

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 };
80 void solve(){
81     string s;
82     getline(cin,s);
83     suffix_array sa(s);
84     int n = s.size();
85     int ans = n*(n+1)/2;
86     for(int i = 1;i<n;++i)ans-=sa.LCP[i];
87     cout<<ans<<endl;
88 }
89 signed main(){
90     IOS;
91     solve();
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 #include <bits/stdc++.h>
3
4 #include <bits/stdc++.h>
5 #include <ext/pb_ds/assoc_container.hpp>
6
7 #ifdef _MSC_VER
8 #include <intrin.h>
9 #endif
10
11 namespace felix {
12
13 namespace internal {
14
15     // @param m `1 <= m`
16     // @return x mod m
17     constexpr long long safe_mod(long long x, long long m) {
18         x %= m;
19         if(x < 0) {
20             x += m;
21         }
22         return x;
23     }
24
25     // Fast modular multiplication by barrett reduction
26     // Reference: https://en.wikipedia.org/wiki/Barrett_reduction
27     // NOTE: reconsider after Ice Lake
28     class barrett {
29     public:
30         unsigned int m;
31         unsigned long long im;

```

```

35 // @param m `1 <= m < 2^31`
36 explicit barrett(unsigned int m) : m(m), im((unsigned long
    long)(-1) / m + 1) {}
37
38 // @return m
39 unsigned int umod() const { return m; }
40
41 // @param a `0 <= a < m`
42 // @param b `0 <= b < m`
43 // @return `a * b % m`
44 unsigned int mul(unsigned int a, unsigned int b) const {
45     // [1] m = 1
46     // a = b = im = 0, so okay
47
48     // [2] m >= 2
49     // im = ceil(2^64 / m)
50     // -> im * m = 2^64 + r (0 <= r < m)
51     // Let z = a*b = c*m + d (0 <= c, d < m)
52     // a*b * im = (c*m + d) * im = c*(im*m) + d*im = c*2^64 +
        c*r + d*im
53     // c*r + d*im < m * m + m * im < m * m + 2^64 + m <= 2^64 +
        m * (m + 1) < 2^64 * 2
54     // ((ab * im) >> 64) == c or c + 1
55     unsigned long long z = a;
56     z *= b;
57     #ifdef _MSC_VER
58     unsigned long long x;
59     _umul128(z, im, &x);
60 #else
61     unsigned long long x = (unsigned long long)(((unsigned
        __int128)(z) * im) >> 64);
62 #endif
63     unsigned int v = (unsigned int)(z - x * m);
64     if(m <= v) {
65         v += m;
66     }
67     return v;
68 }
69 };
70
71 // @param n `0 <= n`
72 // @param m `1 <= m`
73 // @return `(x ** n) % m`
74 constexpr long long pow_mod_constexpr(long long x, long long n,
    int m) {
75     if(m == 1) return 0;
76     unsigned int _m = (unsigned int)(m);
77     unsigned long long r = 1;
78     unsigned long long y = safe_mod(x, m);
79     while(n) {
80         if (n & 1) r = (r * y) % _m;
81         y = (y * y) % _m;
82         n >>= 1;
83     }
84     return r;
85 }
86
87 // Reference:
88 // M. Forisek and J. Jancina,
89 // Fast Primality Testing for Integers That Fit into a Machine
90 Word
91 // @param n `0 <= n`
92 constexpr bool is_prime_constexpr(int n) {
93     if(n <= 1) return false;
94     if(n == 2 || n == 7 || n == 61) return true;
95     if(n % 2 == 0) return false;
96     long long d = n - 1;
97     while(d % 2 == 0) d /= 2;
98     constexpr long long bases[3] = {2, 7, 61};
99     for(long long a : bases) {
100         long long t = d;
101         long long y = pow_mod_constexpr(a, t, n);
102         while(t != n - 1 && y != 1 && y != n - 1) {
103             y = y * y % n;
104             t <<= 1;
105         }
106         if(y != n - 1 && t % 2 == 0) {
107             return false;
108         }
109     }
110     return true;
111 }
112 template<int n> constexpr bool is_prime = is_prime_constexpr(n);
113
114 // @param b `1 <= b`
115 // @return pair(g, x) s.t. g = gcd(a, b), xa = g (mod b), 0 <=
    x < b/g
116 constexpr std::pair<long long, long long> inv_gcd(long long a,
    long long b) {
117     a = safe_mod(a, b);
118     if(a == 0) return {b, 0};
119
120     // Contracts:
121     // [1] s - m0 * a = 0 (mod b)
122     // [2] t - m1 * a = 0 (mod b)

```

```

123 // [3] s * |m1| + t * |m0| <= b
124 long long s = b, t = a;
125 long long m0 = 0, m1 = 1;
126
127 while(t) {
128     long long u = s / t;
129     s -= t * u;
130     m0 -= m1 * u; // |m1 * u| <= |m1| * s <= b
131
132     // [3]:
133     // (s - t * u) * |m1| + t * |m0| - m1 * u |
134     // <= s * |m1| - t * u * |m1| + t * (|m0| + |m1| * u)
135     // = s * |m1| + t * |m0| <= b
136
137     auto tmp = s;
138     s = t;
139     t = tmp;
140     tmp = m0;
141     m0 = m1;
142     m1 = tmp;
143 }
144 // by [3]: |m0| <= b/g
145 // by g != b: |m0| < b/g
146 if(m0 < 0) m0 += b / s;
147 return {s, m0};
148 }
149
150 // Compile time primitive root
151 // @param m must be prime
152 // @return primitive root (and minimum in now)
153 constexpr int primitive_root_constexpr(int m) {
154     if(m == 2) return 1;
155     if(m == 167772161) return 3;
156     if(m == 469762049) return 3;
157     if(m == 754974721) return 11;
158     if(m == 998244353) return 3;
159     int divs[20] = {};
160     divs[0] = 2;
161     int cnt = 1;
162     int x = (m - 1) / 2;
163     while(x % 2 == 0) x /= 2;
164     for(int i = 3; (long long)(i)*i <= x; i += 2) {
165         if(x % i == 0) {
166             divs[cnt++] = i;
167             while(x % i == 0) {
168                 x /= i;
169             }
170         }
171     }
172     if(x > 1) {
173         divs[cnt++] = x;
174     }
175     for(int g = 2; g++ ) {
176         bool ok = true;
177         for(int i = 0; i < cnt; i++) {
178             if(pow_mod_constexpr(g, (m - 1) / divs[i], m) == 1)
179                 ok = false;
180             break;
181         }
182         if(ok) return g;
183     }
184 }
185
186 template<int m> constexpr int primitive_root =
187     primitive_root_constexpr(m);
188
189 // @param n `n < 2^32`
190 // @param m `1 <= m < 2^32`
191 // @return sum_{i=0}^{n-1} floor((ai + b) / m) (mod 2^64)
192 unsigned long long floor_sum_unsigned(unsigned long long n,
193     unsigned long long m, unsigned long long a, unsigned long
194     long b) {
195     unsigned long long ans = 0;
196     while(true) {
197         if(a >= m) {
198             ans += n * (n - 1) / 2 * (a / m);
199             a %= m;
200         }
201         if(b >= m) {
202             ans += n * (b / m);
203             b %= m;
204         }
205         unsigned long long y_max = a * n + b;
206         if(y_max < m) {
207             break;
208         }
209         // y_max < m * (n + 1)
210         // floor(y_max / m) <= n
211         n = (unsigned long long)(y_max / m);
212         b = (unsigned long long)(y_max % m);
213         std::swap(m, a);
214     }
215     return ans;
216 }
217 } // namespace internal

```

