南京大学 ACM-ICPC 集训队代码模版库

Linux-4.15.0-48-generic-x86_64-with-Ubuntu-18.04-bionic
XeTeX 3.14159265-2.6-0.99998 (TeX Live 2017/Debian)
CPython 2.7.15rc1
2019-05-14 00:40:17.558001, build 0043

Contents

1 General

1.1 Code library checksum

```
ab14 #!/usr/bin/python3
c502 import re, sys, hashlib
427e
f7db for line in sys.stdin.read().strip().split("\n") :
ddf5     print(hashlib.md5(re.sub(r'\s|//.*', '', line).encode('utf8')).hexdigest()
           [-4:], line)
```

1.2 Makefile

```
dab2 .PHONY : run
427e
207e $(t) : $(t).cpp
2d16   g++ --std=c++14 -Wall -D__LOCAL_DEBUG__ -fsanitize=undefined -fsanitize=
         address -ggdb -pipe -o $@ $<
427e
5f25 run : $(t)
bf3e   ./$(t) < $(t).in
```

1.3 .vimrc

```
914c set nocompatible
733d syntax on
6bbc colorscheme slate
7db5 set number
b0e3 set cursorline
061b set shiftwidth=2
8011 set softtabstop=2
a66d set tabstop=2
d23a set expandtab
5245 set magic
740c set smartindent
bee8 set backspace=indent,eol,start
815d set cmdheight=1
0a40 set laststatus=2
1c67 set whichwrap=b,s,<,>,[,]
```

1.4 Stack

```
const int STK_SZ = 2000000; bebe
char STK[STK_SZ * sizeof(void*)]; effc
void *STK_BAK; 4e99
 427e
#if defined(__i386__) 7bc9
#define SP "%%esp" 0894
#elif defined(__x86_64__) ac7a
#define SP "%%rsp" a9ea
#endif 1937
 427e
int main() { 3117
  asm volatile("mov␣" SP ",%0;␣mov␣%1," SP: "=g"(STK_BAK):"g"(STK+sizeof(STK)):) 3750
    ;
 427e
  // main program 427e
 427e
  asm volatile("mov␣%0," SP::"g"(STK_BAK)); 6856
  return 0; 7021
} 95cf
```

1.5 Template

```
#include <bits/stdc++.h> 302f
using namespace std; 421c
 427e
#ifdef __LOCAL_DEBUG__ 426f
# define _debug(fmt, ...) fprintf(stderr, "[%s]␣" fmt "\n", \ 3341
    __func__, ##__VA_ARGS__) 611f
#else a8cb
# define _debug(...) ((void) 0) e6b5
#endif 1937
#define rep(i, n) for (int i=0; i<(n); i++) 0d6c
#define Rep(i, n) for (int i=1; i<=(n); i++) cfe3
#define range(x) begin(x), end(x) 3505
typedef long long LL; 5cad
typedef unsigned long long ULL; b773
```

2 Miscellaneous Algorithms

2.1 2-SAT

Usage:

`init(n)`	Initialize the solver with n variables.
`add_clause(x, xval, y, yval)`	Add a clause `(x == xval)-> (y == yval)`.
`solve()`	Solve the problem. Return **`true`** if SAT, or **`false`** if UNSAT.
`operator``[](i)`	Get the value of i-th variable.

```
0f42 const int MAXN = 100005;
03a9 struct twoSAT {
5c83     int n;
8f72     vector<int> G[MAXN*2];
d060     bool mark[MAXN*2];
b42d     int S[MAXN*2], c;
427e
d34f     void init(int n) {
b985         this->n = n;
f9ec         for (int i=0; i < n*2; i++) G[i].clear();
0609         memset(mark, 0, sizeof(mark));
95cf     }
427e
3bd5     bool dfs(int x) {
bd70         if (mark[x^1]) return false;
c96a         if (mark[x]) return true;
fd23         mark[x] = true;
4bea         S[c++] = x;
bd55         for (int u : G[x]) if (!dfs(u)) return false;
3361         return true;
95cf     }
427e
5894     void add_clause(int x, bool xval, int y, bool yval) {
6afe         x = x * 2 + xval;
e680         y = y * 2 + yval;
81cc         G[x^1].push_back(y);
95cf     }
427e
d0cb     bool solve() {
7c39         for (int i=0; i<n*2; i+=2) {
e63f             if (!mark[i] && !mark[i+1]) {
88fb                 c = 0;
                     if (!dfs(i)) {                              f4b9
                         while (c > 0) mark[S[--c]] = false;     3f03
                         if (!dfs(i+1)) return false;            86c5
                     }                                           95cf
                 }                                               95cf
             }                                                   95cf
             return true;                                        3361
         }                                                       95cf
                                                                 427e
         bool operator[] (int x) { return mark[2*x+1]; }         fb3b
     };                                                          329b
```

