::now().time_since_epoch().
    count());
33 int cnt(int x){
34     if(x<0)return 0;
35     string s = std::to_string(x);
36     reverse(all(s));
37     int n = s.size(),ans = 0;
38     int dp[n][2][10] = {};
39     for(int i = 0;i<10;++i){
40         dp[0][(i>s[0]-'0')][i]++;
41     }
42     for(int i = 1;i<n;++i){
43         for(int j = 0;j<2;++j){
44             for(int last = 0;last<10;++last){
45                 for(int add = 0;add<10;++add){
46                     if(add==last)continue;
47                     bool flag = (add>s[i]-'0') or (add==(s[i]-'0') and
                        j);
48                     dp[i][flag][add]+=dp[i-1][j][last];
49                 }
50             }
51         }
52     }
53     for(int i = 0;i<n-1;++i){
54         for(int j = 0;j<2;++j){
55             for(int k = 1;k<10;++k){
56                 ans+=dp[i][j][k];
57             }
58         }
59     }
60     for(int i = 1;i<10;++i){
61         ans+=dp[n-1][0][i];
62     }
63     return ans+1;
64 }
65 void solve(){
66     int a,b;
67     cin>>a>>b;
68     cout<<cnt(b)-cnt(a-1)<<endl;
69 }
70 signed main(){
71     IOS;
72     solve();
73 }

```

## 1.11 WordCombinations

```

1 //You are given a string of length n and a dictionary
    containing k words. In how many ways can you create the
    string using the words?
2 #include <bits/stdc++.h>
3
4 #pragma GCC optimize("Ofast,unroll-loops,no-stack-protector,
    fast-math")
5
6 using namespace std;
7

```

```

8 #define fastio ios::sync_with_stdio(false), cin.tie(NULL), cout
   .tie(NULL)
9
10 typedef int64_t ll;
11
12 const ll A = 912345693, B = 987654327, mxN = 5005, mod = 1e9 +
   7;
13
14 int n;
15 string s;
16 ll h[mxN], p[mxN], dp[mxN];
17 vector<ll> num[mxN];
18
19 ll Get(int a, int b) {
20     return ((h[b] - h[a - 1] * p[b - a + 1]) % B + B) % B;
21 }
22
23 int main() {
24     fastio;
25     p[0] = 1;
26     for(int i = 1; i < mxN; ++i)
27         p[i] = p[i - 1] * A % B;
28     cin >> s >> n;
29     s = " " + s;
30     h[0] = 0;
31     for(int i = 1; i <= s.size(); ++i)
32         h[i] = (A * h[i - 1] + s[i]) % B;
33     for(int i = 0; i < n; ++i) {
34         string temp;
35         cin >> temp;
36         ll val = 0;
37         for(char c : temp)
38             val = (val * A + c) % B;
39         num[temp.size()].push_back(val);
40     }
41     n = s.size() - 1;
42     dp[0] = 1;
43     for(int i = 0; i < n; ++i) {
44         for(int j = 1; i + j <= n; ++j) {
45             ll val = Get(i + 1, i + j);
46             for(ll x : num[j])
47                 if(val == x)
48                     dp[i + j] = (dp[i + j] + dp[i]) % mod;
49         }
50     }
51     cout << dp[n] << "\n";
52
53     return 0;
54 }

```

## 1.12 SumOfDivisors

```

1 /*
2  Let  $\sigma(n)$  denote the sum of divisors of an integer  $n$ . For
   example,  $\sigma(12)=1+2+3+4+6+12=28$ .
3
4  Your task is to calculate the sum  $\sum_{i=1}^n \sigma(i)$  modulo
    $10^9+7$ .
5 */
6 #include <bits/stdc++.h>
7 #define int long long
8 using namespace std;
9 const int mod = 1e9 + 7;
10 constexpr long long Pow(long long x, long long n, int m) {
11     if(m == 1) return 0;
12     unsigned int _m = (unsigned int)(m);
13     unsigned long long r = 1;
14     x %= m;
15     if(x < 0) x += m;
16     unsigned long long y = x;
17     while(n) {
18         if(n & 1) r = (r * y) % _m;
19         y = (y * y) % _m;
20         n >>= 1;
21     }
22     return r;
23 }
24 signed main(){
25     int n;
26     cin >> n;
27     int ans = 0;
28     for(int l = 1, r = n / (n / l); l <= n; l = r + 1){
29         r = n / (n / l);
30         ans += (((((((1 + r) % mod) * ((r - 1 + 1) % mod)) % mod) *
31             Pow(2, mod - 2, mod)) % mod) * ((n / l) % mod)) % mod;
32     }
33     cout << ans << endl;
34 }

```

## 1.13 FindingPatterns

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 template<int ALPHABET = 26, char MIN_CHAR = 'a'>
5 class suffix_automaton {
6 public:
7     struct Node {
8         int len;
9         int suffLink;
10        int go[ALPHABET] = {};
11
12        Node() : Node(0, -1) {}
13        Node(int a, int b) : len(a), suffLink(b) {}
14    };
15
16    suffix_automaton() : suffix_automaton(string(0, ' ')) {}
17    suffix_automaton(const string& s) {
18        SA.emplace_back();
19        last = 0;
20        for(char c : s) {
21            add(c - MIN_CHAR);
22        }
23    }
24
25    void add(int c) {
26        int u = newNode();
27        SA[u].len = SA[last].len + 1;
28        int p = last;
29        while(p != -1 && SA[p].go[c] == 0) {
30            SA[p].go[c] = u;
31            p = SA[p].suffLink;
32        }
33        if(p == -1) {
34            SA[u].suffLink = 0;
35            last = u;
36            return;
37        }
38        int q = SA[p].go[c];
39        if(SA[p].len + 1 == SA[q].len) {
40            SA[u].suffLink = q;
41            last = u;
42            return;
43        }
44        int x = newNode();
45        SA[x] = SA[q];
46        SA[x].len = SA[p].len + 1;
47        SA[q].suffLink = SA[u].suffLink = x;
48        while(p != -1 && SA[p].go[c] == q) {
49            SA[p].go[c] = x;
50            p = SA[p].suffLink;
51        }
52        last = u;
53        return;
54    }
55
56    bool match(const string& s) {
57        int u = 0;
58        for(char c : s) {
59            int v = c - MIN_CHAR;
60            if(SA[u].go[v] == 0) {
61                return false;
62            }
63            u = SA[u].go[v];
64        }
65        return true;
66    }
67
68 private:
69     vector<Node> SA;
70     int last;
71
72     inline int newNode() {
73         SA.emplace_back();
74         return (int) SA.size() - 1;
75     }
76 };
77
78 int main() {
79     ios::sync_with_stdio(false);
80     cin.tie(0);
81     string s;
82     cin >> s;
83     suffix_automaton SA(s);
84     int q;
85     cin >> q;
86     while(q--) {
87         string t;
88         cin >> t;
89         cout << (SA.match(t) ? "YES" : "NO") << "\n";
90     }
91     return 0;
92 }

```

## 1.14 RemovalGame

```

1  /*
2  There is a list of n numbers and two players who move
   alternately. On each move, a player removes either the
   first or last number from the list, and their score
   increases by that number. Both players try to maximize
   their scores.
3
4  What is the maximum possible score for the first player when
   both players play optimally?
5
6  */
7  #include <bits/stdc++.h>
8  using namespace std;
9
10 int main() {
11     ios::sync_with_stdio(false);
12     cin.tie(0);
13     int n;
14     cin >> n;
15     vector<long long> a(n), pref(n + 1);
16     for(int i = 0; i < n; ++i) {
17         cin >> a[i];
18         pref[i + 1] = pref[i] + a[i];
19     }
20     vector<vector<long long>> dp(n, vector<long long>(n));
21     for(int i = 0; i < n; ++i) {
22         dp[i][i] = a[i];
23     }
24     for(int len = 2; len <= n; ++len) {
25         for(int i = 0; i + len - 1 < n; ++i) {
26             int j = i + len - 1;
27             dp[i][j] = pref[j + 1] - pref[i] - min(dp[i + 1][j], dp[i][j - 1]);
28         }
29     }
30     cout << dp[0][n - 1] << "\n";
31     return 0;
32 }

```

## 1.15 MinimalRotation

```

1  #include <bits/stdc++.h>
2  using namespace std;
3
4  template<int ALPHABET = 26, char MIN_CHAR = 'a'>
5  class suffix_automaton {
6  public:
7      struct Node {
8          int len;
9          int suffLink;
10         int go[ALPHABET] = {};
11
12         Node() : Node(0, -1) {}
13         Node(int a, int b) : len(a), suffLink(b) {}
14     };
15
16     suffix_automaton() : suffix_automaton(string(0, ' ')) {}
17     suffix_automaton(const string& s) {
18         SA.emplace_back();
19         last = 0;
20         for(char c : s) {
21             add(c - MIN_CHAR);
22         }
23     }
24
25     void add(int c) {
26         int u = newNode();
27         SA[u].len = SA[last].len + 1;
28         int p = last;
29         while(p != -1 && SA[p].go[c] == 0) {
30             SA[p].go[c] = u;
31             p = SA[p].suffLink;
32         }
33         if(p == -1) {
34             SA[u].suffLink = 0;
35             last = u;
36             return;
37         }
38         int q = SA[p].go[c];
39         if(SA[p].len + 1 == SA[q].len) {
40             SA[u].suffLink = q;
41             last = u;
42             return;
43         }
44         int x = newNode();
45         SA[x] = SA[q];
46         SA[x].len = SA[p].len + 1;
47         SA[q].suffLink = SA[u].suffLink = x;
48         while(p != -1 && SA[p].go[c] == q) {
49             SA[p].go[c] = x;
50             p = SA[p].suffLink;
51         }
52     }
53 }

```

```

52     last = u;
53     return;
54 }
55
56 // private:
57 vector<Node> SA;
58 int last;
59
60 inline int newNode() {
61     SA.emplace_back();
62     return (int) SA.size() - 1;
63 }
64 };
65
66 int main() {
67     ios::sync_with_stdio(false);
68     cin.tie(0);
69     string s;
70     cin >> s;
71     int n = (int) s.size();
72     suffix_automaton SA(s + s);
73     int p = 0;
74     for(int i = 0; i < n; ++i) {
75         for(int c = 0; c < 26; ++c) {
76             if(SA.SA[p].go[c]) {
77                 cout << char('a' + c);
78                 p = SA.SA[p].go[c];
79                 break;
80             }
81         }
82     }
83     cout << "\n";
84     return 0;
85 }

```

## 1.16 Dice

```

1  //1, 2, 4, 8, 16, 32, 63, 125, 248, 492
2  #include <bits/stdc++.h>
3  using namespace std;
4
5  typedef uint64_t ull;
6  const int mod = 1e9 + 7;
7
8  struct Matrix {
9      ull M[6][6];
10     Matrix() {
11         memset(M, 0, sizeof(M));
12     }
13     Matrix operator*(const Matrix& other) {
14         Matrix ans;
15         for(int i = 0; i < 6; ++i)
16             for(int j = 0; j < 6; ++j)
17                 for(int k = 0; k < 6; ++k)
18                     ans.M[i][j] = (ans.M[i][j] + M[i][k] * other.M[k][j])
19                                     % mod;
20         return ans;
21     };
22
23     Matrix FastPower(Matrix a, ull b) {
24         Matrix ans;
25         for(int i = 0; i < 6; ++i)
26             ans.M[i][i] = 1;
27         while(b) {
28             if(b & 1)
29                 ans = ans * a;
30             a = a * a;
31             b >>= 1;
32         }
33         return ans;
34     }
35
36     int main() {
37         Matrix A;
38         for(int i = 0; i < 6; ++i)
39             A.M[0][i] = 1;
40         for(int i = 1; i < 6; ++i)
41             A.M[i][i - 1] = 1;
42         ull n;
43         cin >> n;
44         cout << FastPower(A, n).M[0][0] << "\n";
45
46         return 0;
47     }

```

## 2 Data-Structure

### 2.1 Treap

```

1 template<class S,
2     S (*node_pull)(S, S),
3     S (*node_init)(S),
4     class T,
5     S (*mapping)(S, T),
6     T (*tag_pull)(T, T),
7     T (*tag_init)()>
8 struct Treap{
9     struct node{
10         node *l = NULL,*r = NULL,*p = NULL;
11         const int pri = rand();
12         int sz = 1;
13         S info;
14         T tag = tag_init();
15         bool rev;
16         node(S k) : info(k){}
17         ~node(){
18             for(auto &i:{l,r})
19                 delete i;
20         }
21         void all_apply(T t,bool is_rev){
22             if(is_rev){
23                 swap(l,r);
24                 rev^=1;
25             }
26             info = mapping(info, t);
27             tag = tag_pull(tag, t);
28         }
29         void push(){
30             for(auto &i:{l,r})
31                 if(i)i->all_apply(tag, rev);
32             tag = tag_init();
33             rev = 0;
34         }
35         void pull(){
36             sz = 1,info = node_init(info);
37             for(auto &i:{l,r}){
38                 if(i){
39                     sz+=i->sz,i->p = this;
40                     info = node_pull(info,i->info);
41                 }
42             }
43         }
44     };
45     node *root = NULL;
46     int size(node *a){
47         return a?a->sz:0;
48     }
49     int size(){
50         return size(root);
51     }
52     node *merge(node *a,node *b){
53         if(!a or !b)return a?:b;
54         if(a->pri>b->pri){
55             a->push();
56             a->r = merge(a->r,b);
57             a->r->p = a;
58             a->pull();
59             return a;
60         }
61         else{
62             b->push();
63             b->l = merge(a,b->l);
64             b->l->p = b;
65             b->pull();
66             return b;
67         }
68     }
69     void split(node *t, long long k, node *&a, node *&b, const
70         bool &bst){
71         if(!t){a = b = NULL;return;}
72         t->push();
73         if((bst==0 and size(t->l)+1<=k) or (bst==1 and t->info.key
74             <=k)){
75             a = t;
76             split(t->r, (bst ? k : k - size(t->l) - 1 ), a->r, b,
77                 bst);
78             if(b)b->p = NULL;
79             a->pull();
80         }
81         else{
82             b = t;
83             split(t->l, k, a, b->l, bst);
84             if(a)a->p = NULL;
85             b->pull();
86         }
87     }
88     node *insert(long long idx, S x,bool bst = 0){
89         node *a,*b;
90         split(root, idx, a, b, bst);
91         node *tmp = new node(x);
92         root = merge(a, merge(tmp, b));
93         return tmp;
94     }
95     void erase(long long l,long long r,bool bst = 0){
96         node *a,*b,*c;
97         split(root, (bst? l-1 : l), a, b, bst);
98         split(b, (bst? r : r - l + 1), b, c, bst);
99         delete b;
100         root = merge(a,c);
101     }
102     S operator [](int x){
103         node *a, *b, *c;
104         split(root, x, a, b, 0);
105         split(b, 1, b, c, 0);
106         assert(b!=NULL);
107         S ans = b->info;
108         root = merge(a, merge(b, c));
109         return ans;
110     }
111     int rank(long long k){
112         node *a, *b;
113         split(root, k - 1, a, b, 1);
114         int ans = size(a);
115         root = merge(a, b);
116         return ans;
117     }
118     S* find_next(long long k){
119         node *a, *b, *c;
120         split(root, k - 1, a, b, 1);
121         split(b, 1, b, c, 0);
122         S* ans = NULL;
123         if(b)ans = &b->info;
124         root = merge(a, merge(b, c));
125         return ans;
126     }
127     S* find_prev(long long k){
128         node *a, *b, *c;
129         split(root, k , a, b, 1);
130         split(a, size(a) - 1, a, c, 0);
131         S* ans = NULL;
132         if(c)ans = &c->info;
133         root = merge(merge(a, c), b);
134         return ans;
135     }
136     void update(long long l,long long r,T t,bool bst = 0){
137         node *a, *b, *c;
138         split(root, (bst? l - 1 : l), a, b, bst);
139         split(b, (bst? r : r - l + 1), b, c, bst);
140         if(b)b->all_apply(t, 0);
141         root = merge(a, merge(b, c));
142     }
143     void reverse(long long l,long long r,bool bst = 0){
144         node *a, *b, *c;
145         split(root, (bst? l - 1 : l), a, b, bst);
146         split(b, (bst? r : r - l + 1), b, c, bst);
147         if(b)b->all_apply(tag_init(), 1);
148         root = merge(a, merge(b, c));
149     }
150     S query(long long l,long long r,bool bst = 0){
151         node *a, *b, *c;
152         split(root, (bst? l - 1 : l), a, b, bst);
153         split(b, (bst? r : r - l + 1), b, c, bst);
154         S ans;
155         if(b)ans = b->info;
156         root = merge(a, merge(b, c));
157         return ans;
158     }
159 }