```

216 } // namespace felix
217
218 namespace felix {
219
220 namespace internal {
221
222 class modint_base {};
223 class static_modint_base : modint_base {};
224
225 template<class T> using is_modint = std::is_base_of<modint_base
226     , T>;
227 template<class T> using is_modint_t = std::enable_if_t<
228     is_modint<T>::value>;
229 } // namespace internal
230
231 template<int m>
232 class static_modint : internal::static_modint_base {
233 public:
234     static constexpr int mod() {
235         return m;
236     }
237
238     static_modint() : value(0) {}
239
240     template<class T>
241     static_modint(T v) {
242         v %= mod();
243         if(v < 0) {
244             v += mod();
245         }
246         value = v;
247     }
248
249     const int& operator()() const {
250         return value;
251     }
252
253     template<class T>
254     explicit operator T() const {
255         return static_cast<T>(value);
256     }
257
258     static_modint& operator+=(const static_modint& rhs) {
259         value += rhs.value;
260         if(value >= mod()) {
261             value -= mod();
262         }
263         return *this;
264     }
265
266     static_modint& operator-=(const static_modint& rhs) {
267         value -= rhs.value;
268         if(value < 0) {
269             value += mod();
270         }
271         return *this;
272     }
273
274     static_modint& operator*=(const static_modint& rhs) {
275         value = (long long) value * rhs.value % mod();
276         return *this;
277     }
278
279     static_modint& operator/=(const static_modint& rhs) {
280         auto eg = internal::inv_gcd(rhs.value, mod());
281         assert(eg.first == 1);
282         return *this *= eg.second;
283     }
284
285     template<class T>
286     static_modint& operator+=(const T& rhs) {
287         return *this += static_modint(rhs);
288     }
289
290     template<class T>
291     static_modint& operator-=(const T& rhs) {
292         return *this -= static_modint(rhs);
293     }
294
295     template<class T>
296     static_modint& operator*=(const T& rhs) {
297         return *this *= static_modint(rhs);
298     }
299
300     template<class T>
301     static_modint& operator/=(const T& rhs) {
302         return *this /= static_modint(rhs);
303     }
304
305     static_modint operator+() const {
306         return *this;
307     }
308
309     static_modint operator-() const {
310

```

```

311     return static_modint() - *this;
312 }
313
314 static_modint& operator++() {
315     return *this += 1;
316 }
317
318 static_modint& operator--() {
319     return *this -= 1;
320 }
321
322 static_modint operator++(int) {
323     static_modint res(*this);
324     *this += 1;
325     return res;
326 }
327
328 static_modint operator--(int) {
329     static_modint res(*this);
330     *this -= 1;
331     return res;
332 }
333
334 static_modint operator+(const static_modint& rhs) {
335     return static_modint(*this) += rhs;
336 }
337
338 static_modint operator-(const static_modint& rhs) {
339     return static_modint(*this) -= rhs;
340 }
341
342 static_modint operator*(const static_modint& rhs) {
343     return static_modint(*this) *= rhs;
344 }
345
346 static_modint operator/(const static_modint& rhs) {
347     return static_modint(*this) /= rhs;
348 }
349
350 inline bool operator==(const static_modint& rhs) const {
351     return value == rhs();
352 }
353
354 inline bool operator!=(const static_modint& rhs) const {
355     return !(*this == rhs);
356 }
357
358 private:
359     int value;
360 };
361
362 template<int m, class T> static_modint<m> operator+(const T&
363     lhs, const static_modint<m>& rhs) {
364     return static_modint<m>(lhs) += rhs;
365 }
366
367 template<int m, class T> static_modint<m> operator-(const T&
368     lhs, const static_modint<m>& rhs) {
369     return static_modint<m>(lhs) -= rhs;
370 }
371
372 template<int m, class T> static_modint<m> operator*(const T&
373     lhs, const static_modint<m>& rhs) {
374     return static_modint<m>(lhs) *= rhs;
375 }
376
377 template<int m, class T> static_modint<m> operator/(const T&
378     lhs, const static_modint<m>& rhs) {
379     return static_modint<m>(lhs) /= rhs;
380 }
381
382 template<int m>
383 std::istream& operator>>(std::istream& in, static_modint<m>&
384     num) {
385     long long x;
386     in >> x;
387     num = static_modint<m>(x);
388     return in;
389 }
390
391 template<int m>
392 std::ostream& operator<<(std::ostream& out, const static_modint
393     <m>& num) {
394     return out << num();
395 }
396
397 template<int id>
398 class dynamic_modint : internal::modint_base {
399 public:
400     static int mod() {
401         return int(bt.umod());
402     }
403
404     static void set_mod(int m) {
405         assert(1 <= m);
406         bt = internal::barrett(m);
407     }
408
409     dynamic_modint() : value(0) {}
410
411     template<class T>
412     dynamic_modint(T v) {
413         v %= mod();
414         if(v < 0) {
415             v += mod();
416         }
417         value = v;
418     }
419
420     const unsigned int& operator()() const {
421         return value;
422     }
423
424     template<class T>
425     explicit operator T() const {
426         return static_cast<T>(value);
427     }
428
429     dynamic_modint& operator+=(const dynamic_modint& rhs) {
430         value += rhs.value;
431         if(value >= umod()) {
432             value -= umod();
433         }
434         return *this;
435     }
436
437     template<class T>
438     dynamic_modint& operator+=(const T& rhs) {
439         return *this += dynamic_modint(rhs);
440     }
441
442     dynamic_modint& operator-=(const dynamic_modint& rhs) {
443         value += mod() - rhs.value;
444         if(value >= umod()) {
445             value -= umod();
446         }
447         return *this;
448     }
449
450     template<class T>
451     dynamic_modint& operator-=(const T& rhs) {
452         return *this -= dynamic_modint(rhs);
453     }
454
455     dynamic_modint& operator*=(const dynamic_modint& rhs) {
456         value = bt.mul(value, rhs.value);
457         return *this;
458     }
459
460     template<class T>
461     dynamic_modint& operator*=(const T& rhs) {
462         return *this *= dynamic_modint(rhs);
463     }
464
465     dynamic_modint& operator/=(const dynamic_modint& rhs) {
466         auto eg = internal::inv_gcd(rhs.value, mod());
467         assert(eg.first == 1);
468         return *this *= eg.second;
469     }
470
471     template<class T>
472     dynamic_modint& operator/=(const T& rhs) {
473         return *this /= dynamic_modint(rhs);
474     }
475
476     dynamic_modint operator+() const {
477         return *this;
478     }
479
480     dynamic_modint operator-() const {
481         return dynamic_modint() - *this;
482     }
483
484     dynamic_modint& operator++() {
485         ++value;
486         if(value == umod()) {
487             value = 0;
488         }
489         return *this;
490     }
491
492     dynamic_modint& operator--() {
493         if(value == 0) {
494             value = umod();
495         }
496         --value;
497         return *this;
498     }
499
500     dynamic_modint operator++(int) {
501         dynamic_modint res(*this);
502         ++*this;
503         return res;
504     }

```