2.2 Mo's algorithm

All intervals are closed on both sides. When running functions `enter()` and `leave()`, the global l and r has not changed yet. **Assume the data structure is initialized for empty interval**.

Usage:

`add_query(id, l, r)`	Add id-th query $[l, r]$.
`run()`	Run Mo's algorithm.
`yield(id)`	**TODO**. Yield answer for id-th query.
`enter(o)`	**TODO**. Add o-th element.
`leave(o)`	**TODO**. Remove o-th element.

```
constexpr int BLOCK_SZ = 300;                          5194
                                                       427e
struct query { int l, r, id; };                        3ec4
vector<query> queries;                                 d26a
                                                       427e
void add_query(int id, int l, int r) {                 1e30
  queries.push_back(query{l, r, id});                  54c9
}                                                      95cf
                                                       427e
int l, r;                                              9f6b
                                                       427e
// ----- functions to implement -----                  427e
inline void yield(int id);                             50e1
inline void enter(int o);                              b20d
inline void leave(int o);                              13af
                                                       427e
void run() {                                           37f0
  if (queries.empty()) return;                         ab0b
  sort(range(queries), [](query lhs, query rhs) {      8508
```

```
    int lb = lhs.l / BLOCK_SZ, rb = rhs.l / BLOCK_SZ;
    if (lb != rb) return lb < rb;
    return lhs.r < rhs.r;
  });
  l = queries[0].l;
  r = queries[0].r;
  for (int i = l; i <= r; i++) enter(i);
  for (query q : queries) {
    while (l > q.l) enter(l - 1), l--;
    while (r < q.r) enter(r + 1), r++;
    while (l < q.l) leave(l), l++;
    while (r > q.r) leave(r), r--;
    yield(q.id);
  }
}
```

3 String

3.1 Knuth-Morris-Pratt algorithm

```
const int SIZE = 10005;

struct kmp_matcher {
  char p[SIZE];
  int fail[SIZE];
  int len;

  void construct(const char* needle) {
    len = strlen(p);
    strcpy(p, needle);
    fail[0] = fail[1] = 0;
    for (int i = 1; i < len; i++) {
      int j = fail[i];
      while (j && p[i] != p[j]) j = fail[j];
      fail[i + 1] = p[i] == p[j] ? j + 1 : 0;
    }
  }

  inline void found(int pos) {
    // ! add codes for having found at pos
  }

  void match(const char* haystack) {  // must be called after construct
    const char* t = haystack;
    int n = strlen(t);
    int j = 0;
    rep(i, n) {
      while (j && p[j] != t[i]) j = fail[j];
      if (p[j] == t[i]) j++;
      if (j == len) found(i - len + 1);
    }
  }
};
```

3.2 Manacher algorithm

```
struct Manacher {
  int Len;
  vector<int> lc;
  string s;

  void work() {
    lc[1] = 1;
    int k = 1;

    for (int i = 2; i <= Len; i++) {
      int p = k + lc[k] - 1;
      if (i <= p) {
        lc[i] = min(lc[2 * k - i], p - i + 1);
      } else {
        lc[i] = 1;
      }
      while (s[i + lc[i]] == s[i - lc[i]]) lc[i]++;
      if (i + lc[i] > k + lc[k]) k = i;
    }
  }

  void init(const char *tt) {
    int len = strlen(tt);
    s.resize(len * 2 + 10);
    lc.resize(len * 2 + 10);
    s[0] = '*';
    s[1] = '#';
```

```
1321     for (int i = 0; i < len; i++) {
e995       s[i * 2 + 2] = tt[i];
69fd       s[i * 2 + 1] = '#';
95cf     }
43fd     s[len * 2 + 1] = '#';
75d1     s[len * 2 + 2] = '\0';
61f7     Len = len * 2 + 2;
3e7a     work();
95cf   }
427e
b194   pair<int, int> maxpal(int l, int r) {
901a     int center = l + r + 1;
ffb2     int rad = lc[center] / 2;
ab54     int rmid = (l + r + 1) / 2;
17e4     int rl = rmid - rad, rr = rmid + rad - 1;
3908     if ((r ^ l) & 1) {
69f3     } else rr++;
69dc     return {max(l, rl), min(r, rr)};
95cf   }
329b };
```