```

```

95     split(b, (bst? r : r - l + 1), b, c, bst);
96     delete b;
97     root = merge(a,c);
98 }
99 S operator [](int x){
100     node *a, *b, *c;
101     split(root, x, a, b, 0);
102     split(b, 1, b, c, 0);
103     assert(b!=NULL);
104     S ans = b->info;
105     root = merge(a, merge(b, c));
106     return ans;
107 }
108 int rank(long long k){
109     node *a, *b;
110     split(root, k - 1, a, b, 1);
111     int ans = size(a);
112     root = merge(a, b);
113     return ans;
114 }
115 S* find_next(long long k){
116     node *a, *b, *c;
117     split(root, k - 1, a, b, 1);
118     split(b, 1, b, c, 0);
119     S* ans = NULL;
120     if(b)ans = &b->info;
121     root = merge(a, merge(b, c));
122     return ans;
123 }
124 S* find_prev(long long k){
125     node *a, *b, *c;
126     split(root, k , a, b, 1);
127     split(a, size(a) - 1, a, c, 0);
128     S* ans = NULL;
129     if(c)ans = &c->info;
130     root = merge(merge(a, c), b);
131     return ans;
132 }
133 void update(long long l,long long r,T t,bool bst = 0){
134     node *a, *b, *c;
135     split(root, (bst? l - 1 : l), a, b, bst);
136     split(b, (bst? r : r - l + 1), b, c, bst);
137     if(b)b->all_apply(t, 0);
138     root = merge(a, merge(b, c));
139 }
140 void reverse(long long l,long long r,bool bst = 0){
141     node *a, *b, *c;
142     split(root, (bst? l - 1 : l), a, b, bst);
143     split(b, (bst? r : r - l + 1), b, c, bst);
144     if(b)b->all_apply(tag_init(), 1);
145     root = merge(a, merge(b, c));
146 }
147 S query(long long l,long long r,bool bst = 0){
148     node *a, *b, *c;
149     split(root, (bst? l - 1 : l), a, b, bst);
150     split(b, (bst? r : r - l + 1), b, c, bst);
151     S ans;
152     if(b)ans = b->info;
153     root = merge(a, merge(b, c));
154     return ans;
155 }
156 }

```

## 2.2 Segtree

```

1 template<class S,
2     S (*node_pull)(S, S),
3     S (*node_init)(),
4     class T,
5     S (*mapping)(S, T),
6     T (*tag_pull)(T, T),
7     T (*tag_init)()>
8 struct segment_tree{
9     struct node{
10         S seg;
11         T tag = tag_init();
12         int l,r;
13         node(S _seg = node_init(),int _l = -1,int _r = -1) : seg(
14             _seg), l(_l), r(_r){}
15         friend node operator +(const node &lhs,const node &rhs){
16             if(lhs.l==1)return rhs;
17             if(rhs.l==1)return lhs;
18             return node(node_pull(lhs.seg,rhs.seg),lhs.l,rhs.r);
19         }
20     };
21     vector<node>arr;
22     void all_apply(int idx,T t){
23         arr[idx].seg = mapping(arr[idx].seg, t);
24         arr[idx].tag = tag_pull(arr[idx].tag, t);
25     }
26     void push(int idx){
27         all_apply(idx<<1, arr[idx].tag);
28         all_apply(idx<<1+1, arr[idx].tag);
29         arr[idx].tag = tag_init();
30     }

```



```

29 }
30 inline void build(const vector<S> &v, const int &l, const int &
    r, int idx = 1){
31     if(idx==1)arr.resize((r-l+1)<<2);
32     if(l==r){
33         arr[idx].seg = v[l];
34         arr[idx].tag = tag_init();
35         arr[idx].l = arr[idx].r = l;
36         return;
37     }
38     int m = (l+r)>>1;
39     build(v, l, m, idx<<1);
40     build(v, m+1, r, idx<<1|1);
41     arr[idx] = arr[idx<<1]+arr[idx<<1|1];
42 }
43 inline void update(const int &q1, const int &qr, T t, int idx =
    1){
44     assert(q1<=qr);
45     if(q1<=arr[idx].l and arr[idx].r<=qr){
46         all_apply(idx, t);
47         return;
48     }
49     push(idx);
50     int m = (arr[idx].l+arr[idx].r)>>1;
51     if(q1<=m)update(q1, qr, t, idx<<1);
52     if(qr>m)update(q1, qr, t, idx<<1|1);
53     arr[idx] = arr[idx<<1]+arr[idx<<1|1];
54 }
55 inline S query(const int &q1, const int &qr, int idx = 1){
56     assert(q1<=qr);
57     if(q1<=arr[idx].l and arr[idx].r<=qr){
58         return arr[idx].seg;
59     }
60     push(idx);
61     int m = (arr[idx].l+arr[idx].r)>>1;
62     S ans = node_init(), lhs = node_init(), rhs = node_init();
63     if(q1<=m)lhs = query(q1, qr, idx<<1);
64     if(qr>m)rhs = query(q1, qr, idx<<1|1);
65     ans = node_pull(lhs, rhs);
66     return ans;
67 }
68 };

```

## 2.3 BinaryTrie

```

1 template<class T>
2 struct binary_trie {
3 public:
4     binary_trie() {
5         new_node();
6     }
7
8     void clear() {
9         trie.clear();
10        new_node();
11    }
12
13    void insert(T x) {
14        for(int i = B - 1, p = 0; i >= 0; i--) {
15            int y = x >> i & 1;
16            if(trie[p].go[y] == 0) {
17                trie[p].go[y] = new_node();
18            }
19            p = trie[p].go[y];
20            trie[p].cnt += 1;
21        }
22    }
23
24    void erase(T x) {
25        for(int i = B - 1, p = 0; i >= 0; i--) {
26            p = trie[p].go[x >> i & 1];
27            trie[p].cnt -= 1;
28        }
29    }
30
31    bool contains(T x) {
32        for(int i = B - 1, p = 0; i >= 0; i--) {
33            p = trie[p].go[x >> i & 1];
34            if(trie[p].cnt == 0) {
35                return false;
36            }
37        }
38        return true;
39    }
40
41    T get_min() {
42        return get_xor_min(0);
43    }
44
45    T get_max() {
46        return get_xor_max(0);
47    }
48
49    T get_xor_min(T x) {

```

```

50     T ans = 0;
51     for(int i = B - 1, p = 0; i >= 0; i--) {
52         int y = x >> i & 1;
53         int z = trie[p].go[y];
54         if(z > 0 && trie[z].cnt > 0) {
55             p = z;
56         } else {
57             ans |= T(1) << i;
58             p = trie[p].go[y ^ 1];
59         }
60     }
61     return ans;
62 }
63
64 T get_xor_max(T x) {
65     T ans = 0;
66     for(int i = B - 1, p = 0; i >= 0; i--) {
67         int y = x >> i & 1;
68         int z = trie[p].go[y ^ 1];
69         if(z > 0 && trie[z].cnt > 0) {
70             ans |= T(1) << i;
71             p = z;
72         } else {
73             p = trie[p].go[y];
74         }
75     }
76     return ans;
77 }
78
79 private:
80     static constexpr int B = sizeof(T) * 8;
81
82     struct Node {
83         std::array<int, 2> go = {};
84         int cnt = 0;
85     };
86
87     std::vector<Node> trie;
88
89     int new_node() {
90         trie.emplace_back();
91         return (int) trie.size() - 1;
92     }
93 };

```

## 2.4 DsuUndo

```

1 struct dsu_undo{
2     vector<int> sz, p;
3     int comps;
4     dsu_undo(int n){
5         sz.assign(n+5, 1);
6         p.resize(n+5);
7         for(int i = 1; i<=n; ++i) p[i] = i;
8         comps = n;
9     }
10    vector<pair<int, int>> opt;
11    int Find(int x){
12        return x==p[x]?x:Find(p[x]);
13    }
14    bool Union(int a, int b){
15        int pa = Find(a), pb = Find(b);
16        if(pa==pb) return 0;
17        if(sz[pa]<sz[pb]) swap(pa, pb);
18        sz[pa] += sz[pb];
19        p[pb] = pa;
20        opt.push_back({pa, pb});
21        comps--;
22        return 1;
23    }
24    void undo(){
25        auto [pa, pb] = opt.back();
26        opt.pop_back();
27        p[pb] = pb;
28        sz[pa] -= sz[pb];
29        comps++;
30    }
31 };

```

## 2.5 CDQ

```

1 void CDQ(int l, int r) {
2     if(l + 1 == r) return;
3     int mid = (l + r) / 2;
4     CDQ(l, mid), CDQ(mid, r);
5     int i = l;
6     for(int j = mid; j < r; j++) {
7         const Q& q = qry[j];
8         while(i < mid && qry[i].x >= q.x) {
9             if(qry[i].id == -1) fenw.add(qry[i].y, qry[i].w);
10            i++;

```

```

11     }
12     if(q.id >= 0) ans[q.id] += q.w * fenw.sum(q.y, sz - 1);
13 }
14 for(int p = l; p < i; p++) if(qry[p].id == -1) fenw.add(qry[p]
15     ].y, -qry[p].w);
16 inplace_merge(qry.begin() + l, qry.begin() + mid, qry.begin()
17     + r, [](const Q& a, const Q& b) {
18     return a.x > b.x;
19 });
20 }

```

## 2.6 MaximumXorSum

```

1  /*
2   给定一个非负整数序列 {a}{a}，初始长度为 NN。
3
4   有 MM 个操作，有以下两种操作类型：
5
6   A x：添加操作，表示在序列末尾添加一个数 xx，序列的长度 NN
7   加 11。
8   Q l r x：询问操作，你需要找到一个位置 pp，满足  $l \leq p \leq r$ ，
9   使得： $a[p] \oplus a[p+1] \oplus \dots \oplus a[N] \oplus xa[p] \oplus a[p+1] \oplus \dots \oplus a[N] \oplus x$  最
10  大，输出最大值。
11
12  */
13 #include <algorithm>
14 #include <cstdio>
15 #include <cstring>
16 using namespace std;
17 const int maxn = 600010;
18 int n, q, a[maxn], s[maxn], l, r, x;
19 char op;
20
21 struct Trie {
22     int cnt, rt[maxn], ch[maxn * 33][2], val[maxn * 33];
23
24     void insert(int o, int lst, int v) {
25         for (int i = 28; i >= 0; i--) {
26             val[o] = val[lst] + 1; // 在原版的基础上更新
27             if ((v & (1 << i)) == 0) {
28                 if (!ch[o][0]) ch[o][0] = ++cnt;
29                 ch[o][1] = ch[lst][1];
30                 o = ch[o][0];
31                 lst = ch[lst][0];
32             } else {
33                 if (!ch[o][1]) ch[o][1] = ++cnt;
34                 ch[o][0] = ch[lst][0];
35                 o = ch[o][1];
36                 lst = ch[lst][1];
37             }
38             val[o] = val[lst] + 1;
39             // printf("%d\n", o);
40         }
41     }
42
43     int query(int o1, int o2, int v) {
44         int ret = 0;
45         for (int i = 28; i >= 0; i--) {
46             // printf("%d %d %d\n", o1, o2, val[o1] - val[o2]);
47             int t = ((v & (1 << i)) ? 1 : 0);
48             if (val[ch[o1][!t]] - val[ch[o2][!t]])
49                 ret += (1 << i), o1 = ch[o1][!t],
50                 o2 = ch[o2][!t]; // 尽量向不同的地方
51             // 跳
52         }
53         else
54             o1 = ch[o1][t], o2 = ch[o2][t];
55     }
56     return ret;
57 }
58 } st;
59
60 int main() {
61     scanf("%d%d", &n, &q);
62     for (int i = 1; i <= n; i++) scanf("%d", a + i), s[i] = s[i -
63     1] ^ a[i];
64     for (int i = 1; i <= n; i++)
65         st.rt[i] = ++st.cnt, st.insert(st.rt[i], st.rt[i - 1], s[i
66         ]);
67     while (q--) {
68         scanf("%c", &op);
69         if (op == 'A') {
70             n++;
71             scanf("%d", a + n);
72             s[n] = s[n - 1] ^ a[n];
73             st.rt[n] = ++st.cnt;
74             st.insert(st.rt[n], st.rt[n - 1], s[n]);
75         }
76         if (op == 'Q') {
77             scanf("%d%d%d", &l, &r, &x);
78             l--;
79             r--;
80         }
81     }
82 }

```

```

74     if (l == 0)
75         printf("%d\n", max(s[n] ^ x, st.query(st.rt[r], st.rt
76         [0], s[n] ^ x)));
77     else
78         printf("%d\n", st.query(st.rt[r], st.rt[l - 1], s[n] ^
79         x));
80     }
81     return 0;
82 }

```

## 2.7 DSU

```

1 struct DSU{
2     vector<int>sz;
3     int n;
4     DSU(int _n):n(_n){
5         sz.assign(n+1,-1);
6     }
7     int Find(int x){
8         return sz[x]<0?x:sz[x] = Find(sz[x]);
9     }
10    bool Union(int a,int b){
11        int pa = Find(a),pb = Find(b);
12        if(pa==pb)return 0;
13        if((-sz[pa])<(-sz[pb]))swap(pa,pb);
14        sz[pa]+=sz[pb];
15        sz[pb] = pa;
16        return 1;
17    }
18 };

```

## 2.8 LiChao

```

1 template<class T>
2 struct LiChao {
3     static constexpr T INF = numeric_limits<T>::max();
4     struct Line {
5         T a, b;
6         Line(T a, T b) : a(a), b(b) {}
7         T operator()(T x) const { return a * x + b; }
8     };
9     int n;
10    vector<Line> fs;
11    vector<T> xs;
12    LiChao(const vector<T>& xs_) {
13        xs = sort_unique(xs_);
14        n = SZ(xs);
15        fs.assign(2 * n, Line(T(0), INF));
16    }
17    int index(T x) const { return lower_bound(ALL(xs), x) - xs.
18        begin(); }
19    void add_line(T a, T b) { update(a, b, 0, n); }
20    // [xl, xr) ax+b
21    void add_segment(T xl, T xr, T a, T b) {
22        int l = index(xl), r = index(xr);
23        update(a, b, l, r);
24    }
25    void update(T a, T b, int l, int r) {
26        Line g(a, b);
27        for(l += n, r += n; l < r; l >>= 1, r >>= 1){
28            if(l & 1) descend(g, l++);
29            if(r & 1) descend(g, --r);
30        }
31    }
32    void descend(Line g, int i) {
33        int l = i, r = i + 1;
34        while(l < n) l <= 1, r <= 1;
35        while(l < r) {
36            int c = (l + r) / 2;
37            T xl = xs[l - n], xr = xs[r - 1 - n], xc = xs[c - n];
38            Line& f = fs[i];
39            if(f(xl) <= g(xl) && f(xr) <= g(xr)) return;
40            if(f(xl) >= g(xl) && f(xr) >= g(xr)) { f = g; return; }
41            if(f(xc) > g(xc)) swap(f, g);
42            if(f(xl) > g(xl)) i = 2 * i, r = c;
43            else i = 2 * i + 1, l = c;
44        }
45    }
46    T get(T x) {
47        int i = index(x);
48        T res = INF;
49        for(i += n; i >>= 1) res = min(res, fs[i](x));
50        return res;
51    }
52 };

```

## 2.9 Fenwick



```

1 template<class T>struct fenwick_tree{
2     int n;
3     vector<T>arr;
4     inline int lowbit(int x){
5         return x&(-x);
6     }
7     fenwick_tree(int _n) : n(_n){
8         arr.assign(n+5,0);
9     }
10    T query(int x){
11        T ans = 0;
12        for(int i = x;i>0;i-=lowbit(i)){
13            ans+=arr[i];
14        }
15        return ans;
16    }
17    void update(int x,T y){
18        for(int i = x;i<=n;i+=lowbit(i)){
19            arr[i]+=y;
20        }
21    }
22 };