```

499 dynamic_modint operator--(int) {
500     dynamic_modint res(*this);
501     --*this;
502     return res;
503 }
504
505 dynamic_modint operator+(const dynamic_modint& rhs) {
506     return dynamic_modint(*this) += rhs;
507 }
508
509 dynamic_modint operator-(const dynamic_modint& rhs) {
510     return dynamic_modint(*this) -= rhs;
511 }
512
513 dynamic_modint operator*(const dynamic_modint& rhs) {
514     return dynamic_modint(*this) *= rhs;
515 }
516
517 dynamic_modint operator/(const dynamic_modint& rhs) {
518     return dynamic_modint(*this) /= rhs;
519 }
520
521 inline bool operator==(const dynamic_modint& rhs) const {
522     return value == rhs();
523 }
524
525 inline bool operator!=(const dynamic_modint& rhs) const {
526     return !(*this == rhs);
527 }
528
529 private:
530     unsigned int value;
531     static internal::barrett bt;
532     static unsigned int umod() { return bt.umod(); }
533 };
534
535 template<int id, class T> dynamic_modint<id> operator+(const T&
536     lhs, const dynamic_modint<id>& rhs) {
537     return dynamic_modint<id>(lhs) += rhs;
538 }
539
540 template<int id, class T> dynamic_modint<id> operator-(const T&
541     lhs, const dynamic_modint<id>& rhs) {
542     return dynamic_modint<id>(lhs) -= rhs;
543 }
544
545 template<int id, class T> dynamic_modint<id> operator*(const T&
546     lhs, const dynamic_modint<id>& rhs) {
547     return dynamic_modint<id>(lhs) *= rhs;
548 }
549
550 template<int id, class T> dynamic_modint<id> operator/(const T&
551     lhs, const dynamic_modint<id>& rhs) {
552     return dynamic_modint<id>(lhs) /= rhs;
553 }
554
555 template<int id> internal::barrett dynamic_modint<id>::bt
556     (998244353);
557
558 template<int id>
559 std::istream& operator>>(std::istream& in, dynamic_modint<id>&
560     num) {
561     long long x;
562     in >> x;
563     num = dynamic_modint<id>(x);
564     return in;
565 }
566
567 template<int id>
568 std::ostream& operator<<(std::ostream& out, const
569     dynamic_modint<id>& num) {
570     return out << num();
571 }
572
573 using modint998244353 = static_modint<998244353>;
574 using modint1000000007 = static_modint<1000000007>;
575
576 namespace internal {
577
578     template <class T>
579     using is_static_modint = std::is_base_of<static_modint_base, T
580         >;
581
582     template <class T>
583     using is_dynamic_modint_t = std::enable_if_t<is_static_modint<T
584         >::value>;
585
586     template <class> struct is_dynamic_modint : public std::
587         false_type {};
588
589     template <int id>
590     struct is_dynamic_modint<dynamic_modint<id>> : public std::
591         true_type {};
592
593     template <class T>
594     using is_dynamic_modint_t = std::enable_if_t<is_dynamic_modint<T
595         >::value>;
596
597 } // namespace internal
598
599 } // namespace felix
600
601 using namespace std;
602 using namespace felix;
603
604 using mint = modint1000000007;
605
606 mint C(int n, int k) {
607     static vector<mint> fact{1}, inv_fact{1};
608     if(k < 0 || k > n) {
609         return mint(0);
610     }
611     while((int) fact.size() <= n) {
612         fact.push_back(fact.back() * (int) fact.size());
613         inv_fact.push_back(1 / fact.back());
614     }
615     return fact[n] * inv_fact[k] * inv_fact[n - k];
616 }
617
618 int main() {
619     ios::sync_with_stdio(false);
620     cin.tie(0);
621     int n;
622     cin >> n;
623     if(n % 2 == 1) {
624         cout << "0\n";
625         return 0;
626     }
627     string s;
628     cin >> s;
629     int m = (int) s.size();
630     int delta = 0;
631     int left = 0;
632     for(char& c : s) {
633         delta += (c == '(' ? +1 : -1);
634         left += (c == '(');
635         if(delta < 0) {
636             cout << "0\n";
637             return 0;
638         }
639     }
640     left = n / 2 - left;
641     mint ans = C(n - m, left) - C(n - m, left + delta + 1);
642     cout << ans << "\n";
643     return 0;
644 }

```

1.10 CountingNumbers