3.3 Aho-corasick automaton

```
a1ad struct AC : Trie {
9143   int fail[MAXN];
daca   int last[MAXN];
427e
8690   void construct() {
93d2     queue<int> q;
a7a6     fail[0] = 0;
ce3c     rep(c, CHARN) {
b1c6       if (int u = tr[0][c]) {
a506         fail[u] = 0;
3e14         q.push(u);
f689         last[u] = 0;
95cf       }
95cf     }
cc78     while (!q.empty()) {
31f0       int r = q.front();
15dd       q.pop();
ce3c       rep(c, CHARN) {
ab59         int u = tr[r][c];
```

```
        if (!u) {                                          0ef5
          tr[r][c] = tr[fail[r]][c];                       9d58
          continue;                                        b333
        }                                                  95cf
        q.push(u);                                         3e14
        int v = fail[r];                                   b3ff
        while (v && !tr[v][c]) v = fail[v];                d2ea
        fail[u] = tr[v][c];                                c275
        last[u] = tag[fail[u]] ? fail[u] : last[fail[u]];  654c
      }                                                    95cf
    }                                                      95cf
  }                                                        95cf
                                                           427e
  void found(int pos, int j) {                             7752
    if (j) {                                               043e
      // ! add codes for having found word with tag[j]     427e
      found(pos, last[j]);                                 4a96
    }                                                      95cf
  }                                                        95cf
                                                           427e
  void find(const char* text) {  // must be called after construct()  9785
    int p = 0, c, len = strlen(text);                      80a4
    rep(i, len) {                                          9c94
      c = id(text[i]);                                     b3db
      p = tr[p][c];                                        f119
      if (tag[p])                                          f08e
        found(i, p);                                       389b
      else if (last[p])                                    1e67
        found(i, last[p]);                                 299e
    }                                                      95cf
  }                                                        95cf
};                                                         329b
```

3.4 Trie

```
const int MAXN = 12000;                                    e6f1
const int CHARN = 26;                                      dd87
                                                           427e
inline int id(char c) { return c - 'a'; }                  8ff5
                                                           427e
struct Trie {                                              a281
  int n;                                                   5c83
```

```
f4f5   int tr[MAXN][CHARN];  // Trie tree, 0 denotes fail
35a5   int tag[MAXN];
427e
4fee   Trie() {
3ccc     memset(tr[0], 0, sizeof(tr[0]));
4d52     tag[0] = 0;
46bf     n = 1;
95cf   }
427e
427e   // tag should not be 0
30b0   void add(const char* s, int t) {
d50a     int p = 0, c, len = strlen(s);
9c94     rep(i, len) {
3140       c = id(s[i]);
d6c8       if (!tr[p][c]) {
26dd         memset(tr[n], 0, sizeof(tr[n]));
2e5c         tag[n] = 0;
73bb         tr[p][c] = n++;
95cf       }
f119       p = tr[p][c];
95cf     }
35ef     tag[p] = t;
95cf   }
427e
427e   // returns 0 if not found
427e   // AC automaton does not need this function
216c   int search(const char* s) {
d50a     int p = 0, c, len = strlen(s);
9c94     rep(i, len) {
3140       c = id(s[i]);
f339       if (!tr[p][c]) return 0;
f119       p = tr[p][c];
95cf     }
840e     return tag[p];
95cf   }
329b };
```

3.5 Suffix array

The character immediately after the end of the string **MUST** be set to the **UNIQUE SMALLEST** element.
Usage:

`s[]`	the source string
`sa[i]`	the index of starting position of i-th suffix
`rk[i]`	the number of suffixes less than the suffix starting from i
`h[i]`	the longest common prefix between the i-th and $(i-1)$-th lexicographically smallest suffixes
`n`	size of source string
`m`	size of character set

```
void radix_sort(int x[], int y[], int sa[], int n, int m) {    de09
    static int cnt[1000005];    // size > max(n, m)    ec00
    fill(cnt, cnt + m, 0);    6066
    rep (i, n) cnt[x[y[i]]]++;    93b7
    partial_sum(cnt, cnt + m, cnt);    9154
    for (int i = n - 1; i >= 0; i--) sa[--cnt[x[y[i]]]] = y[i];    acac
}    95cf
    427e
void suffix_array(int s[], int sa[], int rk[], int n, int m) {    c939
    static int y[1000005];  // size > n    a69a
    copy(s, s + n, rk);    7306
    iota(y, y + n, 0);    afbb
    radix_sort(rk, y, sa, n, m);    7b42
    for (int j = 1, p = 0; j <= n; j <<= 1, m = p, p = 0) {    c8c2
        for (int i = n - j; i < n; i++) y[p++] = i;    8c3a
        rep (i, n) if (sa[i] >= j) y[p++] = sa[i] - j;    9323
        radix_sort(rk, y, sa, n, m + 1);    9e9d
        swap_ranges(rk, rk + n, y);    ae41
        rk[sa[0]] = p = 1;    ffd2
        for (int i = 1; i < n; i++)    445e
            rk[sa[i]] = ((y[sa[i]] == y[sa[i-1]] and y[sa[i]+j] == y[sa[i-1]+j])    f8dc
                ? p : ++p);
        if (p == n) break;    02f0
    }    95cf
    rep (i, n) rk[sa[i]] = i;    97d9
}    95cf
    427e
void calc_height(int s[], int sa[], int rk[], int h[], int n) {    1715
    int k = 0;    c41f
    h[0] = 0;    f313
    rep (i, n) {    be8e
        k = max(k - 1, 0);    0883
        if (rk[i]) while (s[i+k] == s[sa[rk[i]-1]+k]) ++k;    527d
        h[rk[i]] = k;    56b7
    }    95cf
}    95cf
```

3.6 Rolling hash

PLEASE call `init_hash()` in `int main()`!

Usage:

`build(str)`	Construct the hasher with given string.
`operator()(l, r)`	Get hash value of substring $[l, r)$.

```
1e42 const LL mod = 1006658951440146419, g = 967;
9f60 const int MAXN = 200005;
0291 LL pg[MAXN];
427e 
dfe7 inline LL mul(LL x, LL y) { return __int128_t(x) * y % mod; }
427e 
599a void init_hash() {   // must be called in `int main()`
286f     pg[0] = 1;
4af8     for (int i = 1; i < MAXN; i++) pg[i] = mul(pg[i-1], g);
95cf }
427e 
7e62 struct hasher {
534a     LL val[MAXN];
427e 
4554     void build(const char *str) {   // assume lower-case letter only
f937         for (int i = 0; str[i]; i++)
9645             val[i+1] = (mul(val[i], g) + str[i]) % mod;
95cf     }
427e 
19f8     LL operator() (int l, int r) { // [l, r)
9986         return (val[r] - mul(val[l], pg[r-l]) + mod) % mod;
95cf     }
329b };
```

4 Math

4.1 Extended Euclidean algorithm and Chinese remainder theorem

```
4fba void exgcd(LL a, LL b, LL &g, LL &x, LL &y) {
7db6     if (!b) g = a, x = 1, y = 0;
037f     else {
ffca         exgcd(b, a % b, g, y, x);
d798         y -= x * (a / b);
95cf     }
95cf }
```

```
 427e
LL crt(LL r[], LL p[], int n) { e491
    LL q = 1, ret = 0; 84e6
    rep (i, n) q *= p[i]; 00d9
    rep (i, n) { be8e
        LL m = q / p[i]; 98b4
        LL d, x, y; 9f4f
        exgcd(p[i], m, d, x, y); b082
        ret = (ret + y * m * r[i]) % q; 3cd3
    } 95cf
    return (q + ret) % q; 2e47
} 95cf
```

4.2 Linear basis

```
const int MAXD = 30; 8b44
struct linearbasis { 03a6
    ULL b[MAXD] = {}; 3558
 427e
    bool insert(LL v) { 1566
        for (int j = MAXD - 1; j >= 0; j--) { 9b2b
            if (!(v & (1ll << j))) continue; de36
            if (b[j]) v ^= b[j] ee78
            else { 037f
                for (int k = 0; k < j; k++) 7836
                    if (v & (1ll << k)) v ^= b[k]; f0b4
                for (int k = j + 1; k < MAXD; k++) b0aa
                    if (b[k] & (1ll << j)) b[k] ^= v; 46c9
                b[j] = v; 8295
                return true; 3361
            } 95cf
        } 95cf
        return false; 438e
    } 95cf
}; 329b
```