```

## 2.10 FastSet

```

1 // Can correctly work with numbers in range [0; MAXN]
2 // Supports all std::set operations in O(1) on random queries /
3 // dense arrays, O(log_64(N)) in worst case (sparse array).
4 // Count operation works in O(1) always.
5 template<uint MAXN>
6 class fast_set {
7 private:
8     static const uint PREF = (MAXN <= 64 ? 0 :
9         MAXN <= 4096 ? 1 :
10        MAXN <= 262144 ? 1 + 64 :
11        MAXN <= 16777216 ? 1 + 64 + 4096 :
12        MAXN <= 1073741824 ? 1 + 64 + 4096 + 262144 :
13        227) + 1;
14    static constexpr ull lb(int x) {
15        if(x == 64) return ULLONG_MAX;
16        return (1ULL << x) - 1;
17    };
18    static const uint SZ = PREF + (MAXN + 63) / 64 + 1;
19    ull m[SZ] = {0};
20    inline uint left(uint v) const { return (v - 62) * 64; }
21    inline uint parent(uint v) const { return v / 64 + 62; }
22    inline void setbit(uint v) { m[v >> 6] |= 1ULL << (v & 63); }
23    inline void resetbit(uint v) { m[v >> 6] &= ~(1ULL << (v & 63)); }
24    inline uint getbit(uint v) const { return m[v >> 6] >> (v & 63) & 1; }
25    inline ull childs_value(uint v) const { return m[left(v) >> 6]; }
26    inline int left_go(uint x, const uint c) const {
27        const ull rem = x & 63;
28        uint bt = PREF * 64 + x;
29        ull num = m[bt >> 6] & lb(rem + c);
30        if(num) return (x ^ rem) | __lg(num);
31        for(bt = parent(bt); bt > 62; bt = parent(bt)) {
32            const ull rem = bt & 63;
33            num = m[bt >> 6] & lb(rem);
34            if(num) {
35                bt = (bt ^ rem) | __lg(num);
36                break;
37            }
38        }
39        if(bt == 62) return -1;
40        while(bt < PREF * 64) bt = left(bt) | __lg(m[bt - 62]);
41        return bt - PREF * 64;
42    }
43    inline int right_go(uint x, const uint c) const {
44        const ull rem = x & 63;
45        uint bt = PREF * 64 + x;
46        ull num = m[bt >> 6] & ~lb(rem + c);
47        if(num) return (x ^ rem) | __builtin_ctzll(num);
48        for(bt = parent(bt); bt > 62; bt = parent(bt)) {
49            const ull rem = bt & 63;
50            num = m[bt >> 6] & ~lb(rem + 1);
51            if(num) {
52                bt = (bt ^ rem) | __builtin_ctzll(num);
53                break;
54            }
55        }
56        if(bt == 62) return -1;
57        while(bt < PREF * 64) bt = left(bt) | __builtin_ctzll(m[bt - 62]);
58        return bt - PREF * 64;
59    }
60 public:
61     fast_set() { assert(PREF != 228); setbit(62); }
62     bool empty() const { return getbit(63); }
63     void clear() { fill(m, m + SZ, 0); setbit(62); }

```

```

63 bool count(uint x) const { return m[PREF + (x >> 6)] >> (x & 63) & 1; }
64 void insert(uint x) { for(uint v = PREF * 64 + x; !getbit(v); v = parent(v)) setbit(v); }
65 void erase(uint x) {
66     if(!getbit(PREF * 64 + x)) return;
67     resetbit(PREF * 64 + x);
68     for(uint v = parent(PREF * 64 + x); v > 62 && !childs_value(v); v = parent(v)) resetbit(v);
69 }
70 int find_next(uint x) const { return right_go(x, 0); } // >=
71 int find_prev(uint x) const { return left_go(x, 1); } // <=
72 };

```

## 2.11 Persistent DSU

```

1 int rk[200001] = {};
2 struct Persistent_DSU{
3     rope<int>*p;
4     int n;
5     Persistent_DSU(int _n = 0):n(_n){
6         if(n==0)return;
7         p = new rope<int>;
8         int tmp[n+1] = {};
9         for(int i = 1;i<=n;++i)tmp[i] = i;
10        p->append(tmp,n+1);
11    }
12    Persistent_DSU(const Persistent_DSU &tmp){
13        p = new rope<int>(*tmp.p);
14        n = tmp.n;
15    }
16    int Find(int x){
17        int px = p->at(x);
18        return px==x?x:Find(px);
19    }
20    bool Union(int a,int b){
21        int pa = Find(a),pb = Find(b);
22        if(pa==pb)return 0;
23        if(rk[pa]<rk[pb])swap(pa,pb);
24        p->replace(pb,pa);
25        if(rk[pa]==rk[pb])rk[pa]++;
26        return 1;
27    }
28 };

```

## 2.12 TimingSegtree

```

1 template<class T,class D>struct timing_segment_tree{
2     struct node{
3         int l,r;
4         vector<T>opt;
5     };
6     vector<node>arr;
7     void build(int l,int r,int idx = 1){
8         if(idx==1)arr.resize((r-l+1)<<2);
9         if(l==r){
10            arr[idx].l = arr[idx].r = l;
11            arr[idx].opt.clear();
12            return;
13        }
14        int m = (l+r)>>1;
15        build(l,m,idx<<1);
16        build(m+1,r,idx<<1|1);
17        arr[idx].l = l,arr[idx].r = r;
18        arr[idx].opt.clear();
19    }
20    void update(int ql,int qr,T k,int idx = 1){
21        if(ql<=arr[idx].l and arr[idx].r<=qr){
22            arr[idx].opt.push_back(k);
23            return;
24        }
25        int m = (arr[idx].l+arr[idx].r)>>1;
26        if(ql<=m)update(ql,qr,k,idx<<1);
27        if(qr>m)update(ql,qr,k,idx<<1|1);
28    }
29    void dfs(D &d,vector<int>&ans,int idx = 1){
30        int cnt = 0;
31        for(auto [a,b]:arr[idx].opt){
32            if(d.Union(a,b))cnt++;
33        }
34        if(arr[idx].l==arr[idx].r)ans[arr[idx].l] = d.comps;
35        else{
36            dfs(d,ans,idx<<1);
37            dfs(d,ans,idx<<1|1);
38        }
39        while(cnt-->0)d.undo();
40    }
41 };

```

## 2.13 AreaOfRectangles

```

1 long long AreaOfRectangles(vector<tuple<int,int,int,int>>>v){
2     vector<tuple<int,int,int,int>>tmp;
3     int L = INT_MAX,R = INT_MIN;
4     for(auto [x1,y1,x2,y2]:v){
5         tmp.push_back({x1,y1+1,y2,1});
6         tmp.push_back({x2,y1+1,y2,-1});
7         R = max(R,y2);
8         L = min(L,y1);
9     }
10    vector<long long>seg((R-L+1)<<2),tag((R-L+1)<<2);
11    sort(tmp.begin(),tmp.end());
12    function<void(int,int,int,int,int,int)>update = [&](int q1,
13        int qr,int val,int l,int r,int idx){
14        if(q1<=l and r<=qr){
15            tag[idx]+=val;
16            if(tag[idx])seg[idx] = r-l+1;
17            else if(l==r)seg[idx] = 0;
18            else seg[idx] = seg[idx<<1]+seg[idx<<1|1];
19            return;
20        }
21        int m = (l+r)>>1;
22        if(q1<=m)update(q1,qr,val,l,m,idx<<1);
23        if(qr>m)update(q1,qr,val,m+1,r,idx<<1|1);
24        if(tag[idx])seg[idx] = r-l+1;
25        else seg[idx] = seg[idx<<1]+seg[idx<<1|1];
26    };
27    long long last_pos = 0,ans = 0;
28    for(auto [pos,l,r,val]:tmp){
29        ans+=(pos-last_pos)*seg[l];
30        update(l,r,val,L,R,1);
31        last_pos = pos;
32    }
33    return ans;
34 }

```

## 2.14 SparseTable

```

1 template<class T,T (*op)(T,T)>struct sparse_table{
2     int n;
3     vector<vector<T>>mat;
4     sparse_table(): n(0){}
5     sparse_table(const vector<T>&v){
6         n = (int)(v.size());
7         mat.resize(30);
8         mat[0] = v;
9         for(int i = 1;(1<i)<=n;++i){
10            mat[i].resize(n-(1<i)+1);
11            for(int j = 0;j<n-(1<i)+1;++j){
12                mat[i][j] = op(mat[i-1][j],mat[i-1][j+(1<<(i-1))]);
13            }
14        }
15    }
16    T query(int q1,int qr){
17        int k = __lg(qr-q1+1);
18        return op(mat[k][q1],mat[k][qr-(1<<k)+1]);
19    }
20 };

```

## 2.15 DynamicSegtree

```

1 template<class T>struct dynamic_segment_tree{
2     struct node{
3         node *l = NULL,*r = NULL;
4         T sum;
5         node(T k = 0): sum(k){}
6         node(node *p){if(p)*this = *p;}
7         ~node(){
8             for(auto &i:{l,r})
9                 if(i)delete i;
10        }
11        void pull(){
12            sum = 0;
13            for(auto i:{l,r})
14                if(i)sum+=i->sum;
15        }
16    }*root = NULL;
17    int n;
18    dynamic_segment_tree(){
19        dynamic_segment_tree(const dynamic_segment_tree<T>&tmp){root
20            = new node(tmp.root);}
21    void update(node *t,int pos,T k,int l,int r){
22        if(!t)t = new node();
23        if(l==r)return t = new node(k),void();
24        int m = (l+r)>>1;
25        t = new node(t);
26        if(pos<=m)update(t->l,pos,k,l,m);
27        else update(t->r,pos,k,m+1,r);

```

```

28        t->pull();
29    }void update(int pos,T k,int l = -1e9,int r = 1e9){update(
30        root,pos,k,l,r);}
31    T query(node *t,int q1,int qr,int l,int r){
32        if(!t)return 0;
33        if(q1<=l and r<=qr)return t->sum;
34        int m = (l+r)>>1;
35        T ans = 0;
36        if(q1<=m)ans+=query(t->l,q1,qr,l,m);
37        if(qr>m)ans+=query(t->r,q1,qr,m+1,r);
38        return ans;
39    }T query(int q1,int qr,int l = -1e9,int r = 1e9){return query
40        (root,q1,qr,l,r);}
41 };

```

## 2.16 ZkwSegtree

```

1 template<class S,
2     S (*node_pull)(S, S),
3     S (*node_init)(),
4     class F,
5     S (*mapping)(S, F),
6     F (*tag_pull)(F, F),
7     F (*tag_init)()>
8 class segment_tree {
9 public:
10    segment_tree(): segment_tree(0) {}
11    explicit segment_tree(int _n): segment_tree(vector<S>(_n,
12        node_init())) {}
13    explicit segment_tree(const vector<S>& v): n((int) v.size())
14    {
15        log = std::__lg(2 * n - 1);
16        size = 1 << log;
17        d = vector<S>(size << 1, node_init());
18        lz = vector<F>(size, tag_init());
19        for(int i = 0; i < n; i++) {
20            d[size + i] = v[i];
21        }
22        for(int i = size - 1; i; --i) {
23            update(i);
24        }
25    }
26    void set(int p, S x) {
27        assert(0 <= p && p < n);
28        p += size;
29        for(int i = log; i; --i) {
30            push(p >> i);
31        }
32        d[p] = x;
33        for(int i = 1; i <= log; ++i) {
34            update(p >> i);
35        }
36    }
37    S get(int p) {
38        assert(0 <= p && p < n);
39        p += size;
40        for(int i = log; i; i--) {
41            push(p >> i);
42        }
43        return d[p];
44    }
45    S operator[](int p) {
46        return get(p);
47    }
48    S query(int l, int r) {
49        r++;
50        assert(l<=r);
51        l += size;
52        r += size;
53        for(int i = log; i; i--) {
54            if(((l >> i) << i) != 1) {
55                push(l >> i);
56            }
57            if(((r >> i) << i) != r) {
58                push(r >> i);
59            }
60        }
61        S sml = node_init(), smr = node_init();
62        while(l < r) {
63            if(l & 1) {
64                sml = node_pull(sml, d[l++]);
65            }
66            if(r & 1) {
67                smr = node_pull(d[--r], smr);
68            }
69            l >>= 1;
70            r >>= 1;
71        }
72        return node_pull(sml, smr);
73    }
74    void apply(int p, F f) {
75        assert(0 <= p && p < n);
76        p += size;
77        for(int i = log; i; i--) {

```

```

76     push(p >> i);
77 }
78 d[p] = mapping(f, d[p]);
79 for(int i = 1; i <= log; i++) {
80     update(p >> i);
81 }
82 }
83 void update(int l, int r, F f) {
84     r++;
85     assert(l<=r);
86     l += size;
87     r += size;
88     for(int i = log; i; i--) {
89         if(((l >> i) << i) != 1) {
90             push(l >> i);
91         }
92         if(((r >> i) << i) != r) {
93             push((r - 1) >> i);
94         }
95     }
96     {
97         int l2 = l, r2 = r;
98         while(l < r) {
99             if(l & 1) {
100                 all_apply(l++, f);
101             }
102             if(r & 1) {
103                 all_apply(--r, f);
104             }
105             l >>= 1;
106             r >>= 1;
107         }
108         l = l2;
109         r = r2;
110     }
111     for(int i = 1; i <= log; i++) {
112         if(((l >> i) << i) != 1) {
113             update(l >> i);
114         }
115         if(((r >> i) << i) != r) {
116             update((r - 1) >> i);
117         }
118     }
119 }
120 private:
121 int n, size, log;
122 vector<S> d;
123 vector<F> lz;
124 inline void update(int k) { d[k] = node_pull(d[k << 1], d[k
    << 1 | 1]); }
125 void all_apply(int k, F f) {
126     d[k] = mapping(d[k], f);
127     if(k < size) {
128         lz[k] = tag_pull(lz[k], f);
129     }
130 }
131 void push(int k) {
132     all_apply(k << 1, lz[k]);
133     all_apply(k << 1 | 1, lz[k]);
134     lz[k] = tag_init();
135 }
136 };

```

## 2.17 MoAlgo

```

1 struct qry{
2     int ql,qr,id;
3 };
4 template<class T>struct Mo{
5     int n,m;
6     vector<pii>ans;
7     Mo(int _n,int _m): n(_n),m(_m){
8         ans.resize(m);
9     }
10    void solve(vector<T>&v,vector<qry>&q){
11        int l = 0,r = -1;
12        vector<int>cnt,cntcnt;
13        cnt.resize(n+5);
14        cntcnt.resize(n+5);
15        int mx = 0;
16        function<void(int)>add = [&](int pos){
17            cntcnt[cnt[v[pos]]]--;
18            cnt[v[pos]]++;
19            cntcnt[cnt[v[pos]]]++;
20            mx = max(mx,cnt[v[pos]]);
21        };
22        function<void(int)>sub = [&](int pos){
23            if(!--cntcnt[cnt[v[pos]]] and cnt[v[pos]]==mx)mx--;
24            cnt[v[pos]]--;
25            cntcnt[cnt[v[pos]]]++;
26            mx = max(mx,cnt[v[pos]]);
27        };
28        sort(all(q),[&](qry a,qry b){
29            static int B = max((int)1,n/max((int)sqrt(m),(int)1));

```

```

30         if(a.ql/B!=b.ql/B)return a.ql<b.ql;
31         if((a.ql/B)&1)return a.qr>b.qr;
32         return a.qr<b.qr;
33     });
34     for(auto [ql,qr,id]:q){
35         while(l>ql)add(--l);
36         while(r<qr)add(++r);
37         while(l<ql)sub(l++);
38         while(r>qr)sub(r--);
39         ans[id] = {mx,cntcnt[mx]};
40     }
41 }
42 };

```

## 2.18 Hash

```

1 struct custom_hash {
2     static uint64_t splitmix64(uint64_t x) {
3         x += 0x9e3779b97f4a7c15;
4         x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9;
5         x = (x ^ (x >> 27)) * 0x94d049bb133111eb;
6         return x ^ (x >> 31);
7     }
8     size_t operator()(uint64_t x) const {
9         static const uint64_t FIXED_RANDOM = chrono::steady_clock::
            now().time_since_epoch().count();
10        return splitmix64(x + FIXED_RANDOM);
11    }
12    size_t operator()(pair<uint64_t,uint64_t> x) const {
13        static const uint64_t FIXED_RANDOM = chrono::steady_clock::
            now().time_since_epoch().count();
14        return splitmix64(3*x.first + x.second + FIXED_RANDOM);
15    }
16 };
17 template<class T,class U>using hash_map = gp_hash_table<T,U,
    custom_hash>;