```

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

```

1.13 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 #include <bits/stdc++.h>
7 using namespace std;
8
9 int main() {
10     ios::sync_with_stdio(false);
11     cin.tie(0);
12     int n;
13     cin >> n;
14     vector<long long> a(n), pref(n + 1);
15     for(int i = 0; i < n; ++i) {
16         cin >> a[i];
17         pref[i + 1] = pref[i] + a[i];
18     }
19     vector<vector<long long>> dp(n, vector<long long>(n));
20     for(int i = 0; i < n; ++i) {
21         dp[i][i] = a[i];
22     }
23     for(int len = 2; len <= n; ++len) {
24         for(int i = 0; i + len - 1 < n; ++i) {
25             int j = i + len - 1;
26             dp[i][j] = pref[j + 1] - pref[i] - min(dp[i + 1][j], dp[i]
27                 ][j - 1]);
28         }
29     }
30     cout << dp[0][n - 1] << "\n";
31     return 0;
32 }

```

1.14 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        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 }

```

1.15 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
24 Matrix FastPower(Matrix a, ull b) {
25     Matrix ans;
26     for(int i = 0; i < 6; ++i)
27         ans.M[i][i] = 1;
28     while(b) {
29         if(b & 1)
30             ans = ans * a;
31         a = a * a;
32         b >>= 1;
33     }
34     return ans;
35 }
36
37 int main() {
38     Matrix A;
39     for(int i = 0; i < 6; ++i)
40         A.M[0][i] = 1;
41     for(int i = 1; i < 6; ++i)
42         A.M[i][i - 1] = 1;
43     ull n;
44     cin >> n;
45     cout << FastPower(A, n).M[0][0] << "\n";
46
47     return 0;
48 }

```

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 }

```



```

43     }
44 };
45 node *root = NULL;
46 int size(node *a){
47     return 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);

```

```

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     }
31     inline void build(const vector<S> &v,const int &l,const int &
32         r,int idx = 1){
33         if(idx==1)arr.resize((r-l+1)<<2);
34         if(l==r){
35             arr[idx].seg = v[l];
36             arr[idx].tag = tag_init();
37             arr[idx].l = arr[idx].r = l;
38             return;
39         }
40         int m = (l+r)>>1;
41         build(v,l,m,idx<<1);
42         build(v,m+1,r,idx<<1|1);
43         arr[idx] = arr[idx<<1]+arr[idx<<1|1];
44     }
45     inline void update(const int &ql,const int &qr,T t,int idx =
46         1){
47         assert(ql<=qr);
48         if(ql<=arr[idx].l and arr[idx].r<=qr){
49             all_apply(idx, t);
50             return;
51         }
52         push(idx);
53         int m = (arr[idx].l+arr[idx].r)>>1;
54         if(ql<=m)update(ql,qr,t,idx<<1);
55         if(qr>m)update(ql,qr,t,idx<<1|1);
56         arr[idx] = arr[idx<<1]+arr[idx<<1|1];
57     }
58     inline S query(const int &ql,const int &qr,int idx = 1){
59         assert(ql<=qr);
60         if(ql<=arr[idx].l and arr[idx].r<=qr){
61             return arr[idx].seg;
62         }
63         push(idx);
64         int m = (arr[idx].l+arr[idx].r)>>1;
65         S ans = node_init(),lhs = node_init(),rhs = node_init();
66         if(ql<=m)lhs = query(ql,qr,idx<<1);
67         if(qr>m)rhs = query(ql,qr,idx<<1|1);
68         ans = node_pull(lhs,rhs);
69         return ans;
70     }
71 };

```

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 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 };

```

2.6 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 };

```

2.7 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 };

```

2.8 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 };

```

2.9 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 ql,
13        int qr,int val,int l,int r,int idx){
14        if(ql<=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    };

```

```

20 int m = (l+r)>>1;
21 if(ql<=m)update(ql,qr,val,l,m,idx<<1);
22 if(qr>m)update(ql,qr,val,m+1,r,idx<<1|1);
23 if(tag[idx])seg[idx] = r-l+1;
24 else seg[idx] = seg[idx<<1]+seg[idx<<1|1];
25 };
26 long long last_pos = 0,ans = 0;
27 for(auto [pos,l,r,val]:tmp){
28     ans+=(pos-last_pos)*seg[l];
29     update(l,r,val,L,R,1);
30     last_pos = pos;
31 }
32 return ans;
33 }

```

2.10 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 ql,int qr){
17        int k = __lg(qr-ql+1);
18        return op(mat[k][ql],mat[k][qr-(1<<k)+1]);
19    }
20 };

```

2.11 VEBTree