4.3 Gauss elimination over finite field

```
const LL p = 1000000007; b784
 427e
LL powmod(LL b, LL e) { 2a2c
```

```
95a2   LL r = 1;
3e90   while (e) {
1783     if (e & 1) r = r * b % p;
5549     b = b * b % p;
16fc     e >>= 1;
95cf   }
547e   return r;
95cf }
427e
c130 typedef vector<LL> VLL;
42ac typedef vector<VLL> VVLL;
427e
2c62 LL gauss(VVLL &a, VVLL &b) {
561b   const int n = a.size(), m = b[0].size();
a25e   vector<int> irow(n), icol(n), ipiv(n);
2976   LL det = 1;
427e
be8e   rep (i, n) {
d2b5     int pj = -1, pk = -1;
6b4a     rep (j, n) if (!ipiv[j])
e582       rep (k, n) if (!ipiv[k])
6112         if (pj == -1 || a[j][k] > a[pj][pk]) {
a905           pj = j;
657b           pk = k;
95cf         }
d480     if (a[pj][pk] == 0) return 0;
0305     ipiv[pk]++;
8dad     swap(a[pj], a[pk]);
aad8     swap(b[pj], b[pk]);
be4d     if (pj != pk) det = (p - det) % p;
d080     irow[i] = pj;
f156     icol[i] = pk;
427e
4ecd     LL c = powmod(a[pk][pk], p - 2);
865b     det = det * a[pk][pk] % p;
c36a     a[pk][pk] = 1;
dd36     rep (j, n) a[pk][j] = a[pk][j] * c % p;
1b23     rep (j, m) b[pk][j] = b[pk][j] * c % p;
f8f3     rep (j, n) if (j != pk) {
e97f       c = a[j][pk];
c449       a[j][pk] = 0;
820b       rep (k, n) a[j][k] = (a[j][k] + p - a[pk][k] * c % p) % p;
f039       rep (k, m) b[j][k] = (b[j][k] + p - b[pk][k] * c % p) % p;
95cf     }
```

```
  }                                                                          95cf
                                                                             427e
  for (int j = n - 1; j >= 0; j--) if (irow[j] != icol[j]) {                 37e1
    for (int k = 0; k < n; k++) swap(a[k][irow[j]], a[k][icol[j]]);          50dc
  }                                                                          95cf
  return det;                                                                f27f
}                                                                            95cf
```

4.4 Berlekamp-Massey algorithm

Call `berlekamp()` with input sequence $(x_0, x_1, \cdots, x_{n-1})$. Return a vector of coefficients $(c_0 = 1, c_1, \cdots, c_{m-1})$ with minimum m, such that $\sum_{i=0}^{m} c_i x_{j-i} = 0$ for all possible j.

```
LL mod = 1000000007;                                                         6e50
vector<LL> berlekamp(const vector<LL>& a) {                                  97db
    vector<LL> p = {1}, r = {1};                                             8904
    LL dif = 1;                                                              075b
    rep (i, a.size()) {                                                      8bc9
        LL u = 0;                                                            1b35
        rep (j, p.size()) u = (u + p[j] * a[i-j]) % mod;                     bd0b
        if (u == 0) {                                                        eae9
            r.insert(r.begin(), 0);                                          b14c
        } else {                                                             8e2e
            auto op = p;                                                     0c78
            p.resize(max(p.size(), r.size() + 1));                           02f6
            LL idif = powmod(dif, mod - 2);                                  0a2e
            rep (j, r.size())                                                9b57
                p[j+1] = (p[j+1] - r[j] * idif % mod * u % mod + mod) % mod; dacc
            dif = u; r = op;                                                 bcd1
        }                                                                    95cf
    }                                                                        95cf
    return p;                                                                e149
}                                                                            95cf
```