```

## 2.19 RedBlackTree

```

1 template<class T, typename cmp=less<>>struct _tree{//#include<
    bits/extc++.h>
2     tree<pair<T,int>,null_type,cmp,rb_tree_tag,
        tree_order_statistics_node_update>st;
3     int id = 0;
4     void insert(T x){st.insert({x,id++});}
5     void erase(T x){st.erase(st.lower_bound({x,0}));}
6     int order_of_key(T x){return st.order_of_key(*st.lower_bound
        ({x,0}));}
7     T find_by_order(int x){return st.find_by_order(x)->first;}
8     T lower_bound(T x){return st.lower_bound({x,0})->first;}
9     T upper_bound(T x){return st.upper_bound({x,(int)1e9+7})->
        first;}
10    T smaller_bound(T x){return (--st.lower_bound({x,0}))->first
        ;}
11 };

```

# 3 Geometry

## 3.1 GeometryTheorem

- Pick' s Theorem

$$A = I + \frac{B}{2} - 1$$

$$A := \text{Area}$$

$$i := \text{PointsInside}$$

$$B := \text{PointsBoundary}$$

## 3.2 PointInPolygon

```

1 template<class T>
2 int PointInPolygon(const vector<Point<T>> &Poly, const Point<T>
    p){
3     int ans = 0;
4     for(auto a = --Poly.end(),b = Poly.begin();b!=Poly.end();a =
        b++){
5         if(PointOnSegment(*a,*b,p)){
6             return -1;
7         }
8         if(seg_intersect(p,p+Point<T>(2e9+7,1),*a,*b)){
9             ans = !ans;
10        }
11    }
12    return ans;
13 }

```

### 3.3 PointInConvex

```

1 template<class T>
2 int PointInConvex(const vector<Point<T>>&C, const Point<T>&p){
3     if(btw(C[0],C[1],p) || btw(C[0],C.back(),p))return -1;
4     int l = 0, r = (int)C.size()-1;
5     while(l<=r){
6         int m = (l+r)>>1;
7         auto a1 = (C[m]-C[0])^(p-C[0]);
8         auto a2 = (C[(m+1)%C.size()-1]-C[0])^(p-C[0]);
9         if(a1>=0 and a2<=0){
10             auto res = (C[(m+1)%C.size()-1]-C[m])^(p-C[m]);
11             return res > 0 ? 1 : (res >= 0 ? -1 : 0);
12         }
13         if(a1 < 0) r = m-1;
14         else l = m+1;
15     }
16     return 0;
17 }

```

### 3.4 MaximumDistance

```

1 template<class T>
2 T MaximumDistance(vector<Point<T>>&p){
3     vector<Point<T>>C = ConvexHull(p,0);
4     int n = C.size(), t = 2;
5     T ans = 0;
6     for(int i = 0; i < n; i++){
7         while(((C[i] - C[t]) ^ (C[(i+1)%n] - C[t])) < ((C[i] - C[(t+1)%n]) ^ (C[(i+1)%n] - C[(t+1)%n]))) t = (t+1)%n;
8         ans = max({ans, abs2(C[i] - C[t]), abs2(C[(i+1)%n] - C[t])});
9     }
10     return ans;
11 }

```

### 3.5 PolarAngleSort

```

1 template<class T>
2 bool cmp(const Point<T> &a, const Point<T> &b){
3     int lhs = (a.y < 0 || a.y==0 && a.x > 0) ? 0 : (1 + (a.x != 0 || a.y != 0));
4     int rhs = (b.y < 0 || b.y==0 && b.x > 0) ? 0 : (1 + (b.x != 0 || b.y != 0));
5     if(lhs != rhs) {
6         return lhs < rhs;
7     }
8     long long area = (a^b);
9     return area ? area > 0 : abs(a.x) + abs(a.y) < abs(b.x) + abs(b.y);
10 }

```

### 3.6 MinimumDistance

```

1 template<class T>
2 T MinimumDistance(vector<Point<T>>&p, int l = -1, int r = -1){
3     if(l==l and r==r){
4         sort(p.begin(), p.end(), [](Point<T> a, Point<T> b){
5             if(a.x!=b.x)return a.x<b.x;
6             return a.y<b.y;
7         });
8         p.erase(unique(p.begin(), p.end()), p.end());
9         return MinimumDistance(p, 0, p.size()-1);
10    }
11    if(l==r)return numeric_limits<T>::max();
12    int m = (l+r)>>1, mid_pos = p[m].x;
13    T ans = min(MinimumDistance(p, l, m), MinimumDistance(p, m+1, r));
14    vector<Point<T>>tmp((r-l+1), Point<T>(0,0));
15    merge(p.begin()+l, p.begin()+m+1, p.begin()+m+1, p.begin()+r+1,
16          tmp.begin(), [](Point<T> a, Point<T> b){return a.y<b.y;});
17    for(int i = l; i<=r; ++i)p[i] = tmp[i-l];
18    tmp.clear();
19    for(int i = l; i<=r; ++i){
20        if((p[i].x-mid_pos)*(p[i].x-mid_pos)<ans){
21            tmp.push_back(p[i]);
22        }
23    }
24    int n = tmp.size();
25    for(int i = 0; i<=n; ++i){
26        for(int j = i+1; j<=n; ++j){
27            ans = min(ans, abs2(tmp[i]-tmp[j]));
28            if(((tmp[i].y-tmp[j].y)*(tmp[i].y-tmp[j].y))>ans){
29                break;
30            }
31        }
32    }
33 }

```

```

31 }
32 return ans;
33 }

```

### 3.7 ConvexHull

```

1 template<class T>
2 vector<Point<T>> ConvexHull(vector<Point<T>> v, bool Boundary = 1){
3     sort(begin(v), end(v), [](Point<T> &a, Point<T> &b){
4         if(a.x!=b.x)return a.x<b.x;
5         return a.y<b.y;
6     });
7     vector<Point<T>>ans;
8     int t = 1;
9     auto add = [](Point<T> &p){
10         while(ans.size() > t and ((p - ans[ans.size()-2])^(ans.back() - ans[ans.size()-2])) > (Boundary ? 0 : 0-eps))
11             ans.pop_back();
12         ans.push_back(p);
13     };
14     for(int i = 0; i < v.size(); ++i) add(v[i]);
15     t = ans.size();
16     for(int i = (int)(v.size()-2); i >= 0; --i) add(v[i]);
17     if(v.size() > 1) ans.pop_back();
18     return ans;
19 }

```

### 3.8 Template

```

1 template<class T>
2 struct Point{
3     T x,y;
4     Point(T x = 0, T y = 0) : x(x), y(y) {}
5     Point operator + (const Point &b) const {
6         return Point(x + b.x, y + b.y);
7     }
8     Point operator - (const Point &b) const {
9         return Point(x - b.x, y - b.y);
10    }
11    Point operator * (T b) const {
12        return Point(x*b, y*b);
13    }
14    Point operator / (T b) const {
15        return Point(x/b, y/b);
16    }
17    T operator * (const Point &b) const {
18        return x * b.x + y * b.y;
19    }
20    T operator ^ (const Point &b) const {
21        return x * b.y - y * b.x;
22    }
23 };
24 int sign(double a){
25     return fabs(a) < eps ? 0 : a > 0 ? 1 : -1;
26 }
27 template<class T>
28 double abs(const Point<T>&p){
29     return sqrt(1+p*p);
30 }
31 template<class T>
32 T abs2(const Point<T>&p){
33     return p*p;
34 }
35 template<class T>
36 int ori(Point<T> a, Point<T> b, Point<T> c){
37     return sign((b-a)^(c-a));
38 }
39 template<class T>
40 bool collinearity(Point<T> p1, Point<T> p2, Point<T> p3){
41     return sign((p1-p3)^(p2-p3)) == 0;
42 }
43 template<class T>
44 bool btw(Point<T> p1, Point<T> p2, Point<T> p3) {
45     if(!collinearity(p1, p2, p3)) return 0;
46     return sign((p1-p3)*(p2-p3)) <= 0;
47 }
48 template<class T>
49 bool PointOnSegment(const Point<T> &p1, const Point<T> &p2, const Point<T> &p3){
50     return collinearity(p1, p2, p3) && btw(p1, p2, p3);
51 }
52 template<class T>
53 bool seg_intersect(Point<T> p1, Point<T> p2, Point<T> p3, Point<T> p4) {
54     int a123 = ori(p1, p2, p3);
55     int a124 = ori(p1, p2, p4);
56     int a341 = ori(p3, p4, p1);
57     int a342 = ori(p3, p4, p2);
58     if(a123 == 0 && a124 == 0)

```

```

59     return btw(p1, p2, p3) || btw(p1, p2, p4) || btw(p3, p4, p1
60         ) || btw(p3, p4, p2);
61     return a123 * a124 <= 0 && a341 * a342 <= 0;
62 }
63 template<class T>
64 double area(vector<Point<T>> v){
65     if(v.size()<=2)return 0;
66     double ans = 0;
67     for(int i = 1;i<v.size()-1;++i){
68         ans+=((v[i]-v[0])^(v[i+1]-v[0]));
69     }
70     return abs(ans)/2.;
71 }

```

## 4 Graph

### 4.1 HLD

```

1 struct heavy_light_decomposition{
2     int n;
3     vector<int>dep,father,sz,mxson,topf,id;
4     vector<vector<int>>>g;
5     heavy_light_decomposition(int _n = 0) : n(_n) {
6         g.resize(n+5);
7         dep.resize(n+5);
8         father.resize(n+5);
9         sz.resize(n+5);
10        mxson.resize(n+5);
11        topf.resize(n+5);
12        id.resize(n+5);
13    }
14    void add_edge(int u, int v){
15        g[u].push_back(v);
16        g[v].push_back(u);
17    }
18    void dfs(int u,int p){
19        dep[u] = dep[p]+1;
20        father[u] = p;
21        sz[u] = 1;
22        mxson[u] = 0;
23        for(auto v:g[u]){
24            if(v==p)continue;
25            dfs(v,u);
26            sz[u]+=sz[v];
27            if(sz[v]>sz[mxson[u]])mxson[u] = v;
28        }
29    }
30    void dfs2(int u,int top){
31        static int idn = 0;
32        topf[u] = top;
33        id[u] = ++idn;
34        if(mxson[u])dfs2(mxson[u],top);
35        for(auto v:g[u]){
36            if(v!=father[u] and v!=mxson[u]){
37                dfs2(v,v);
38            }
39        }
40    }
41    void build(int root){
42        dfs(root,0);
43        dfs2(root,root);
44    }
45    vector<pair<int, int>> path(int u,int v){
46        vector<pair<int, int>>ans;
47        while(topf[u]!=topf[v]){
48            if(dep[topf[u]]<dep[topf[v]])swap(u,v);
49            ans.push_back({id[topf[u]], id[u]});
50            u = father[topf[u]];
51        }
52        if(id[u]>id[v])swap(u,v);
53        ans.push_back({id[u], id[v]});
54        return ans;
55    }
56 };

```

### 4.2 Bridges

```

1 vector<pii> findBridges(const vector<vector<int>>& g) {
2     int n = (int) g.size();
3     vector<int> id(n, -1), low(n);
4     vector<pii> bridges;
5     function<void(int, int)> dfs = [&](int u, int p) {
6         static int cnt = 0;
7         id[u] = low[u] = cnt++;
8         for(auto v : g[u]) {
9             if(v == p) continue;
10            if(id[v] != -1) low[u] = min(low[u], id[v]);
11            else {
12                dfs(v, u);

```

```

13        low[u] = min(low[u], low[v]);
14        if(low[v] > id[u]) bridges.EB(u, v);
15    }
16 }
17 };
18 for(int i = 0; i < n; ++i) {
19     if(id[i] == -1) dfs(i, -1);
20 }
21 return bridges;
22 }

```

### 4.3 FlowTheorems

- Max-Flow Min-Cut Theorem

Max-Flow  $\equiv$  Min-Cut

- Kőnig's theorem

Minimum Vertex Cover  $\equiv$  Maximum Matching

- Independent Set

Maximum independent set  $\equiv N - \text{Minimum vertex cover}$

### 4.4 TwoSat

```

1 struct two_sat{
2     SCC s;
3     vector<bool>ans;
4     int have_ans = 0;
5     int n;
6     two_sat(int _n) : n(_n) {
7         ans.resize(n+1);
8         s = SCC(2*n);
9     }
10    int inv(int x){
11        if(x>n)return x-n;
12        return x+n;
13    }
14    void add_or_clause(int u, bool x, int v, bool y){
15        if(!x)u = inv(u);
16        if(!y)v = inv(v);
17        s.add_edge(inv(u), v);
18        s.add_edge(inv(v), u);
19    }
20    void check(){
21        if(have_ans!=0)return;
22        s.build();
23        for(int i = 0;i<=n;++i){
24            if(s.scc[i]==s.scc[inv(i)]){
25                have_ans = -1;
26                return;
27            }
28            ans[i] = (s.scc[i]<s.scc[inv(i)]);
29        }
30        have_ans = 1;
31    }
32 };

```

### 4.5 MCMF

```

1 template<class Cap_t, class Cost_t>
2 class MCMF {
3 public:
4     struct Edge {
5         int from;
6         int to;
7         Cap_t cap;
8         Cost_t cost;
9         Edge(int u, int v, Cap_t _cap, Cost_t _cost) : from(u), to(
10            v), cap(_cap), cost(_cost) {}
11    };
12    static constexpr Cap_t EPS = static_cast<Cap_t>(1e-9);
13    int n;
14    vector<Edge> edges;
15    vector<vector<int>>> g;
16    vector<Cost_t> d;
17    vector<bool> in_queue;
18    vector<int> previous_edge;
19    MCMF() {}
20    MCMF(int _n) : n(_n+1), g(_n+1), d(_n+1), in_queue(_n+1),
21        previous_edge(_n+1) {}
22    void add_edge(int u, int v, Cap_t cap, Cost_t cost) {
23        assert(0 <= u && u < n);
24    }
25 };

```

```

26     assert(0 <= v && v < n);
27     g[u].push_back(edges.size());
28     edges.emplace_back(u, v, cap, cost);
29     g[v].push_back(edges.size());
30     edges.emplace_back(v, u, 0, -cost);
31 }
32
33 bool spfa(int s, int t) {
34     bool found = false;
35     fill(d.begin(), d.end(), numeric_limits<Cost_t>::max());
36     d[s] = 0;
37     in_queue[s] = true;
38     queue<int> que;
39     que.push(s);
40     while(!que.empty()) {
41         int u = que.front();
42         que.pop();
43         if(u == t) {
44             found = true;
45         }
46         in_queue[u] = false;
47         for(auto& id : g[u]) {
48             const Edge& e = edges[id];
49             if(e.cap > EPS && d[u] + e.cost < d[e.to]) {
50                 d[e.to] = d[u] + e.cost;
51                 previous_edge[e.to] = id;
52                 if(!in_queue[e.to]) {
53                     que.push(e.to);
54                     in_queue[e.to] = true;
55                 }
56             }
57         }
58     }
59     return found;
60 }
61
62 pair<Cap_t, Cost_t> flow(int s, int t, Cap_t f =
    numeric_limits<Cap_t>::max()) {
63     assert(0 <= s && s < n);
64     assert(0 <= t && t < n);
65     Cap_t cap = 0;
66     Cost_t cost = 0;
67     while(f > 0 && spfa(s, t)) {
68         Cap_t send = f;
69         int u = t;
70         while(u != s) {
71             const Edge& e = edges[previous_edge[u]];
72             send = min(send, e.cap);
73             u = e.from;
74         }
75         u = t;
76         while(u != s) {
77             Edge& e = edges[previous_edge[u]];
78             e.cap -= send;
79             Edge& b = edges[previous_edge[u] ^ 1];
80             b.cap += send;
81             u = e.from;
82         }
83         cap += send;
84         f -= send;
85         cost += send * d[t];
86     }
87     return make_pair(cap, cost);
88 }
89 };

```

## 4.6 LCA

```

1 vector<vector<int>>>g,dp;
2 vector<int>deep;
3 void build(int root,int n){
4     dp.assign(25,vector<int>(n+5));
5     deep.assign(n+5,0);
6     function<void(int,int,int)>dfs = [&](int u,int p,int dis){
7         dp[0][u] = p;
8         deep[u] = dis;
9         for(auto v:g[u]){
10             if(v==p)continue;
11             dfs(v,u,dis+1);
12         }
13     };
14     dfs(root,0,1);
15     for(int i = 1;i<=20;++i){
16         for(int j = 1;j<=n;++j){
17             dp[i][j] = dp[i-1][dp[i-1][j]];
18         }
19     }
20 }
21 int LCA(int u,int v){
22     if(deep[u]<deep[v])swap(u,v);
23     for(int i = 20;i>=0;--i){
24         if(deep[dp[i][u]]>=deep[v])
25             u = dp[i][u];
26     }