```

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(log64(N)) in worst case (sparse array).
4 // Count operation works in O(1) always.
5 template<unsigned int MAXN>
6 class fast_set {
7 private:
8     static const unsigned int 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 unsigned long long lowest_bits1[] = {0ULL,
15        1ULL, 3ULL, 7ULL, 15ULL, 31ULL, 63ULL, 127ULL, 255ULL,
16        511ULL, 1023ULL, 2047ULL, 4095ULL, 8191ULL, 16383ULL,
17        32767ULL, 65535ULL, 131071ULL, 262143ULL, 524287ULL,
18        1048575ULL, 2097151ULL, 4194303ULL, 8388607ULL, 16777215
19        ULL, 33554431ULL, 67108863ULL, 134217727ULL, 268435455ULL,
20        536870911ULL, 1073741823ULL, 2147483647ULL, 4294967295
21        ULL, 8589934591ULL, 17179869183ULL, 34359738367ULL,
22        68719476735ULL, 137438953471ULL, 274877906943ULL,
23        549755813887ULL, 1099511627775ULL, 2199023255551ULL,
24        4398046511103ULL, 8796093022207ULL, 17592186044415ULL,
25        35184372088831ULL, 70368744177663ULL, 140737488355327ULL,
26        28147496710655ULL, 562949953421311ULL, 1125899906842623
27        ULL, 2251799813685247ULL, 4503599627370495ULL,
28        9007199254740991ULL, 18014398509481983ULL,
29        36028797018963967ULL, 72057594037927935ULL,
30        144115188075855871ULL, 288230376151711743ULL,
31        576460752303423487ULL, 1152921504606846975ULL,
32        2305843009213693951ULL, 4611686018427387903ULL,
33        9223372036854775807ULL, 18446744073709551615ULL};
34    static const unsigned int SZ = PREF + (MAXN + 63) / 64 + 1;
35    unsigned long long m[SZ] = {0};
36
37    inline unsigned int left(unsigned int v) const {
38        return (v - 62) * 64;
39    }
40
41    inline unsigned int parent(unsigned int v) const {
42        return v / 64 + 62;
43    }
44
45    inline void setbit(unsigned int v) {
46        m[v >> 6] |= 1ULL << (v & 63);
47    }
48
49    inline void resetbit(unsigned int v) {
50        m[v >> 6] &= ~(1ULL << (v & 63));
51    }
52 };

```

```

31
32 inline unsigned int getbit(unsigned int v) const {
33     return m[v >> 6] >> (v & 63) & 1;
34 }
35
36 inline unsigned long long childs_value(unsigned int v) const
37 {
38     return m[left(v) >> 6];
39 }
40
41 inline int left_go(unsigned int x, const unsigned int c)
42 const {
43     const unsigned long long rem = x & 63;
44     unsigned int bt = PREF * 64 + x;
45     unsigned long long num = m[bt >> 6] & lowest_bitsll[rem + c];
46     if(num) {
47         return (x ^ rem) | __lg(num);
48     }
49     for(bt = parent(bt); bt > 62; bt = parent(bt)) {
50         const unsigned long long rem = bt & 63;
51         num = m[bt >> 6] & lowest_bitsll[rem];
52         if(num) {
53             bt = (bt ^ rem) | __lg(num);
54             break;
55         }
56     }
57     if(bt == 62) {
58         return -1;
59     }
60     while(bt < PREF * 64) {
61         bt = left(bt) | __lg(m[bt - 62]);
62     }
63     return bt - PREF * 64;
64 }
65
66 inline int right_go(unsigned int x, const unsigned int c)
67 const {
68     const unsigned long long rem = x & 63;
69     unsigned int bt = PREF * 64 + x;
70     unsigned long long num = m[bt >> 6] & ~lowest_bitsll[rem + c];
71     if(num) {
72         return (x ^ rem) | __builtin_ctzll(num);
73     }
74     for(bt = parent(bt); bt > 62; bt = parent(bt)) {
75         const unsigned long long rem = bt & 63;
76         num = m[bt >> 6] & ~lowest_bitsll[rem + 1];
77         if(num) {
78             bt = (bt ^ rem) | __builtin_ctzll(num);
79             break;
80         }
81     }
82     if(bt == 62) {
83         return -1;
84     }
85     while(bt < PREF * 64) {
86         bt = left(bt) | __builtin_ctzll(m[bt - 62]);
87     }
88     return bt - PREF * 64;
89 }
90
91 public:
92 fast_set() {
93     assert(PREF != 228);
94     setbit(62);
95 }
96
97 bool empty() const {return getbit(63);}
98
99 void clear() {
100     fill(m, m + SZ, 0);
101     setbit(62);
102 }
103
104 bool count(unsigned int x) const {
105     return m[PREF + (x >> 6)] >> (x & 63) & 1;
106 }
107
108 void insert(unsigned int x) {
109     for(unsigned int v = PREF * 64 + x; !getbit(v); v = parent(v)) {
110         setbit(v);
111     }
112 }
113
114 void erase(unsigned int x) {
115     if(!getbit(PREF * 64 + x)) {
116         return;
117     }
118     resetbit(PREF * 64 + x);
119     for(unsigned int v = parent(PREF * 64 + x); v > 62 && !childs_value(v); v = parent(v)) {
120         resetbit(v);
121     }
122 }

```

```

121 int find_next(unsigned int x) const {
122     return right_go(x, 0);
123 }
124
125 int find_prev(unsigned int x) const {
126     return left_go(x, 1);
127 }
128 };

```

2.12 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 ql, int qr, int l, int r){
32        if(!t)return 0;
33        if(ql<=l and r<=qr)return t->sum;
34        int m = (l+r)>>1;
35        T ans = 0;
36        if(ql<=m)ans+=query(t->l, ql, qr, l, m);
37        if(qr>m)ans+=query(t->r, ql, qr, m+1, r);
38        return ans;
39    }T query(int ql, int qr, int l = -1e9, int r = 1e9){return query(
40        root, ql, qr, l, r);}
41 };

```

2.13 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     }

```

```

34 }
35 S get(int p) {
36     assert(0 <= p && p < n);
37     p += size;
38     for(int i = log; i; i--) {
39         push(p >> i);
40     }
41     return d[p];
42 }
43 S operator[](int p) {
44     return get(p);
45 }
46 S query(int l, int r) {
47     r++;
48     assert(l<=r);
49     l += size;
50     r += size;
51     for(int i = log; i; i--) {
52         if(((l >> i) << i) != 1) {
53             push(l >> i);
54         }
55         if(((r >> i) << i) != r) {
56             push(r >> i);
57         }
58     }
59     S sm1 = node_init(), smr = node_init();
60     while(l < r) {
61         if(l & 1) {
62             sm1 = node_pull(sm1, d[l++]);
63         }
64         if(r & 1) {
65             smr = node_pull(d[--r], smr);
66         }
67         l >>= 1;
68         r >>= 1;
69     }
70     return node_pull(sm1, smr);
71 }
72 void apply(int p, F f) {
73     assert(0 <= p && p < n);
74     p += size;
75     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 };

```

2.14 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 };