4.5 Fast Walsh-Hadamard transform

```
void fwt(int* a, int n){                                                     061e
    for (int d = 1; d < n; d <<= 1)                                          5595
        for (int i = 0; i < n; i += d << 1)                                  05f2
            rep (j, d){                                                      b833
                int x = a[i+j], y = a[i+j+d];                                7796
```

```
427e                 // a[i+j] = x+y, a[i+j+d] = x-y;    // xor
427e                 // a[i+j] = x+y;                    // and
427e                 // a[i+j+d] = x+y;                  // or
95cf             }
95cf }
427e
4db1 void ifwt(int* a, int n){
5595     for (int d = 1; d < n; d <<= 1)
05f2         for (int i = 0; i < n; i += d << 1)
b833             rep (j, d){
7796                 int x = a[i+j], y = a[i+j+d];
427e                 // a[i+j] = (x+y)/2, a[i+j+d] = (x-y)/2;    // xor
427e                 // a[i+j] = x-y;                            // and
427e                 // a[i+j+d] = y-x;                          // or
95cf             }
95cf }
427e
2ab6 void conv(int* a, int* b, int n){
950a     fwt(a, n);
e427     fwt(b, n);
8a42     rep(i, n) a[i] *= b[i];
430f     ifwt(a, n);
95cf }
```

4.6 Fast fourier transform

```
4e09 const int NMAX = 1<<20;
427e
3fbf typedef complex<double> cplx;
427e
abd1 const double PI = 2*acos(0.0);
12af struct FFT{
c47c     int rev[NMAX];
27d7     cplx omega[NMAX], oinv[NMAX];
9827     int K, N;
427e
1442     FFT(int k){
e209         K = k; N = 1 << k;
b393         rep (i, N){
7ba3             rev[i] = (rev[i>>1]>>1) | ((i&1)<<(K-1));
1908             omega[i] = polar(1.0, 2.0 * PI / N * i);
a166             oinv[i] = conj(omega[i]);
            }                                                   95cf
        }                                                       95cf
                                                                427e
        void dft(cplx* a, cplx* w){                             b941
            rep (i, N) if (i < rev[i]) swap(a[i], a[rev[i]]);   a215
            for (int l = 2; l <= N; l *= 2){                    ac6e
                int m = l/2;                                    2969
                for (cplx* p = a; p != a + N; p += l)           b3cf
                    rep (k, m){                                 c24f
                        cplx t = w[N/l*k] * p[k+m];             fe06
                        p[k+m] = p[k] - t; p[k] += t;           ecbf
                    }                                           95cf
            }                                                   95cf
        }                                                       95cf
                                                                427e
        void fft(cplx* a){dft(a, omega);}                       617b
        void ifft(cplx* a){                                     a123
            dft(a, oinv);                                       3b2f
            rep (i, N) a[i] /= N;                               57fc
        }                                                       95cf
                                                                427e
        void conv(cplx* a, cplx* b){                            bdc0
            fft(a); fft(b);                                     6497
            rep (i, N) a[i] *= b[i];                            12a5
            ifft(a);                                            f84e
        }                                                       95cf
};                                                              329b
```

4.7 Number theoretic transform

```
const int NMAX = 1<<21;                                         4ab9
                                                                427e
// 998244353 = 7*17*2^23+1, G = 3                               427e
const int P = 1004535809, G = 3; // = 479*2^21+1                fb9a
                                                                427e
struct NTT{                                                     87ab
    int rev[NMAX];                                              c47c
    LL omega[NMAX], oinv[NMAX];                                 0eda
    int g, g_inv; // g: g_n = G^((P-1)/n)                       81af
    int K, N;                                                   9827
                                                                427e
    LL powmod(LL b, LL e){                                      2a2c
```

```
        LL r = 1;
        while (e){
            if (e&1) r = r * b % P;
            b = b * b % P;
            e >>= 1;
        }
        return r;
    }

    NTT(int k){
        K = k; N = 1 << k;
        g = powmod(G, (P-1)/N);
        g_inv = powmod(g, N-1);
        omega[0] = oinv[0] = 1;
        rep (i, N){
            rev[i] = (rev[i>>1]>>1) | ((i&1)<<(K-1));
            if (i){
                omega[i] = omega[i-1] * g % P;
                oinv[i] = oinv[i-1] * g_inv % P;
            }
        }
    }

    void _ntt(LL* a, LL* w){
        rep (i, N) if (i < rev[i]) swap(a[i], a[rev[i]]);
        for (int l = 2; l <= N; l *= 2){
            int m = l/2;
            for (LL* p = a; p != a + N; p += l)
                rep (k, m){
                    LL t = w[N/l*k] * p[k+m] % P;
                    p[k+m] = (p[k] - t + P) % P;
                    p[k] = (p[k] + t) % P;
                }
        }
    }

    void ntt(LL* a){_ntt(a, omega);}
    void intt(LL* a){
        LL inv = powmod(N, P-2);
        _ntt(a, oinv);
        rep (i, N) a[i] = a[i] * inv % P;
    }

    void conv(LL* a, LL* b){
        ntt(a); ntt(b);
        rep (i, N) a[i] = a[i] * b[i] % P;
        intt(a);
    }
};
```