```

```

27     if(u==v)return u;
28     for(int i = 20;i>=0;--i){
29         if(dp[i][u]!=dp[i][v])u = dp[i][u],v = dp[i][v];
30     }
31     return dp[0][u];
32 }

```

## 4.7 CentroidDecomposition

```

1 vector<vector<int>>>g;
2 vector<int>sz,tmp;
3 vector<bool>vis; //visit_centroid
4 int tree_centroid(int u,int n){
5     function<void(int,int)>dfs1 = [&](int u,int p){
6         sz[u] = 1;
7         for(auto v:g[u]){
8             if(v==p)continue;
9             if(vis[v])continue;
10            dfs1(v,u);
11            sz[u]+=sz[v];
12        }
13    };
14    function<int(int,int)>dfs2 = [&](int u,int p){
15        for(auto v:g[u]){
16            if(v==p)continue;
17            if(vis[v])continue;
18            if(sz[v]*2<n)continue;
19            return dfs2(v,u);
20        }
21        return u;
22    };
23    dfs1(u,-1);
24    return dfs2(u,-1);
25 }
26 int cal(int u,int p = -1,int deep = 1){
27     int ans = 0;
28     tmp.pb(deep);
29     sz[u] = 1;
30     for(auto v:g[u]){
31         if(v==p)continue;
32         if(vis[v])continue;
33         ans+=cal(v,u,deep+1);
34         sz[u]+=sz[v];
35     }
36     //calculate the answer
37     return ans;
38 }
39 int centroid_decomposition(int u,int tree_size){
40     int center = tree_centroid(u,tree_size);
41     vis[center] = 1;
42     int ans = 0;
43     for(auto v:g[center]){
44         if(vis[v])continue;
45         ans+=cal(v);
46         for(int i = sz(tmp)-sz[v];i<sz(tmp);++i){
47             //update
48         }
49     }
50     while(!tmp.empty()){
51         //roll_back(tmp.back())
52         tmp.pop_back();
53     }
54     for(auto v:g[center]){
55         if(vis[v])continue;
56         ans+=centroid_decomposition(v,sz[v]);
57     }
58     return ans;
59 }

```

## 4.8 BCC AP

```

1 struct BCC_AP{
2     int dfn_cnt = 0,bcc_cnt = 0,n;
3     vector<int>dfn,low,ap,bcc_id;
4     stack<int>st;
5     vector<bool>vis,is_ap;
6     vector<vector<int>>>bcc;
7     BCC_AP(int _n):n(_n){
8         dfn.resize(n+5),low.resize(n+5),bcc.resize(n+5),vis.resize(
9             n+5),is_ap.resize(n+5),bcc_id.resize(n+5);
10    }
11    inline void build(const vector<vector<int>>>&g,int u,int p =
12        -1){
13        int child = 0;
14        dfn[u] = low[u] = ++dfn_cnt;
15        st.push(u);
16        vis[u] = 1;
17        if(g[u].empty() and p==-1){
18            bcc_id[u] = ++bcc_cnt;
19            bcc[bcc_cnt].push_back(u);
20            return;

```



```

19     }
20     for(auto v:g[u]){
21         if(v==p)continue;
22         if(!dfn[v]){
23             build(g,v,u);
24             child++;
25             if(dfn[u]<=low[v]){
26                 is_ap[u] = 1;
27                 bcc_id[u] = ++bcc_cnt;
28                 bcc[bcc_cnt].push_back(st.top());
29                 while(vis[v]){
30                     bcc_id[st.top()] = bcc_cnt;
31                     bcc[bcc_cnt].push_back(st.top());
32                     vis[st.top()] = 0;
33                     st.pop();
34                 }
35             }
36             low[u] = min(low[u],low[v]);
37         }
38         low[u] = min(low[u],dfn[v]);
39     }
40     if(p==-1 and child<2)is_ap[u] = 0;
41     if(is_ap[u])ap.push_back(u);
42 }
43 };

```

## 4.9 CentroidTree

```

1 pair<int, vector<vi>> centroid_tree(const vector<vi>& g) {
2     int n = sz(g);
3     vi siz(n);
4     vector<bool> vis(n);
5     auto dfs_sz = [&](auto f, int u, int p) -> void {
6         siz[u] = 1;
7         for(auto v : g[u]) {
8             if(v == p || vis[v]) continue;
9             f(f, v, u);
10            siz[u] += siz[v];
11        }
12    };
13    auto find_cd = [&](auto f, int u, int p, int all) -> int {
14        for(auto v : g[u]) {
15            if(v == p || vis[v]) continue;
16            if(siz[v] * 2 > all) return f(f, v, u, all);
17        }
18        return u;
19    };
20    vector<vi> h(n);
21    auto build = [&](auto f, int u) -> int {
22        dfs_sz(dfs_sz, u, -1);
23        int cd = find_cd(find_cd, u, -1, siz[u]);
24        vis[cd] = true;
25        for(auto v : g[cd]) {
26            if(vis[v]) continue;
27            int child = f(f, v);
28            h[cd].pb(child);
29        }
30        return cd;
31    };
32    int root = build(build, 0);
33    return {root, h};
34 }

```

## 4.10 SCC

```

1 struct SCC{
2     int n,cnt = 0,dfn_cnt = 0;
3     vector<vector<int>>>g;
4     vector<int>sz,scc,low,dfn;
5     stack<int>st;
6     vector<bool>vis;
7     SCC(int _n = 0) : n(_n){
8         sz.resize(n+5),scc.resize(n+5),low.resize(n+5),dfn.resize(n
9         +5),vis.resize(n+5);
10        g.resize(n+5);
11    }
12    inline void add_edge(int u, int v){
13        g[u].push_back(v);
14    }
15    inline void build(){
16        function<void(int, int)>dfs = [&](int u,int dis){
17            low[u] = dfn[u] = ++dfn_cnt,vis[u] = 1;
18            st.push(u);
19            for(auto v:g[u]){
20                if(!dfn[v]){
21                    dfs(v, dis+1);
22                    low[u] = min(low[u],low[v]);
23                }
24                else if(vis[v]){
25                    low[u] = min(low[u],dfn[v]);
26                }
27            }
28        };
29        build();
30    }

```

```

26     }
27     if(low[u]==dfn[u]){
28         ++cnt;
29         while(vis[u]){
30             auto v = st.top();
31             st.pop();
32             vis[v] = 0;
33             scc[v] = cnt;
34             sz[cnt]++;
35         }
36     }
37     };
38     for(int i = 0;i<=n;++i){
39         if(!scc[i]){
40             dfs(i, 1);
41         }
42     }
43 }
44 vector<vector<int>> compress(){
45     vector<vector<int>>ans(cnt+1);
46     for(int u = 0;u<=n;++u){
47         for(auto v:g[u]){
48             if(scc[u] == scc[v]){
49                 continue;
50             }
51             ans[scc[u]].push_back(scc[v]);
52         }
53     }
54     for(int i = 0;i<=cnt;++i){
55         sort(ans[i].begin(), ans[i].end());
56         ans[i].erase(unique(ans[i].begin(), ans[i].end()), ans[i
57         ].end());
58     }
59     return ans;
60 };

```

## 4.11 TriangleSum

```

1 // Three vertices a < b < cconnected by three edges {a, b}, {a,
2 // c}, {b, c}. Find xa * xb * xc over all triangles.
3 int triangle_sum(vector<array<int, 2>> edges, vi x) {
4     int n = SZ(x);
5     vi deg(n);
6     vector<vector<int>> g(n);
7     for(auto& [u, v] : edges) {
8         if(u > v) swap(u, v);
9         deg[u]++, deg[v]++;
10    }
11    REP(i, n) g[i].reserve(deg[i]);
12    for(auto [u, v] : edges) {
13        if(deg[u] > deg[v]) swap(u, v);
14        g[u].pb(v);
15    }
16    vi val(n);
17    __int128 ans = 0;
18    REP(a, n) {
19        for(auto b : g[a]) val[b] = x[b];
20        for(auto b : g[a]) {
21            ll tmp = 0;
22            for(auto c : g[b]) tmp += val[c];
23            ans += __int128(tmp) * x[a] * x[b];
24        }
25        for(auto b : g[a]) val[b] = 0;
26    }
27    return ans % mod;
28 }

```

## 4.12 LineContainer

```

1 template<class T>
2 T floor_div(T a, T b) {
3     return a / b - ((a ^ b) < 0 && a % b != 0);
4 }
5
6 template<class T>
7 T ceil_div(T a, T b) {
8     return a / b + ((a ^ b) > 0 && a % b != 0);
9 }
10
11 namespace line_container_internal {
12
13 struct line_t {
14     mutable long long k, m, p;
15
16     inline bool operator<(const line_t& o) const { return k < o.k
17     ; }
18     inline bool operator<(long long x) const { return p < x; }
19 };
20 } // line_container_internal

```

```

21
22 template<bool MAX>
23 struct line_container : std::multiset<line_container_internal::
    line_t, std::less<>> {
24     static const long long INF = std::numeric_limits<long long>::
        max();
25
26     bool isect(iterator x, iterator y) {
27         if(y == end()) {
28             x->p = INF;
29             return 0;
30         }
31         if(x->k == y->k) {
32             x->p = (x->m > y->m ? INF : -INF);
33         } else {
34             x->p = floor_div(y->m - x->m, x->k - y->k);
35         }
36         return x->p >= y->p;
37     }
38
39     void add_line(long long k, long long m) {
40         if(!MAX) {
41             k = -k;
42             m = -m;
43         }
44         auto z = insert({k, m, 0}); y = z++; x = y;
45         while(isect(y, z)) {
46             z = erase(z);
47         }
48         if(x != begin() && isect(--x, y)) {
49             isect(x, y = erase(y));
50         }
51         while((y = x) != begin() && (--x)->p >= y->p) {
52             isect(x, erase(y));
53         }
54     }
55
56     long long get(long long x) {
57         assert(!empty());
58         auto l = *lower_bound(x);
59         return (l.k * x + l.m) * (MAX ? +1 : -1);
60     }
61 };

```

## 4.13 Dinic

```

1 template<class T>
2 struct Dinic{
3     struct edge{
4         int from, to;
5         T cap;
6         edge(int _from, int _to, T _cap) : from(_from), to(_to),
            cap(_cap) {}
7     };
8     int n;
9     vector<edge> edges;
10    vector<vector<int>> g;
11    vector<int> cur, h;
12    Dinic(int _n) : n(_n+1), g(_n+1) {}
13    void add_edge(int u, int v, T cap){
14        g[u].push_back(edges.size());
15        edges.push_back(edge(u, v, cap));
16        g[v].push_back(edges.size());
17        edges.push_back(edge(v, u, 0));
18    }
19    bool bfs(int s, int t){
20        h.assign(n, -1);
21        h[s] = 0;
22        queue<int> que;
23        que.push(s);
24        while(!que.empty()) {
25            int u = que.front();
26            que.pop();
27            for(auto id : g[u]) {
28                const edge& e = edges[id];
29                int v = e.to;
30                if(e.cap > 0 && h[v] == -1) {
31                    h[v] = h[u] + 1;
32                    if(v == t) {
33                        return 1;
34                    }
35                    que.push(v);
36                }
37            }
38        }
39        return 0;
40    }
41    T dfs(int u, int t, T f) {
42        if(u == t) {
43            return f;
44        }
45        T r = f;
46        for(int& i = cur[u]; i < (int) g[u].size(); ++i) {
47            int id = g[u][i];

```

```

48            const edge& e = edges[id];
49            int v = e.to;
50            if(e.cap > 0 && h[v] == h[u] + 1) {
51                T send = dfs(v, t, min(r, e.cap));
52                edges[id].cap -= send;
53                edges[id ^ 1].cap += send;
54                r -= send;
55                if(r == 0) {
56                    return f;
57                }
58            }
59        }
60        return f - r;
61    }
62    T flow(int s, int t, T f = numeric_limits<T>::max()) {
63        T ans = 0;
64        while(f > 0 && bfs(s, t)) {
65            cur.assign(n, 0);
66            T send = dfs(s, t, f);
67            ans += send;
68            f -= send;
69        }
70        return ans;
71    }
72    vector<pair<int, int>> min_cut(int s) {
73        vector<bool> vis(n);
74        vis[s] = true;
75        queue<int> que;
76        que.push(s);
77        while(!que.empty()) {
78            int u = que.front();
79            que.pop();
80            for(auto id : g[u]) {
81                const auto& e = edges[id];
82                int v = e.to;
83                if(e.cap > 0 && !vis[v]) {
84                    vis[v] = true;
85                    que.push(v);
86                }
87            }
88        }
89        vector<pair<int, int>> cut;
90        for(int i = 0; i < (int) edges.size(); i += 2) {
91            const auto& e = edges[i];
92            if(vis[e.from] && !vis[e.to]) {
93                cut.push_back(make_pair(e.from, e.to));
94            }
95        }
96        return cut;
97    }
98 };

```

## 5 Math

### 5.1 FFT

```

1 using cd = complex<double>;
2 const double PI = acos(-1);
3
4 void FFT(vector<cd>& a, bool inv) {
5     int n = (int) a.size();
6     for(int i = 1, j = 0; i < n; ++i) {
7         int bit = n >> 1;
8         for(; j & bit; bit >>= 1) {
9             j ^= bit;
10        }
11        j ^= bit;
12        if(i < j) {
13            swap(a[i], a[j]);
14        }
15    }
16    for(int len = 2; len <= n; len <= 1) {
17        const double ang = 2 * PI / len * (inv ? -1 : +1);
18        cd rot(cos(ang), sin(ang));
19        for(int i = 0; i < n; i += len) {
20            cd w(1);
21            for(int j = 0; j < len / 2; ++j) {
22                cd u = a[i + j], v = a[i + j + len / 2] * w;
23                a[i + j] = u + v;
24                a[i + j + len / 2] = u - v;
25                w *= rot;
26            }
27        }
28    }
29    if(inv) {
30        for(auto& x : a) {
31            x /= n;
32        }
33    }
34 }
35

```

```

36 vector<int> multiply(const vector<int>& a, const vector<int>& b
37 ) {
38     vector<cd> fa(a.begin(), a.end());
39     vector<cd> fb(b.begin(), b.end());
40     int n = 1;
41     while(n < (int) a.size() + (int) b.size() - 1) {
42         n <= 1;
43     }
44     fa.resize(n);
45     fb.resize(n);
46     FFT(fa, false);
47     FFT(fb, false);
48     for(int i = 0; i < n; ++i) {
49         fa[i] *= fb[i];
50     }
51     FFT(fa, true);
52     vector<int> c(a.size() + b.size() - 1);
53     for(int i = 0; i < (int) c.size(); ++i) {
54         c[i] = round(fa[i].real());
55     }
56     return c;
57 }

```

## 5.2 Numbers

- Bernoulli numbers

$$B_0 = 1, B_1^\pm = \pm \frac{1}{2}, B_2 = \frac{1}{6}, B_3 = 0$$

$$\sum_{j=0}^m \binom{m+1}{j} B_j = 0, \text{ EGF is } B(x) = \frac{x}{e^x - 1} = \sum_{n=0}^{\infty} B_n \frac{x^n}{n!}.$$

$$S_m(n) = \sum_{k=1}^n k^m = \frac{1}{m+1} \sum_{k=0}^m \binom{m+1}{k} B_k^+ n^{m+1-k}$$

- Stirling numbers of the second kind Partitions of  $n$  distinct elements into exactly  $k$  groups.

$$S(n, k) = S(n-1, k-1) + kS(n-1, k), S(n, 1) = S(n, n) = 1$$

$$S(n, k) = \frac{1}{k!} \sum_{i=0}^k (-1)^{k-i} \binom{k}{i} i^n$$

$$x^n = \sum_{i=0}^n S(n, i) (x)_i$$

- Pentagonal number theorem

$$\prod_{n=1}^{\infty} (1 - x^n) = 1 + \sum_{k=1}^{\infty} (-1)^k \left( x^{k(3k+1)/2} + x^{k(3k-1)/2} \right)$$