```

2.15 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)) * 0x94d049b133111eb;
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>;

```

2.16 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 Theorem

- 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==-1 and r==-1){
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,
        tmp.begin(), [](Point<T> a,Point<T> b){return a.y<b.y;});
16 |     for(int i = 1; i<=r; ++i) p[i] = tmp[i-1];
17 |     tmp.clear();
18 |     for(int i = 1; i<=r; ++i){
19 |         if((p[i].x-mid_pos)*(p[i].x-mid_pos)<ans){
20 |             tmp.push_back(p[i]);
21 |         }
22 |     }
23 |     int n = tmp.size();
24 |     for(int i = 0; i<n; ++i){
25 |         for(int j = i+1; j<n; ++j){
26 |             ans = min(ans,abs2(tmp[i]-tmp[j]));
27 |             if(((tmp[i].y-tmp[j].y)*(tmp[i].y-tmp[j].y))>ans){
28 |                 break;
29 |             }
30 |         }
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 sqrtl(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,
50     const Point<T> &p3){
51     return collinearity(p1,p2,p3) && btw(p1,p2,p3);
52 }
53 template<class T>
54 bool seg_intersect(Point<T> p1, Point<T> p2, Point<T> p3, Point
55     <T> p4) {
56     int a123 = ori(p1, p2, p3);
57     int a124 = ori(p1, p2, p4);
58     int a341 = ori(p3, p4, p1);
59     int a342 = ori(p3, p4, p2);
60     if(a123 == 0 && a124 == 0)
61         return btw(p1, p2, p3) || btw(p1, p2, p4) || btw(p3, p4, p1
62             ) || btw(p3, p4, p2);
63     return a123 * a124 <= 0 && a341 * a342 <= 0;
64 }
65 template<class T>
66 double area(vector<Point<T>> v){
67     if(v.size()<=2)return 0;
68     double ans = 0;
69     for(int i = 1;i<v.size()-1;++i){
70         ans+=((v[i]-v[0])^(v[i+1]-v[0]));
71     }
72     return abs(ans)/2.;
73 }

```

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 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 };

```

4.4 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
20     MCMF() {}
21     MCMF(int _n) : n(_n+1), g(_n+1), d(_n+1), in_queue(_n+1),
22         previous_edge(_n+1) {}
23
24     void add_edge(int u, int v, Cap_t cap, Cost_t cost) {
25         assert(0 <= u && u < n);
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 =
63         numeric_limits<Cap_t>::max()) {
64         assert(0 <= s && s < n);
65         assert(0 <= t && t < n);
66         Cap_t cap = 0;
67         Cost_t cost = 0;
68         while(f > 0 && spfa(s, t)) {
69             Cap_t send = f;
70             int u = t;
71             while(u != s) {
72                 const Edge& e = edges[previous_edge[u]];
73                 send = min(send, e.cap);
74                 u = e.from;
75             }
76             u = t;
77             while(u != s) {
78                 Edge& e = edges[previous_edge[u]];
79                 e.cap -= send;
80                 Edge& b = edges[previous_edge[u] ^ 1];
81                 b.cap += send;
82                 u = e.from;
83             }
84             cap += send;
85             f -= send;
86             cost += send * d[t];
87         }
88         return make_pair(cap, cost);
89     };

```

4.5 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 }

```

4.6 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     //calcuat 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 }

```

4.7 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;
21        }
22        for(auto v: g[u]){
23            if(v == p) continue;
24            if(!dfn[v]){
25                build(g, v, u);
26                child++;
27                if(dfn[u] <= low[v]){
28                    is_ap[u] = 1;
29                    bcc_id[u] = ++bcc_cnt;
30                    bcc[bcc_cnt].push_back(u);
31                    while(vis[v]){
32                        bcc_id[st.top()] = bcc_cnt;
33                        bcc[bcc_cnt].push_back(st.top());
34                        vis[st.top()] = 0;
35                        st.pop();
36                    }
37                    low[u] = min(low[u], low[v]);
38                }
39                low[u] = min(low[u], dfn[v]);
40            }
41            if(p == -1 and child < 2) is_ap[u] = 0;
42            if(is_ap[u]) ap.push_back(u);
43        }
44    }
45 }

```

4.8 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            if(low[u] == dfn[u]){
29                ++cnt;
30                while(vis[u]){
31                    auto v = st.top();
32                    st.pop();
33                    vis[v] = 0;
34                    scc[v] = cnt;
35                    sz[cnt]++;
36                }
37            }
38        }
39    }
40 }

```

```

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 }

```

4.9 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::
24     line_t, std::less<>> {
25     static const long long INF = std::numeric_limits<long long>::
26         max();
27
28     bool isect(iterator x, iterator y) {
29         if(y == end()) {
30             x->p = INF;
31             return 0;
32         }
33         if(x->k == y->k) {
34             x->p = (x->m > y->m ? INF : -INF);
35         }
36         else {
37             x->p = floor_div(y->m - x->m, x->k - y->k);
38         }
39         return x->p >= y->p;
40     }
41
42     void add_line(long long k, long long m) {
43         if(!MAX) {
44             k = -k;
45             m = -m;
46         }
47         auto z = insert({k, m, 0}); y = z++, x = y;
48         while(isect(y, z)) {
49             z = erase(z);
50         }
51         if(x != begin() && isect(--x, y)) {
52             isect(x, y = erase(y));
53         }
54         while((y = x) != begin() && (--x)->p >= y->p) {
55             isect(x, erase(y));
56         }
57     }
58
59     long long get(long long x) {
60         assert(!empty());
61         auto l = *lower_bound(x);
62         return (l.k * x + l.m) * (MAX ? +1 : -1);
63     }
64 };

```

4.10 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 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 s.t. $\pi(j) > \pi(j+1)$, $k+1$ j :s s.t. $\pi(j) \geq j$, k j :s s.t. $\pi(j) > j$.

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

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

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

5.2 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