4.8 Sieve of Euler

```
const int MAXX = 1e7+5;
bool p[MAXX];
int prime[MAXX], sz;

void sieve(){
    p[0] = p[1] = 1;
    for (int i = 2; i < MAXX; i++){
        if (!p[i]) prime[sz++] = i;
        for (int j = 0; j < sz && i*prime[j] < MAXX; j++){
            p[i*prime[j]] = 1;
            if (i % prime[j] == 0) break;
        }
    }
}
```

4.9 Sieve of Euler (General)

```
namespace sieve {
  constexpr int MAXN = 10000007;
  bool p[MAXN]; // true if not prime
  int prime[MAXN], sz;
  int pval[MAXN], pcnt[MAXN];
  int f[MAXN];

  void exec(int N = MAXN) {
    p[0] = p[1] = 1;

    pval[1] = 1;
    pcnt[1] = 0;
    f[1] = 1;

    for (int i = 2; i < N; i++) {
      if (!p[i]) {
```

```
b2b2        prime[sz++] = i;
37d9        for (LL j = i; j < N; j *= i) {
758c          int b = j / i;
81fd          pval[j] = i * pval[b];
e0f3          pcnt[j] = pcnt[b] + 1;
a96c          f[j] = ______; // f[j] = f(i^pcnt[j])
95cf        }
95cf      }
34c0      for (int j = 0; i * prime[j] < N; j++) {
f87a        int x = i * prime[j]; p[x] = 1;
20cc        if (i % prime[j] == 0) {
9985          pval[x] = pval[i] * prime[j];
3f93          pcnt[x] = pcnt[i] + 1;
8e2e        } else {
cc91          pval[x] = prime[j];
6322          pcnt[x] = 1;
95cf        }
6191        if (x != pval[x]) {
d614          f[x] = f[x / pval[x]] * f[pval[x]]
95cf        }
5f51        if (i % prime[j] == 0) break;
95cf      }
95cf    }
95cf  }
95cf }
```

4.10 Miller-Rabin primality test

The array a[] (excluding senitel, i.e. LLONG_MAX) should be

{2}	when $n < 2,047$.
{2, 7, 61}	when $n < 4,759,123,141\ (2^{32})$.
{2, 3, 5, 7, 11}	when $n < 2.1 \times 10^{12}$.
{2, 325, 9375, 28178, 450775, 9780504, 1795265022}	when $n < 2^{64}$.

```
f16f bool test(LL n){
59f2     if (n < 3) return n==2;
427e     // ! The array a[] should be modified if the range of x changes.
3f11     const LL a[] = {2LL, 7LL, 61LL, LLONG_MAX};
c320     LL r = 0, d = n-1, x;
f410     while (~d & 1) d >>= 1, r++;
2975     for (int i=0; a[i] < n; i++){
ece1         x = powmod(a[i], d, n); // ! powmod must use for 64bit mulmod
        if (x == 1 || x == n-1) goto next;       7f99
        rep (i, r) {                             e257
            x = mulmod(x, x, n);                 d7ff
            if (x == n-1) goto next;             8d2e
        }                                        95cf
        return false;                            438e
next:;                                           d490
    }                                            95cf
    return true;                                 3361
}                                                95cf
```