- Catalan numbers

$$C_n^{(k)} = \frac{1}{(k-1)n+1} \binom{kn}{n}$$

$$C^{(k)}(x) = 1 + x[C^{(k)}(x)]^k$$

- Eulerian numbers

Number of permutations  $\pi \in S_n$  in which exactly  $k$  elements are greater than the previous element.  $k$   $j$ :s.t.  $\pi(j) > \pi(j+1)$ ,  $k+1$   $j$ :s.t.  $\pi(j) \geq j$ ,  $k$   $j$ :s.t.  $\pi(j) > j$ .

$$E(n, k) = (n-k)E(n-1, k-1) + (k+1)E(n-1, k)$$

$$E(n, 0) = E(n, n-1) = 1$$

$$E(n, k) = \sum_{j=0}^k (-1)^j \binom{n+1}{j} (k+1-j)^n$$

## 5.3 ExtendGCD

```

1 // @return $x, $y s.t. $ax + by = \gcd(a, b)$
2 #define ll long long
3 ll ext_gcd(ll a, ll b, ll& x, ll& y) {
4     if(b == 0) {
5         x = 1; y = 0;
6         return a;
7     }
8     ll x2, y2;
9     ll c = a % b;
10    if(c < 0) c += b;
11    ll g = ext_gcd(b, c, x2, y2);
12    x = y2;
13    y = x2 - (a / b) * y2;
14    return g;
15 }
16 //a^{-1} % p = x % p

```

## 5.4 InvGCD

```

1 pair<long long, long long> inv_gcd(long long a, long long b) {
2     a %= b;
3     if(a < 0) a += b;
4     if(a == 0) return {b, 0};
5     long long s = b, t = a;
6     long long m0 = 0, m1 = 1;
7     while(t) {
8         long long u = s / t;
9         s -= t * u;
10        m0 -= m1 * u;
11        swap(s, t);
12        swap(m0, m1);
13    }
14    if(m0 < 0) m0 += b / s;
15    return {s, m0};
16 }

```

## 5.5 Generating Functions

- Ordinary Generating Function  $A(x) = \sum_{i \geq 0} a_i x^i$

$$\begin{aligned}
 & - A(rx) \Rightarrow r^n a_n \\
 & - A(x) + B(x) \Rightarrow a_n + b_n \\
 & - A(x)B(x) \Rightarrow \sum_{i=0}^n a_i b_{n-i} \\
 & - A(x)^k \Rightarrow \sum_{i_1+i_2+\dots+i_k=n} a_{i_1} a_{i_2} \dots a_{i_k} \\
 & - xA(x)' \Rightarrow n a_n \\
 & - \frac{A(x)}{1-x} \Rightarrow \sum_{i=0}^n a_i
 \end{aligned}$$

- Exponential Generating Function  $A(x) = \sum_{i \geq 0} \frac{a_i}{i!} x^i$

$$\begin{aligned}
 & - A(x) + B(x) \Rightarrow a_n + b_n \\
 & - A^{(k)}(x) \Rightarrow a_n + b_n \\
 & - A(x)B(x) \Rightarrow \sum_{i=0}^n \binom{n}{i} a_i b_{n-i} \\
 & - A(x)^k \Rightarrow \sum_{i_1+i_2+\dots+i_k=n} \binom{n}{i_1, i_2, \dots, i_k} a_{i_1} a_{i_2} \dots a_{i_k} \\
 & - xA(x) \Rightarrow n a_n
 \end{aligned}$$

- Special Generating Function

$$\begin{aligned}
 & - (1+x)^n = \sum_{i \geq 0} \binom{n}{i} x^i \\
 & - \frac{1}{(1-x)^n} = \sum_{i \geq 0} \binom{n-1}{i} x^i
 \end{aligned}$$

## 5.6 XorSum

```

1 long long all_pair_xor_sum(vector<long long>v){
2     int n = v.size();
3     long long res = 0;
4     for(int i = 0; i < 64; ++i){
5         int cnt0 = 0, cnt1 = 0;
6         for(int j = 0; j < n; ++j){
7             if(v[j] & 1) cnt1++;
8             else cnt0++;
9             v[j] >>= 1;
10        }
11        res += (((1ll * cnt0 * cnt1) % mod) * ((1ll << i) % mod)) %
12               mod;
13        res %= mod;
14    }
15    return res;
16 }

```

## 5.7 Primes

```

1 /* 12721 13331 14341 75577 123457 222557 556679 999983
   1097774749 1076767633 100102021 999997771 1001010013
   1000512343 987654361 999991231 999888733 98789101 987777733
   999991921 1010101333 1010102101 100000000039
   100000000000037 2305843009213693951 4611686018427387847
   9223372036854775783 18446744073709551557 */

```

## 5.8 Theorem

- Fermat's little theorem

$$a^{m-1} \equiv 1 \pmod{m}$$

$$a^b \bmod m = (a \bmod m)^{b \bmod m-1} \bmod m$$

- Cramer's rule

$$\begin{aligned}
 ax + by &= e & x &= \frac{ed - bf}{ad - bc} \\
 cx + dy &= f & y &= \frac{af - ec}{ad - bc}
 \end{aligned}$$

- Kirchhoff's Theorem

Denote  $L$  be a  $n \times n$  matrix as the Laplacian matrix of graph  $G$ , where  $L_{ii} = d(i)$ ,  $L_{ij} = -c$  where  $c$  is the number of edge  $(i, j)$  in  $G$ .

- The number of undirected spanning in  $G$  is  $|\det(\tilde{L}_{11})|$ .
- The number of directed spanning tree rooted at  $r$  in  $G$  is  $|\det(\tilde{L}_{rr})|$ .

- Tutte's Matrix

Let  $D$  be a  $n \times n$  matrix, where  $d_{ij} = x_{ij}$  ( $x_{ij}$  is chosen uniformly at random) if  $i < j$  and  $(i, j) \in E$ , otherwise  $d_{ij} = -d_{ji}$ .  $\frac{rank(D)}{2}$  is the maximum matching on  $G$ .

- Cayley's Formula

- Given a degree sequence  $d_1, d_2, \dots, d_n$  for each labeled vertices, there are  $\frac{(d_1-1)!(d_2-1)!\dots(d_n-1)!}{(n-2)!}$  spanning trees.
- Let  $T_{n,k}$  be the number of labeled forests on  $n$  vertices with  $k$  components, such that vertex  $1, 2, \dots, k$  belong to different components. Then  $T_{n,k} = kn^{n-k-1}$ .

- Erdős–Gallai theorem

A sequence of nonnegative integers  $d_1 \geq \dots \geq d_n$  can be represented as the degree sequence of a finite simple graph on  $n$  vertices if and only if  $d_1 + \dots + d_n$

is even and  $\sum_{i=1}^k d_i \leq k(k-1) + \sum_{i=k+1}^n \min(d_i, k)$  holds for every  $1 \leq k \leq n$ .

- Gale–Ryser theorem

A pair of sequences of nonnegative integers  $a_1 \geq \dots \geq a_n$  and  $b_1, \dots, b_n$  is bigraphic if and only if  $\sum_{i=1}^n a_i = \sum_{i=1}^n b_i$  and  $\sum_{i=1}^k a_i \leq \sum_{i=1}^n \min(b_i, k)$  holds for every  $1 \leq k \leq n$ .

- Fulkerson–Chen–Anstee theorem

A sequence  $(a_1, b_1), \dots, (a_n, b_n)$  of nonnegative integer pairs with  $a_1 \geq \dots \geq a_n$  is digraphic if and only if  $\sum_{i=1}^n a_i = \sum_{i=1}^n b_i$  and  $\sum_{i=1}^k a_i \leq \sum_{i=1}^k \min(b_i, k-1) + \sum_{i=k+1}^n \min(b_i, k)$  holds for every  $1 \leq k \leq n$ .

- Möbius inversion formula

- $f(n) = \sum_{d|n} g(d) \Leftrightarrow g(n) = \sum_{d|n} \mu(d) f(\frac{n}{d})$
- $f(n) = \sum_{n|d} g(d) \Leftrightarrow g(n) = \sum_{n|d} \mu(\frac{d}{n}) f(d)$

- Spherical cap

- A portion of a sphere cut off by a plane.
- $r$ : sphere radius,  $a$ : radius of the base of the cap,  $h$ : height of the cap,  $\theta$ :  $\arcsin(a/r)$ .
- Volume =  $\pi h^2(3r - h)/3 = \pi h(3a^2 + h^2)/6 = \pi r^3(2 + \cos \theta)(1 - \cos \theta)^2/3$ .
- Area =  $2\pi rh = \pi(a^2 + h^2) = 2\pi r^2(1 - \cos \theta)$ .

## 5.9 FloorSum

```
1 //f(a, b, c, n) = \sum_{i=0}^{n-1} \lfloor \frac{ai+b}{c} \rfloor
2 long long floor_sum(long long a, long long b, long long c, long
3   long n) {
4   long long ans = 0;
5   if(a >= c) {
6     ans += (n - 1) * n * (a / c) / 2;
7     a %= c;
8   }
9   if(b >= c) {
10    ans += n * (b / c);
11    b %= c;
12  }
13  long long y_max = (a * n + b) / c;
14  long long x_max = y_max * c - b;
15  if(y_max == 0) {
16    return ans;
17  }
18  ans += (n - (x_max + a - 1) / a) * y_max;
19  return ans + floor_sum(c, (a - x_max % a) % a, a, y_max);
20 }
```

## 5.10 XorBasis

```
1 /*
2 xor_basis<60> b; // [0, 2^60)
3
4 long long x, k;
5 b.insert(x);
6 long long mn = b.get_min();
7 long long mx = b.get_max();
8 long long kth = b.get_kth(k); // 如果超過範圍回傳 -1
9 bool has_x = b.contains(x);
10
11 xor_basis<60> c;
12 b.merge(c); // 把 c 的基底合併進 b
13 */
14 namespace xor_basis_internal {
15
16 template<int B, class T>
17 struct xor_basis_helper {
18 public:
19   void insert(T x) {
20     for(int i = B - 1; i >= 0; i--) {
21       if(x >> i & 1) {
22         if(!p[i]) {
23           p[i] = x;
24           cnt += 1;
25           change = true;
26           return;
27         } else {
28           x ^= p[i];
29         }
30       }
31     }
32   }
33   if(zero == false) {
34     zero = change = true;
35   }
36 }
37
38 T get_min() {
39   if(zero) {
40     return 0;
41   }
42   for(int i = 0; i < B; i++) {
43     if(p[i]) {
44       return p[i];
45     }
46   }
47 }
48
49 T get_max() {
50   T ans = 0;
51   for(int i = B - 1; i >= 0; i--) {
52     if((ans ^ p[i]) > ans) {
53       ans ^= p[i];
54     }
55   }
56   return ans;
57 }
58
59 T get_kth(long long k) {
60   k += 1;
61   if(k == 1 && zero) {
62     return 0;
63   }
64   if(zero) {
65     k -= 1;
66   }
67   if(k >= (1LL << cnt)) {
68     return -1;
69   }
70   update();
71   T ans = 0;
72   for(int i = 0; i < (int) d.size(); i++) {
73     if(k >> i & 1) {
74       ans ^= d[i];
75     }
76   }
77   return ans;
78 }
79
80 bool contains(T x) {
81   if(x == 0) {
82     return zero;
83   }
84   for(int i = B - 1; i >= 0; i--) {
85     if(x >> i & 1) {
86       x ^= p[i];
87     }
88   }
89   return x == 0;
90 }
91
92 void merge(const xor_basis_helper& other) {
93   for(int i = 0; i < B; i++) {
94     if(other.p[i]) {
95       insert(other.p[i]);
96     }
97   }
98 }
```

```

95     }
96 }
97 }
98
99 private:
100 bool zero = false;
101 bool change = false;
102 int cnt = 0;
103 std::array<T, B> p = {};
104 std::vector<T> d;
105
106 void update() {
107     if(!change) {
108         return;
109     }
110     change = false;
111     d.clear();
112     for(int j = 0; j < B; j++) {
113         for(int i = j - 1; i >= 0; i--) {
114             if(p[j] >> i & 1) {
115                 p[j] ^= p[i];
116             }
117         }
118     }
119     for(int i = 0; i < B; i++) {
120         if(p[i]) {
121             d.push_back(p[i]);
122         }
123     }
124 }
125 };
126
127 } // namespace xor_basis_internal
128
129
130 template<int B, class ENABLE = void> struct xor_basis : public
    xor_basis_internal::xor_basis_helper<B, __int128> {};
131 template<int B> struct xor_basis<B, std::enable_if_t<(B >= 32
    && B < 64)>> : public xor_basis_internal::xor_basis_helper<
    B, long long> {};
132 template<int B> struct xor_basis<B, std::enable_if_t<(B >= 16
    && B < 32)>> : public xor_basis_internal::xor_basis_helper<
    B, int> {};
133 template<int B> struct xor_basis<B, std::enable_if_t<(B >= 8 &&
    B < 16)>> : public xor_basis_internal::xor_basis_helper<B,
    short> {};
134 template<int B> struct xor_basis<B, std::enable_if_t<(B < 8)>>
    : public xor_basis_internal::xor_basis_helper<B, int8_t>
    {};

```

## 5.11 GuessKth

```

1 template <typename Tfield>
2 std::pair<int, std::vector<Tfield>> find_linear_recurrence(
3     const std::vector<Tfield> &S) {
4     int N = S.size();
5     using poly = std::vector<Tfield>;
6     poly C_reversed{1}, B{1};
7     int L = 0, m = 1;
8     Tfield b = 1;
9
10    // adjust: C(x) <- C(x) - (d / b) x^m B(x)
11    auto adjust = [](poly C, const poly &B, Tfield d, Tfield b,
12        int m) -> poly {
13        C.resize(std::max(C.size(), B.size() + m));
14        Tfield a = d / b;
15        for (unsigned i = 0; i < B.size(); i++) C[i + m] -= a *
16            B[i];
17        return C;
18    };
19
20    for (int n = 0; n < N; n++) {
21        Tfield d = S[n];
22        for (int i = 1; i <= L; i++) d += C_reversed[i] * S[n -
23            i];
24
25        if (d == 0)
26            m++;
27        else if (2 * L <= n) {
28            poly T = C_reversed;
29            C_reversed = adjust(C_reversed, B, d, b, m);
30            L = n + 1 - L;
31            B = T;
32            b = d;
33            m = 1;
34        } else
35            C_reversed = adjust(C_reversed, B, d, b, m++);
36    }
37    return std::make_pair(L, C_reversed);
38 }
39
40 // Calculate $x^N \bmod f(x)$
41 // Known as `Kitamasa method`
42 // Input: f_reversed: monic, reversed (f_reversed[0] = 1)

```

```

39 // Complexity: $O(K^2 \log N)$ ($K$: deg. of $f$)
40 // Example: (4, [1, -1, -1]) -> [2, 3]
41 // (x^4 = (x^2 + x + 2)(x^2 - x - 1) + 3x + 2)
42 // Reference: http://misawa.github.io/others/
43 // http://sugarknri.hatenablog.com/entry/2017/11/18/233936
44 template <typename Tfield>
45 std::vector<Tfield> monomial_mod_polynomial(long long N, const
    std::vector<Tfield> &f_reversed) {
46     assert(!f_reversed.empty() and f_reversed[0] == 1);
47     int K = f_reversed.size() - 1;
48     if (!K) return {};
49     int D = 64 - __builtin_clzll(N);
50     std::vector<Tfield> ret(K, 0);
51     ret[0] = 1;
52     auto self_conv = [](std::vector<Tfield> x) -> std::vector<
    Tfield> {
53         int d = x.size();
54         std::vector<Tfield> ret(d * 2 - 1);
55         for (int i = 0; i < d; i++) {
56             ret[i * 2] += x[i] * x[i];
57             for (int j = 0; j < i; j++) ret[i + j] += x[i] * x[
58                 j] * 2;
59         }
60         return ret;
61     };
62     for (int d = D; d--;) {
63         ret = self_conv(ret);
64         for (int i = 2 * K - 2; i >= K; i--) {
65             for (int j = 1; j <= K; j++) ret[i - j] -= ret[i] *
66                 f_reversed[j];
67         }
68         ret.resize(K);
69         if ((N >> d) & 1) {
70             std::vector<Tfield> c(K);
71             c[0] = -ret[K - 1] * f_reversed[K];
72             for (int i = 1; i < K; i++) { c[i] = ret[i - 1] -
73                 ret[K - 1] * f_reversed[K - i]; }
74             ret = c;
75         }
76     }
77     return ret;
78 }
79 // Guess k-th element of the sequence, assuming linear
80 // recurrence
81 // initial_elements: 0-ORIGIN
82 // Verify: abc198f https://atcoder.jp/contests/abc198/
83 // submissions/21837815
84 template <typename Tfield>
85 Tfield guess_kth_term(const std::vector<Tfield> &
    initial_elements, long long k) {
86     assert(k >= 0);
87     if (k < static_cast<long long>(initial_elements.size()))
88         return initial_elements[k];
89     const auto f = find_linear_recurrence<Tfield>(
90         initial_elements).second;
91     const auto g = monomial_mod_polynomial<Tfield>(k, f);
92     Tfield ret = 0;
93     for (unsigned i = 0; i < g.size(); i++) ret += g[i] *
94         initial_elements[i];
95     return ret;
96 }