```

5.3 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 }

```

5.4 Generating Functions

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

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

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

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

- Special Generating Function

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

5.5 Theorem

- Modular Arithmetic

$$(a+b) \bmod m = (a \bmod m + b \bmod m) \bmod m$$

$$(a-b) \bmod m = (a \bmod m - b \bmod m) \bmod m$$

$$(a \cdot b) \bmod m = ((a \bmod m) \cdot (b \bmod m)) \bmod m$$

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

- 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 $d_{ij} = -d_{ji}$. $\frac{\text{rank}(D)}{2}$ 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 $\frac{(n-2)!}{(d_1-1)!(d_2-1)!\dots(d_n-1)!}$ 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

$$\begin{aligned} - f(n) &= \sum_{d|n} g(d) \Leftrightarrow g(n) = \sum_{d|n} \mu(d) f\left(\frac{n}{d}\right) \\ - f(n) &= \sum_{n|d} g(d) \Leftrightarrow g(n) = \sum_{n|d} \mu\left(\frac{d}{n}\right) f(d) \end{aligned}$$

- 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, θ : $\arcsin(a/r)$.
- 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 $= 2\pi r h = \pi(a^2 + h^2) = 2\pi r^2(1 - \cos \theta)$.

5.6 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 += (x_max + a - 1) / a * y_max;
19    return ans + floor_sum(c, (a - x_max % a) % a, a, y_max);
20 }
```

5.7 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)
43 // Complexity: $O(K^2 \log N)$ ($K$: deg. of $f$)
44 // Example: (4, [1, -1, -1]) -> [2, 3]
45 // (x^4 = (x^2 + x + 2)(x^2 - x - 1) + 3x + 2)
46 // Reference: http://misawa.github.io/others/
47 // fast_kitamasa_method.html
48 // http://sugarknri.hatenablog.com/entry/2017/11/18/233936
49
50 template <typename Tfield>
51 std::vector<Tfield> monomial_mod_polynomial(long long N, const
52     std::vector<Tfield> &f_reversed) {
53     assert(!f_reversed.empty() and f_reversed[0] == 1);
54     int K = f_reversed.size() - 1;
55     if (!K) return {};
56 }
```

```

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[
            j] * 2;
58     }
59     return ret;
60 };
61 for (int d = D; d--;) {
62     ret = self_conv(ret);
63     for (int i = 2 * K - 2; i >= K; i--) {
64         for (int j = 1; j <= K; j++) ret[i - j] -= ret[i] *
            f_reversed[j];
65     }
66     ret.resize(K);
67     if ((N >> d) & 1) {
68         std::vector<Tfield> c(K);
69         c[0] = -ret[K - 1] * f_reversed[K];
70         for (int i = 1; i < K; i++) { c[i] = ret[i - 1] -
            ret[K - 1] * f_reversed[K - i]; }
71         ret = c;
72     }
73 }
74 return ret;
75 }
76
77 // Guess k-th element of the sequence, assuming linear
    recurrence
78 // initial_elements: 0-ORIGIN
79 // Verify: abc198f https://atcoder.jp/contests/abc198/
    submissions/21837815
80 template <typename Tfield>
81 Tfield guess_kth_term(const std::vector<Tfield> &
    initial_elements, long long k) {
82     assert(k >= 0);
83     if (k < static_cast<long long>(initial_elements.size()))
84         return initial_elements[k];
85     const auto f = find_linear_recurrence<Tfield>(
        initial_elements).second;
86     const auto g = monomial_mod_polynomial<Tfield>(k, f);
87     Tfield ret = 0;
88     for (unsigned i = 0; i < g.size(); i++) ret += g[i] *
        initial_elements[i];
89     return ret;
90 }

```

5.8 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 }

```

5.9 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>;

```

5.10 CRT

```

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

```

5.11 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/splitm64.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::steady_clock::now().time_since_epoch().count();
27         return splitmix64(x + FIXED_RANDOM);
28     }
29 };
30
31 } // namespace internal
32
33
34 } // namespace felix
35
36 #line 4 "Library/data-structure/pbds.hpp"
37
38 namespace felix {
39
40 template<class T, class U, class H = internal::splitmix64_hash>
41     using hash_map = __gnu_pbds::gp_hash_table<T, U, H>;
42 template<class T, class H = internal::splitmix64_hash> using
43     hash_set = hash_map<T, __gnu_pbds::null_type, H>;
44
45 template<class T, class Comp = std::less<T>> using ordered_set
46     = __gnu_pbds::tree<T, __gnu_pbds::null_type, Comp,
47     __gnu_pbds::rb_tree_tag, __gnu_pbds::tree_order_statistics_node_update>;
48 template<class T> using ordered_multiset = ordered_set<T, std::less_equal<T>>;
49
50 } // namespace felix
51
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 long)(-1) / _m + 1) {}
66
67     unsigned int umod() const { return m; }
68
69     unsigned int mul(unsigned int a, unsigned int b) const {
70         unsigned long long z = a;
71         z *= b;
72 #ifdef _MSC_VER
73         unsigned long long x;
74         _umul128(z, im, &x);
75 #else
76         unsigned long long x = (unsigned long long)(((unsigned __int128)(z) * im) >> 64);
77 #endif
78         unsigned long long y = x * m;
79         return (unsigned int)(z - y + (z < y ? m : 0));
80     }
81 };
82
83 } // namespace internal
84
85 } // namespace felix
86
87 #line 2 "Library/math/binary-gcd.hpp"
88
89 namespace felix {
90
91 template<class T>
92 inline T binary_gcd(T a, T b) {
93     if(a == 0 || b == 0) {
94         return a | b;
95     }
96     int8_t n = __builtin_ctzll(a);
97     int8_t m = __builtin_ctzll(b);
98     a >>= n;
99     b >>= m;
100     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

```

5.12 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 #ifdef 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);
9     return ans;
10 }

```

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 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 =
    thin_heap_tag> using _heap = __gnu_pbds::priority_queue<K,
    cmp, T>;
28 template <typename K, typename M = null_type> using _hash =
    gp_hash_table<K, M>;
29 const int N = 1e6+5,L = 20,mod = 1e9+7;
30 const long long inf = 2e18+5;
31 const double eps = 1e-7,pi = acos(-1);
32 void solve(){
33 }
34 signed main(){
35     IOS;
36     solve();
37 }

```

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,
        9999999937}, {998244353, 9999999929}, {146672737,
        922722049}, {204924373, 952311013}, {585761567,
        955873937}, {484547929, 901981687}, {856009481,
        987877511}, {852853249, 996724213}, {937381759,
        994523539}, {116508269, 993179543}};
8
9     int n;
10    vector<int> POW[MAX_HASH_PAIRS];
11    array<vector<int>, HASH_COUNT> pref;
12
13    int substr(int k, int l, int r) {
14        const auto& p = HASH_PAIRS[k];
15        if(l == r) {
16            return 0;
17        }
18        int res = pref[k][r - 1];
19        if(l > 0) {
20            res -= 1LL * pref[k][l - 1] * get_power(k, r - 1) % p.
                second;
21        }
22        if(res < 0) {
23            res += p.second;
24        }
25        return res;
26    }
27
28    // build powers up to x^k
29    void build_powers(int k) {
30        for(int i = 0; i < HASH_COUNT; ++i) {
31            const auto& p = HASH_PAIRS[i];
32            int sz = (int) POW[i].size();
33            if(sz > k) {
34                continue;
35            }
36            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.
                    second);
42                sz += 1;
43            }
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 <=
            MAX_HASH_PAIRS);
56        for(int i = 0; i < HASH_COUNT; ++i) {
57            const auto& p = HASH_PAIRS[i];
58            pref[i].resize(n);
59            pref[i][0] = s[0];
60            for(int j = 1; j < n; ++j) {

```

```

61                pref[i][j] = (1LL * pref[i][j - 1] * p.first + s[j]) %
                    p.second;
62            }
63        }
64        build_powers(n);
65    }
66
67    void add_char(char c) {
68        for(int i = 0; i < HASH_COUNT; ++i) {
69            const auto& p = HASH_PAIRS[i];
70            pref[i].push_back((1LL * (n == 0 ? 0 : pref[i].back()) *
                p.first + c) % p.second);
71        }
72        n += 1;
73        build_powers(n);
74    }
75
76    // Return hash values for [l, r)
77    array<int, HASH_COUNT> substr(int l, int r) {
78        array<int, HASH_COUNT> res{};
79        for(int i = 0; i < HASH_COUNT; ++i) {
80            res[i] = substr(i, l, r);
81        }
82        return res;
83    }
84
85    array<int, HASH_COUNT> merge(const vector<pair<int, int>>&
        seg) {
86        array<int, HASH_COUNT> res{};
87        for(int i = 0; i < HASH_COUNT; ++i) {
88            const auto& p = HASH_PAIRS[i];
89            for(auto [l, r] : seg) {
90                res[i] = (1LL * res[i] * get_power(i, r - 1) + substr(i
                    , l, r)) % p.second;
91            }
92        }
93        return res;
94    }
95
96    inline int size() const {
97        return n;
98    }
99 };

```

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 #line 2 "Library/string/kmp.hpp"
2 template<class T>
3 vector<int> KMP(const vector<T>& a) {
4     int n = (int) a.size();
5     vector<int> k(n);
6     for(int i = 1; i < n; ++i) {
7         int j = k[i - 1];
8         while(j > 0 && a[i] != a[j]) {
9             j = k[j - 1];
10        }
11        j += (a[i] == a[j]);
12        k[i] = j;
13    }
14    return k;
15 }
16
17 vector<int> KMP(const std::string& s) {
18     return KMP(vector<int>(s.begin(), s.end()));
19 }

```

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

```

```

37         break;
38     }
39 }
40 return ans;
41 }
42
43 private:
44     vector<Node> nodes;
45
46     inline int newNode() {
47         nodes.emplace_back();
48         return (int) nodes.size() - 1;
49     }
50 };

```

8.5 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     }
14
15     inline int next(int p, int v) {
16         return nodes[p].go[v] != -1 ? nodes[p].go[v] : nodes[p].go[
            v] = newNode();
17     }
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
        = 0) const {
31         int ans = 0;
32         for(int v : a) {
33             if(nodes[p].go[v] != -1) {
34                 ans += 1;
35                 p = nodes[p].go[v];
36             } else {

```

ACM ICPC Team

Reference -

LeeJiaHuaPlayMinecraft

Contents

1 CSES	1	3 Geometry	14
1.1 CountingTilings	1	3.1 Theorem	14
1.2 Sequence1	1	3.2 PointInPolygon	14
1.3 JosephusQueries	1	3.3 PointInConvex	14
1.4 AnotherGame	1	3.4 MaximumDistance	14
1.5 CountingCoprimePairs	1	3.5 PolarAngleSort	14
1.6 Sequence2	2	3.6 MinimumDistance	14
1.7 LongestPalindrome	2	3.7 ConvexHull	14
1.8 DistinctSubstrings	2	3.8 Template	14
1.9 BracketSequencesII	3	4 Graph	15
1.10 CountingNumbers	6	4.1 HLD	15
1.11 WordCombinations	7	4.2 Bridges	15
1.12 SumOfDivisors	7	4.3 TwoSat	15
1.13 RemovalGame	7	4.4 MCMF	16
1.14 MinimalRotation	8	4.5 LCA	16
1.15 Dice	8	4.6 CentroidDecomposition	16
2 Data-Structure	8	4.7 BCC AP	17
2.1 Treap	8	4.8 SCC	17
2.2 Segtree	9	4.9 LineContainer	17
2.3 BinaryTrie	10	4.10 Dinic	18
2.4 DsuUndo	10	5 Math	18
2.5 DSU	10	5.1 Numbers	18
2.6 Fenwick	10	5.2 ExtendGCD	18
2.7 Persistent DSU	10	5.3 InvGCD	18
2.8 TimingSegtree	11	5.4 GeneratingFunctions	19
2.9 AreaOfRectangles	11	5.5 Theorem	19
2.10 SparseTable	11	5.6 FloorSum	19
2.11 VEBTree	11	5.7 GuessKth	19
2.12 DynamicSegtree	12	5.8 PowMod	20
2.13 ZkwSegtree	12	5.9 ModInt	20
2.14 MoAlgo	13	5.10 CRT	20
2.15 Hash	13	5.11 DiscreteLog	20
2.16 RedBlackTree	13	5.12 LinearSieve	21
		6 Misc	21
		6.1 FastIO	21
		6.2 Debug	22
		6.3 Discrete	22
		6.4 DuiPai	22
		6.5 Timer	22
		6.6 TenarySearch	22
		7 Setup	22
		7.1 Template	22
		8 String	23
		8.1 RollingHash	23
		8.2 Z	23
		8.3 KMP	23
		8.4 SuffixArray	24
		8.5 Trie	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    }

```