```

## 5.12 PowMod

```

1 constexpr long long Pow(long long x, long long n, int m) {
2     if(m == 1) return 0;
3     unsigned int _m = (unsigned int)(m);
4     unsigned long long r = 1;
5     x %= m;
6     if(x < 0) x += m;
7     unsigned long long y = x;
8     while(n) {
9         if(n & 1) r = (r * y) % _m;
10        y = (y * y) % _m;
11        n >>= 1;
12    }
13    return r;
14 }

```

## 5.13 ModInt

```

1 template<int id>
2 struct modint {
3 public:
4     static constexpr int mod() { return id; }
5
6     constexpr modint() : value(0) {}
7     modint(long long x) : value(x % mod()) {}

```

```

8     if(value < 0) value += mod();
9 }
10
11 constexpr int val() const { return value; }
12
13 constexpr modint inv() const {
14     return Pow(value, mod()-2, mod());
15 }
16
17 constexpr modint& operator+=(const modint& rhs) & {
18     value += rhs.value;
19     if(value >= mod()) {
20         value -= mod();
21     }
22     return *this;
23 }
24
25 constexpr modint& operator-=(const modint& rhs) & {
26     value -= rhs.value;
27     if(value < 0) {
28         value += mod();
29     }
30     return *this;
31 }
32
33 constexpr modint& operator*=(const modint& rhs) & {
34     value = 1LL * value * rhs.value % mod();
35     return *this;
36 }
37
38 constexpr modint& operator/=(const modint& rhs) & {
39     return *this *= rhs.inv();
40 }
41
42 friend constexpr modint operator+(modint lhs, modint rhs) {
43     return lhs += rhs; }
44 friend constexpr modint operator-(modint lhs, modint rhs) {
45     return lhs -= rhs; }
46 friend constexpr modint operator*(modint lhs, modint rhs) {
47     return lhs *= rhs; }
48 friend constexpr modint operator/(modint lhs, modint rhs) {
49     return lhs /= rhs; }
50
51 constexpr modint operator+() const { return *this; }
52 constexpr modint operator-() const { return modint() - *this; }
53
54 constexpr bool operator==(const modint& rhs) const { return
    value == rhs.value; }
55 constexpr bool operator!=(const modint& rhs) const { return
    value != rhs.value; }
56
57 int value;
58 };
59 using mint = modint<mod>;

```

## 5.14 CRT

```

1 // #include "InvGCD.h"
2 // @return
3 // $\text{remainder, modulo}$
4 // or
5 // $0$, or $if do not exist$
6 pair<long long, long long> crt(const vector<long long>& r,
7     const vector<long long>& m) {
8     assert(r.size()==m.size());
9     int n = r.size();
10    // Contracts: 0 <= r0 < m0
11    long long r0 = 0, m0 = 1;
12    for(int i = 0; i < n; i++) {
13        assert(1 <= m[i]);
14        long long r1 = r[i] % m[i];
15        if(r1 < 0) r1 += m[i];
16        long long m1 = m[i];
17        if(m0 < m1) {
18            swap(r0, r1);
19            swap(m0, m1);
20        }
21        if(m0 % m1 == 0) {
22            if(r0 % m1 != r1) return {0, 0};
23            continue;
24        }
25        long long g, im;
26        tie(g, im) = inv_gcd(m0, m1);
27        long long u1 = (m1 / g);
28        if((r1 - r0) % g) return {0, 0};
29        long long x = (r1 - r0) / g % u1 * im % u1;
30        r0 += x * m0;
31        m0 *= u1;
32        if(r0 < 0) r0 += m0;
33    }
34    return {r0, m0};
35 }

```

## 5.15 DiscreteLog

```

1 //give you $a, b, m$ find $x$ such that $a^x \equiv b \pmod m$
2 #line 2 "Library/math/discrete-log.hpp"
3 #include <vector>
4 #include <cmath>
5 #include <cassert>
6 #line 2 "Library/data-structure/pbds.hpp"
7 #include <ext/pb_ds/assoc_container.hpp>
8 #line 2 "Library/random/splitmix64.hpp"
9 #include <chrono>
10
11 namespace felix {
12
13 namespace internal {
14
15 struct splitmix64_hash {
16     // http://xoshiro.di.unimi.it/splitmix64.c
17
18     static unsigned long long splitmix64(unsigned long long x) {
19         x += 0x9e3779b97f4a7c15;
20         x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9;
21         x = (x ^ (x >> 27)) * 0x94d049bb133111eb;
22         return x ^ (x >> 31);
23     }
24
25     unsigned long long operator()(unsigned long long x) const {
26         static const unsigned long long FIXED_RANDOM = std::chrono
27             ::steady_clock::now().time_since_epoch().count();
28         return splitmix64(x + FIXED_RANDOM);
29     }
30 };
31 } // namespace internal
32
33 } // namespace felix
34
35 #line 4 "Library/data-structure/pbds.hpp"
36
37 namespace felix {
38
39 template<class T, class U, class H = internal::splitmix64_hash>
40     using hash_map = __gnu_pbds::gp_hash_table<T, U, H>;
41 template<class T, class H = internal::splitmix64_hash> using
42     hash_set = hash_map<T, __gnu_pbds::null_type, H>;
43
44 template<class T, class Comp = std::less<T>> using ordered_set
45     = __gnu_pbds::tree<T, __gnu_pbds::null_type, Comp,
46         __gnu_pbds::rb_tree_tag, __gnu_pbds::
47         tree_order_statistics_node_update>;
48 template<class T> using ordered_multiset = ordered_set<T, std::
49     less_equal<T>>;
50
51 } // namespace felix
52 #line 2 "Library/modint/barrett.hpp"
53
54 namespace felix {
55
56 namespace internal {
57
58 // Fast modular multiplication by barrett reduction
59 // Reference: https://en.wikipedia.org/wiki/Barrett_reduction
60
61 struct barrett {
62     unsigned int m;
63     unsigned long long im;
64
65     explicit barrett(unsigned int _m) : m(_m), im((unsigned long
66         long)(-1) / _m + 1) {}
67
68     unsigned int umod() const { return m; }
69
70     unsigned int mul(unsigned int a, unsigned int b) const {
71         unsigned long long z = a;
72         z *= b;
73 #ifdef _MSC_VER
74         unsigned long long x;
75         _umul128(z, im, &x);
76 #else
77         unsigned long long x = (unsigned long long)(((unsigned
78             __int128)(z) * im) >> 64);
79 #endif
80         unsigned long long y = x * m;
81         return (unsigned int)(z - y + (z < y ? m : 0));
82     }
83 };
84 } // namespace internal
85
86 } // namespace felix
87
88 #line 2 "Library/math/binary-gcd.hpp"

```



```

86 namespace felix {
87
88 template<class T>
89 inline T binary_gcd(T a, T b) {
90     if(a == 0 || b == 0) {
91         return a | b;
92     }
93     int8_t n = __builtin_ctzll(a);
94     int8_t m = __builtin_ctzll(b);
95     a >>= n;
96     b >>= m;
97     while(a != b) {
98         T d = a - b;
99         int8_t s = __builtin_ctzll(d);
100         bool f = a > b;
101         b = f ? b : a;
102         a = (f ? d : -d) >> s;
103     }
104     return a << (n < m ? n : m);
105 }
106
107 } // namespace felix
108
109 #line 8 "Library/math/discrete-Log.hpp"
110
111 namespace felix {
112
113 int discrete_log(int a, int b, int m) {
114     assert(b < m);
115     if(b == 1 || m == 1) {
116         return 0;
117     }
118     int n = (int) std::sqrt(m) + 1, e = 1, f = 1, j = 1;
119     hash_map<int, int> baby;
120     internal::barrett bt(m);
121     while(j <= n && (e = f = bt.mul(e, a)) != b) {
122         baby[bt.mul(e, b)] = j++;
123     }
124     if(e == b) {
125         return j;
126     }
127     if(binary_gcd(m, e) == binary_gcd(m, b)) {
128         for(int i = 2; i < n + 2; i++) {
129             e = bt.mul(e, f);
130             if(baby.find(e) != baby.end()) {
131                 return n * i - baby[e];
132             }
133         }
134     }
135     return -1;
136 }
137
138 } // namespace felix

```

## 5.16 LinearSieve

```

1 vector<bool> is_prime;
2 vector<int> primes, phi, mobius, least;
3 void linear_sieve(int n) {
4     n += 1;
5     is_prime.resize(n);
6     least.resize(n);
7     fill(2 + begin(is_prime), end(is_prime), true);
8     phi.resize(n); mobius.resize(n);
9     phi[1] = mobius[1] = 1;
10    least[0] = 0, least[1] = 1;
11    for(int i = 2; i < n; ++i) {
12        if(is_prime[i]) {
13            primes.push_back(i);
14            phi[i] = i - 1;
15            mobius[i] = -1;
16            least[i] = i;
17        }
18        for(auto j : primes) {
19            if(i * j >= n) break;
20            is_prime[i * j] = false;
21            least[i * j] = j;
22            if(i % j == 0) {
23                mobius[i * j] = 0;
24                phi[i * j] = phi[i] * j;
25                break;
26            } else {
27                mobius[i * j] = mobius[i] * mobius[j];
28                phi[i * j] = phi[i] * phi[j];
29            }
30        }
31    }
32 }

```

## 6 Misc

### 6.1 FastIO

```

1 inline char gc() {
2     static const int BUF_SIZE = 1 << 22;
3     static int Counts = 1 << 23;
4     static char Buffer[BUF_SIZE];
5     static char *Pointer = Buffer, *End = Buffer;
6     if(Pointer == End) {
7         if(Counts < BUF_SIZE) {
8             return EOF;
9         }
10        Counts = fread(Buffer, 1, BUF_SIZE, stdin);
11        Pointer = Buffer;
12        End = Buffer + Counts;
13    }
14    return *(Pointer++);
15 }
16
17 template<class T>
18 inline void read(T& x) {
19     static char c;
20     do {
21         c = gc();
22     } while(c < '0' && c != '-');
23     bool neg = (c == '-');
24     if(!neg) {
25         x = c - '0';
26     } else x = 0;
27     while((c = gc()) >= '0') {
28         x = (x << 3) + (x << 1) + (c & 15);
29     }
30     if(neg) {
31         x = -x;
32     }
33 }
34
35 template<class T, class... U>
36 inline void read(T& a, U&... b) {
37     read(a);
38     read(b...);
39 }
40
41 template<class T>
42 inline void write(T temp, char end = '\n') {
43     static short digits[20], P;
44     if(temp == 0) {
45         putchar_unlocked('0');
46         putchar_unlocked(end);
47         return;
48     }
49     if(temp < 0) {
50         putchar_unlocked('-');
51         write(-temp, end);
52         return;
53     }
54     P = -1;
55     while(temp) {
56         digits[++P] = temp % 10;
57         temp /= 10;
58     }
59     while(P >= 0) {
60         putchar_unlocked(digits[P--] + '0');
61     }
62     putchar_unlocked(end);
63     return;
64 }

```

### 6.2 Debug

```

1 #ifndef LOCAL
2     #define eprintf(...) { fprintf(stderr, __VA_ARGS__); fflush(
3         stderr); }
4 #else
5     #define eprintf(...) 42
6 #endif

```

### 6.3 Discrete

```

1 template<class T>
2 vector<int> Discrete(const vector<T>&v){
3     vector<int> ans;
4     vector<T> tmp(v);
5     sort(begin(tmp), end(tmp));
6     tmp.erase(unique(begin(tmp), end(tmp)), end(tmp));
7     for(auto i:v) ans.push_back(lower_bound(begin(tmp), end(tmp), i)
8         -tmp.begin()+1);

```

```
8 | return ans;
9 | }
```

## 6.4 DuiPai

```
1 | #include <bits/stdc++.h>
2 | using namespace std;
3 | int main(){
4 |     string sol,bf,make;
5 |     cout<<"Your solution file name :";
6 |     cin>>sol;
7 |     cout<<"Brute force file name :";
8 |     cin>>bf;
9 |     cout<<"Make data file name :";
10 |    cin>>make;
11 |    system(("g++ "+sol+" -o sol").c_str());
12 |    system(("g++ "+bf+" -o bf").c_str());
13 |    system(("g++ "+make+" -o make").c_str());
14 |    for(int t = 0;t<10000;++t){
15 |        system("./make > ./1.in");
16 |        double st = clock();
17 |        system("./sol < ./1.in > ./1.ans");
18 |        double et = clock();
19 |        system("./bf < ./1.in > ./1.out");
20 |        if(system("diff ./1.out ./1.ans")) {
21 |            printf("\033[0;31mWrong Answer\033[0m on test #%d",t);
22 |            return 0;
23 |        }
24 |        else if(et-st>=2000){
25 |            printf("\033[0;32mTime Limit exceeded\033[0m on test #%d,
26 |                Time %.0lfms\n",t,et-st);
27 |            return 0;
28 |        }
29 |        else {
30 |            printf("\033[0;32mAccepted\033[0m on test #%d, Time
31 |                %.0lfms\n", t, et - st);
32 |        }
33 |    }
34 | }
```

## 6.5 Timer

```
1 | const clock_t startTime = clock();
2 | inline double getCurrentTime() {
3 |     return (double) (clock() - startTime) / CLOCKS_PER_SEC;
4 | }
```

## 6.6 TernarySearch

```
1 | // return the maximum of f(x) in [l, r]
2 | double ternary_search(double l, double r) {
3 |     while(r - l > EPS) {
4 |         double m1 = l + (r - l) / 3;
5 |         double m2 = r - (r - l) / 3;
6 |         double f1 = f(m1), f2 = f(m2);
7 |         if(f1 < f2) l = m1;
8 |         else r = m2;
9 |     }
10 |    return f(l);
11 | }
12 |
13 | // return the maximum of f(x) in [l, r]
14 | int ternary_search(int l, int r) {
15 |     while(r - l > 1) {
16 |         int mid = (l + r) / 2;
17 |         if(f(m) > f(m + 1)) r = m;
18 |         else l = m;
19 |     }
20 |    return r;
21 | }
```

# 7 Setup

## 7.1 vimrc

```
1 | se nu ai hls et ru ic is sc cul
2 | se re=1 ts=4 sts=4 sw=4 ls=2 mouse=a
3 | syntax on
4 | hi cursorline cterm=none ctermbg=89
5 | set bg=dark
6 | autocmd vimEnter *.cpp map <F5> :w <CR> :!clear ; g++ -g --std=
7 |     c++17 % && echo Compiled successfully. && time ./a.out; <
8 |     CR>
```

## 7.2 Template

```
1 | #include <bits/extc++.h>
2 | #include <bits/stdc++.h>
3 | #pragma GCC optimize("O3,unroll-loops")
4 | #pragma GCC target("avx2,bmi,bmi2,lzcnt,popcnt")
5 | #define IOS ios::sync_with_stdio(0),cin.tie(0),cout.tie(0)
6 | #define int long long
7 | #define double long double
8 | #define pb push_back
9 | #define sz(x) (int)(x).size()
10 | #define all(v) begin(v),end(v)
11 | #define debug(x) cerr<<"#x<<" = "<<x<<"'\n'
12 | #define LINE cout<<"\n-----\n"
13 | #define endl '\n'
14 | #define VI vector<int>
15 | #define F first
16 | #define S second
17 | #define MP(a,b) make_pair(a,b)
18 | #define rep(i,m,n) for(int i = m;i<=n;++i)
19 | #define res(i,m,n) for(int i = m;i>=n;--i)
20 | #define gcd(a,b) __gcd(a,b)
21 | #define lcm(a,b) a*b/gcd(a,b)
22 | #define Case() int _;cin>>_;for(int Case = 1;Case<=_;++Case)
23 | #define pii pair<int,int>
24 | using namespace __gnu_cxx;
25 | using namespace __gnu_pbds;
26 | using namespace std;
27 | template <typename K, typename cmp = less<K>, typename T =
28 |     thin_heap_tag> using _heap = __gnu_pbds::priority_queue<K,
29 |     cmp, T>;
30 | template <typename K, typename M = null_type> using _hash =
31 |     gp_hash_table<K, M>;
32 | const int N = 1e6+5,L = 20,mod = 1e9+7;
33 | const long long inf = 2e18+5;
34 | const double eps = 1e-7,pi = acos(-1);
35 | void solve(){
36 | }
37 | signed main(){
38 |     IOS;
39 |     solve();
40 | }
```

# 8 String

## 8.1 RollingHash

```
1 | template<int HASH_COUNT>
2 | struct RollingHash {
3 |
4 |     static const int MAX_HASH_PAIRS = 10;
5 |
6 |     // {mul, mod}
7 |     const vector<pair<int, int>> HASH_PAIRS = {{827167801,
8 |         999999937}, {998244353, 999999929}, {146672737,
9 |         922722049}, {204924373, 952311013}, {585761567,
10 |        955873937}, {484547929, 901981687}, {856009481,
11 |        987877511}, {852853249, 996724213}, {937381759,
12 |        994523539}, {116508269, 993179543}};
13 |
14 |     int n;
15 |     vector<int> POW[MAX_HASH_PAIRS];
16 |     array<vector<int>, HASH_COUNT> pref;
17 |
18 |     int substr(int k, int l, int r) {
19 |         const auto& p = HASH_PAIRS[k];
20 |         if(l == r) {
21 |             return 0;
22 |         }
23 |         int res = pref[k][r - 1];
24 |         if(l > 0) {
25 |             res -= 1LL * pref[k][l - 1] * get_power(k, r - l) % p.
26 |                 second;
27 |         }
28 |         if(res < 0) {
29 |             res += p.second;
30 |         }
31 |         return res;
32 |     }
33 |
34 |     // build powers up to x^k
35 |     void build_powers(int k) {
36 |         for(int i = 0; i < HASH_COUNT; ++i) {
37 |             const auto& p = HASH_PAIRS[i];
38 |             int sz = (int) POW[i].size();
39 |             if(sz > k) {
40 |                 continue;
41 |             }
42 |             if(sz == 0) {
```

```

37     POW[i].push_back(1);
38     sz = 1;
39 }
40 while(sz <= k) {
41     POW[i].push_back(1LL * POW[i].back() * p.first % p.
42         second);
43     sz += 1;
44 }
45 }
46
47 int get_power(int a, int b) {
48     build_powers(b);
49     return POW[a][b];
50 }
51
52 RollingHash() : RollingHash("") {}
53
54 RollingHash(const string& s) : n(s.size()) {
55     //static_assert(HASH_COUNT > 0 && HASH_COUNT <=
56         MAX_HASH_PAIRS);
57     for(int i = 0; i < HASH_COUNT; ++i) {
58         const auto& p = HASH_PAIRS[i];
59         pref[i].resize(n);
60         pref[i][0] = s[0];
61         for(int j = 1; j < n; ++j) {
62             pref[i][j] = (1LL * pref[i][j - 1] * p.first + s[j]) %
63                 p.second;
64         }
65     }
66     build_powers(n);
67
68 void add_char(char c) {
69     for(int i = 0; i < HASH_COUNT; ++i) {
70         const auto& p = HASH_PAIRS[i];
71         pref[i].push_back((1LL * (n == 0 ? 0 : pref[i].back()) *
72             p.first + c) % p.second);
73     }
74     n += 1;
75     build_powers(n);
76 }
77
78 // Return hash values for [l, r)
79 array<int, HASH_COUNT> substr(int l, int r) {
80     array<int, HASH_COUNT> res{};
81     for(int i = 0; i < HASH_COUNT; ++i) {
82         res[i] = substr(i, l, r);
83     }
84     return res;
85 }
86
87 array<int, HASH_COUNT> merge(const vector<pair<int, int>>&
88     seg) {
89     array<int, HASH_COUNT> res{};
90     for(int i = 0; i < HASH_COUNT; ++i) {
91         const auto& p = HASH_PAIRS[i];
92         for(auto [l, r] : seg) {
93             res[i] = (1LL * res[i] * get_power(i, r - l) + substr(i,
94                 l, r)) % p.second;
95         }
96     }
97     return res;
98 }
99
100 inline int size() const {
101     return n;
102 }
103 };

```

## 8.2 Z

```

1 //z[i] := LCP(s, s[i, n)), z[0] is dont care
2 template<class T>
3 vector<int> Z(const vector<T>& a) {
4     int n = (int) a.size();
5     vector<int> z(n);
6     for(int i = 1, j = 0; i < n; ++i) {
7         if(i <= j + z[j]) {
8             z[i] = min(z[i - j], j + z[j] - i);
9         }
10        while(i + z[i] < n && a[i + z[i]] == a[z[i]]) {
11            z[i] += 1;
12        }
13        if(i + z[i] > j + z[j]) {
14            j = i;
15        }
16    }
17    return z;
18 }
19
20 vector<int> Z(const string& s) {
21     return Z(vector<int>(s.begin(), s.end()));
22 }

```

## 8.3 KMP

```

1 template<class T>
2 vector<int> KMP(const vector<T>& a) {
3     int n = (int) a.size();
4     vector<int> k(n);
5     for(int i = 1; i < n; ++i) {
6         int j = k[i - 1];
7         while(j > 0 && a[i] != a[j]) {
8             j = k[j - 1];
9         }
10        j += (a[i] == a[j]);
11        k[i] = j;
12    }
13    return k;
14 }
15
16 vector<int> KMP(const std::string& s) {
17     return KMP(vector<int>(s.begin(), s.end()));
18 }

```

## 8.4 SuffixArray

```

1 struct suffix_array{
2     int n;
3     vector<int> SA, Rank, LCP;
4     void counting_sort(vector<int>&v, auto getkey){
5         int n = 0;
6         for(auto i:v) n = max(n, getkey(i)+1);
7         vector<int> bucket(n, ans(v.size()));
8         for(auto i:v) ++bucket[getkey(i)];
9         partial_sum(begin(bucket), end(bucket), begin(bucket));
10        for(auto ite = v.rbegin(); ite!=v.rend(); ++ite) ans[--bucket[
11            getkey(*ite)]] = move(*ite);
12        v.swap(ans);
13        return;
14    }
15    suffix_array(string s):n(s.size()){
16        SA.resize(n), Rank.resize(n), LCP.resize(n);
17        for(int i = 0; i<n; ++i) SA[i] = i;
18        sort(SA.begin(), SA.end(), [&](int a, int b){
19            return s[a]<s[b];
20        });
21        for(int i = 0; i<n; ++i){
22            Rank[SA[i]] = (i?Rank[SA[i-1]]+(s[SA[i]]!=s[SA[i-1]]):SA
23                [0]);
24        }
25        for(int k = 0; (1<<k)<=n; ++k){
26            vector<int> idx;
27            for(int i = n-(1<<k); i<n; ++i) idx.push_back(i);
28            for(auto i:SA) if(i>=(1<<k)) idx.push_back(i-(1<<k));
29            counting_sort(idx, [&](int a){return Rank[a];});
30            SA.swap(idx);
31            vector<int> new_rank(n);
32            new_rank[SA[0]] = 0;
33            for(int i = 1; i<n; ++i){
34                auto cmp = [&](int a, int b){
35                    return Rank[a]!=Rank[b] or a+(1<<k)>=n or Rank[a+(1<<
36                        k)]!=Rank[b+(1<<k)];
37                };
38                new_rank[SA[i]] = new_rank[SA[i-1]]+cmp(SA[i-1], SA[i]);
39            }
40            Rank.swap(new_rank);
41        }
42        for(int i = 0, k = 0; i<n; ++i){
43            if(Rank[i]==0) continue;
44            if(k--<0) k=0;
45            while(i+k<n and SA[Rank[i]-1]+k<n and s[i+k]==s[SA[Rank[i]
46                ]-1+k])) ++k;
47            LCP[Rank[i]] = k;
48        }
49    }
50 };

```

## 8.5 Trie

```

1 template<int ALPHABET = 26, char MIN_CHAR = 'a'>
2 class trie {
3 public:
4     struct Node {
5         int go[ALPHABET];
6         Node() {
7             memset(go, -1, sizeof(go));
8         }
9     };
10
11    trie() {
12        newNode();
13    }

```

```

13 | }
14 |
15 | inline int next(int p, int v) {
16 |     return nodes[p].go[v] != -1 ? nodes[p].go[v] : nodes[p].go[
17 |         v] = newNode();
18 | }
19 | inline void insert(const vector<int>& a, int p = 0) {
20 |     for(int v : a) {
21 |         p = next(p, v);
22 |     }
23 | }
24 |
25 | inline void clear() {
26 |     nodes.clear();
27 |     newNode();
28 | }
29 |
30 | inline int longest_common_prefix(const vector<int>& a, int p
31 |     = 0) const {
32 |     int ans = 0;
33 |     for(int v : a) {
34 |         if(nodes[p].go[v] != -1) {
35 |             ans += 1;
36 |             p = nodes[p].go[v];
37 |         } else {
38 |             break;
39 |         }
40 |     }
41 |     return ans;
42 | }
43 | private:
44 |     vector<Node> nodes;
45 |
46 |     inline int newNode() {
47 |         nodes.emplace_back();
48 |         return (int) nodes.size() - 1;
49 |     }
50 | };

```

## 8.6 MinimalRotation

```

1 | string small_rot(string s) {
2 |     int n = sz(s), i = 0, j = 1;
3 |     s += s;
4 |     while(i < n && j < n) {
5 |         int k = 0;
6 |         while(k < n && s[i + k] == s[j + k]) k++;
7 |         if(s[i + k] <= s[j + k]) j += k + 1;
8 |         else i += k + 1;
9 |         if(i == j) j++;
10 |     }
11 |     int ans = i < n ? i : j;
12 |     return s.substr(ans, n);
13 | }

```

# ACM ICPC Team

## Reference -

### LeeJiaHuaPlayMinecraft

#### Contents

<b>1 CSES</b>	<b>1</b>
1.1 CountingTilings . . . . .	1
1.2 Sequence1 . . . . .	1
1.3 JosephusQueries . . . . .	1
1.4 AnotherGame . . . . .	1
1.5 CountingCoprimePairs . . . . .	1
1.6 Sequence2 . . . . .	2
1.7 LongestPalindrome . . . . .	2
1.8 DistinctSubstrings . . . . .	2
1.9 BracketSequencesII . . . . .	3
1.10 CountingNumbers . . . . .	3
1.11 WordCombinations . . . . .	3
1.12 SumOfDivisors . . . . .	4
1.13 FindingPatterns . . . . .	4
1.14 RemovalGame . . . . .	5
1.15 MinimalRotation . . . . .	5
1.16 Dice . . . . .	5
<b>2 Data-Structure</b>	<b>5</b>
2.1 Treap . . . . .	5
2.2 Segtree . . . . .	6
2.3 BinaryTrie . . . . .	7
2.4 DsuUndo . . . . .	7
2.5 CDQ . . . . .	7
2.6 MaximumXorSum . . . . .	8
2.7 DSU . . . . .	8
2.8 LiChao . . . . .	8
2.9 Fenwick . . . . .	8
2.10 FastSet . . . . .	9
2.11 Persistent DSU . . . . .	9
2.12 TimingSegtree . . . . .	9
2.13 AreaOfRectangles . . . . .	10
2.14 SparseTable . . . . .	10
2.15 DynamicSegtree . . . . .	10
2.16 ZkwSegtree . . . . .	10
2.17 MoAlgo . . . . .	11
2.18 Hash . . . . .	11
2.19 RedBlackTree . . . . .	11
<b>3 Geometry</b>	<b>11</b>
3.1 GeometryTheorem . . . . .	11
3.2 PointInPolygon . . . . .	11
3.3 PointInConvex . . . . .	12
3.4 MaximumDistance . . . . .	12

3.5 PolarAngleSort . . . . .	12
3.6 MinimumDistance . . . . .	12
3.7 ConvexHull . . . . .	12
3.8 Template . . . . .	12
<b>4 Graph</b>	<b>13</b>
4.1 HLD . . . . .	13
4.2 Bridges . . . . .	13
4.3 FlowTheorems . . . . .	13
4.4 TwoSat . . . . .	13
4.5 MCMF . . . . .	13
4.6 LCA . . . . .	14
4.7 CentroidDecomposition . . . . .	14
4.8 BCC AP . . . . .	14
4.9 CentroidTree . . . . .	15
4.10 SCC . . . . .	15
4.11 TriangleSum . . . . .	15
4.12 LineContainer . . . . .	15
4.13 Dinic . . . . .	16
<b>5 Math</b>	<b>16</b>
5.1 FFT . . . . .	16
5.2 Numbers . . . . .	17
5.3 ExtendGCD . . . . .	17
5.4 InvGCD . . . . .	17
5.5 GeneratingFunctions . . . . .	17
5.6 XorSum . . . . .	17
5.7 Primes . . . . .	17
5.8 Theorem . . . . .	17
5.9 FloorSum . . . . .	18
5.10 XorBasis . . . . .	18
5.11 GuessKth . . . . .	19
5.12 PowMod . . . . .	19
5.13 ModInt . . . . .	19
5.14 CRT . . . . .	20
5.15 DiscreteLog . . . . .	20
5.16 LinearSieve . . . . .	21
<b>6 Misc</b>	<b>21</b>
6.1 FastIO . . . . .	21
6.2 Debug . . . . .	21
6.3 Discrete . . . . .	21
6.4 DuiPai . . . . .	22
6.5 Timer . . . . .	22
6.6 TenarySearch . . . . .	22
<b>7 Setup</b>	<b>22</b>
7.1 vimrc . . . . .	22
7.2 Template . . . . .	22
<b>8 String</b>	<b>22</b>
8.1 RollingHash . . . . .	22
8.2 Z . . . . .	23
8.3 KMP . . . . .	23
8.4 SuffixArray . . . . .	23
8.5 Trie . . . . .	23
8.6 MinimalRotation . . . . .	24



# ACM ICPC Judge Test - LeeJiaHuaPlayMinecraft

## C++ Resource Test

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 namespace system_test {
5
6     const size_t KB = 1024;
7     const size_t MB = KB * 1024;
8     const size_t GB = MB * 1024;
9
10    size_t block_size, bound;
11    void stack_size_dfs(size_t depth = 1) {
12        if (depth >= bound)
13            return;
14        int8_t ptr[block_size]; // 若無法編譯將 block_size 改成常數
15        memset(ptr, 'a', block_size);
16        cout << depth << endl;
17        stack_size_dfs(depth + 1);
18    }
19
20    void stack_size_and_runtime_error(size_t block_size, size_t
        bound = 1024) {
21        system_test::block_size = block_size;
22        system_test::bound = bound;
23        stack_size_dfs();
24    }
25
26    double speed(int iter_num) {
27        const int block_size = 1024;
28        volatile int A[block_size];
29        auto begin = chrono::high_resolution_clock::now();
30        while (iter_num--)
31            for (int j = 0; j < block_size; ++j)
32                A[j] += j;
33        auto end = chrono::high_resolution_clock::now();
34        chrono::duration<double> diff = end - begin;
35        return diff.count();
36    }
37
38    void runtime_error_1() {
39        // Segmentation fault
40        int *ptr = nullptr;
41        *(ptr + 7122) = 7122;
42    }
43
44    void runtime_error_2() {
45        // Segmentation fault
46        int *ptr = (int *)memset;
47        *ptr = 7122;
48    }
49
50    void runtime_error_3() {
51        // munmap_chunk(): invalid pointer
52        int *ptr = (int *)memset;
53        delete ptr;
54    }
55
56    void runtime_error_4() {
57        // free(): invalid pointer
58        int *ptr = new int[7122];
59        ptr += 1;
60        delete[] ptr;
61    }
62
63    void runtime_error_5() {
64        // maybe illegal instruction
65        int a = 7122, b = 0;
66        cout << (a / b) << endl;
67    }
68
69    void runtime_error_6() {
70        // floating point exception
71        volatile int a = 7122, b = 0;
72        cout << (a / b) << endl;
73    }
74
75    void runtime_error_7() {
76        // call to abort.
77        assert(false);
78    }
79
80 } // namespace system_test
81
82 #include <sys/resource.h>
83 void print_stack_limit() { // only work in Linux
84     struct rlimit l;
85     getrlimit(RLIMIT_STACK, &l);
86     cout << "stack_size = " << l.rlim_cur << " byte" << endl;
87 }

```