4.11 Integer factorization (Pollard's rho)

```
ULL gcd(ULL a, ULL b) {return b ? gcd(b, a % b) : a;}   2e6b
                                                        427e
ULL PollardRho(ULL n){                                  54a5
    ULL c, x, y, d = n;                                 45eb
    if (~n&1) return 2;                                 d3e5
    while (d == n){                                     3c69
        x = y = 2;                                      0964
        d = 1;                                          4753
        c = rand() % (n - 1) + 1;                       5952
        while (d == 1){                                 9e5b
            x = (mulmod(x, x, n) + c) % n;              33d5
            y = (mulmod(y, y, n) + c) % n;              e1bf
            y = (mulmod(y, y, n) + c) % n;              e1bf
            d = gcd(x>y ? x-y : y-x, n);                a313
        }                                               95cf
    }                                                   95cf
    return d;                                           5d89
}                                                       95cf
```

5 Graph Theory

5.1 Strongly connected component

```
const int MAXV = 100005;                                837c
                                                        427e
struct graph{                                           2ea0
    vector<int> adj[MAXV];                              88e3
```

```
9cad     stack<int> s;
3d02     int V; // number of vertices
8b6c     int pre[MAXV], lnk[MAXV], scc[MAXV];
27ee     int time, sccn;
427e
bfab     void add_edge(int u, int v){
c71a         adj[u].push_back(v);
95cf     }
427e
d714     void dfs(int u){
7e41         pre[u] = lnk[u] = ++time;
80f6         s.push(u);
18f6         for (int v : adj[u]){
173e             if (!pre[v]){
5f3c                 dfs(v);
002c                 lnk[u] = min(lnk[u], lnk[v]);
6068             } else if (!scc[v]){
d5df                 lnk[u] = min(lnk[u], pre[v]);
95cf             }
95cf         }
8de2         if (lnk[u] == pre[u]){
660f             sccn++;
3c9e             int x;
a69f             do {
3834                 x = s.top(); s.pop();
b0e9                 scc[x] = sccn;
6757             } while (x != u);
95cf         }
95cf     }
427e
4c88     void find_scc(){
f4a2         time = sccn = 0;
8de7         memset(scc, 0, sizeof scc);
8c2f         memset(pre, 0, sizeof pre);
6901         Rep (i, V){
56d1             if (!pre[i]) dfs(i);
95cf         }
95cf     }
427e
27ce     vector<int> adjc[MAXV];
364d     void contract(){
1a1e         Rep (i, V)
21a2             rep (j, adj[i].size()){
b730                 if (scc[i] != scc[adj[i][j]])
                    adjc[scc[i]].push_back(scc[adj[i][j]]);   b46e
            }   95cf
    }   95cf
};   329b
```

5.2 Vertex biconnected component

```
const int MAXN = 100005;   0f42
struct graph {   2ea0
    int pre[MAXN], iscut[MAXN], bccno[MAXN], dfs_clock, bcc_cnt;   33ae
    vector<int> adj[MAXN], bcc[MAXN];   848f
    set<pair<int, int>> bcce[MAXN];   6b06
   427e
    stack<pair<int, int>> s;   76f7
   427e
    void add_edge(int u, int v) {   bfab
        adj[u].push_back(v);   c71a
        adj[v].push_back(u);   a717
    }   95cf
   427e
    int dfs(int u, int fa) {   7d3c
        int lowu = pre[u] = ++dfs_clock;   9fe6
        int child = 0;   ec14
        for (int v : adj[u]) {   18f6
            if (!pre[v]) {   173e
                s.push({u, v});   e7f8
                child++;   fdcf
                int lowv = dfs(v, u);   f851
                lowu = min(lowu, lowv);   189c
                if (lowv >= pre[u]) {   b687
                    iscut[u] = 1;   6323
                    bcc[bcc_cnt].clear();   57eb
                    bcce[bcc_cnt].clear();   90b8
                    while (1) {   a147
                        int xu, xv;   a6a3
                        tie(xu, xv) = s.top(); s.pop();   a0c3
                        bcce[bcc_cnt].insert({min(xu, xv), max(xu, xv)});   0ef5
                        if (bccno[xu] != bcc_cnt) {   3db2
                            bcc[bcc_cnt].push_back(xu);   e0db
                            bccno[xu] = bcc_cnt;   d27f
                        }   95cf
                        if (bccno[xv] != bcc_cnt) {   f357
```

```
752b                            bcc[bcc_cnt].push_back(xv);
57c9                            bccno[xv] = bcc_cnt;
95cf                        }
7096                        if (xu == u && xv == v) break;
95cf                    }
03f5                    bcc_cnt++;
95cf                }
7470            } else if (pre[v] < pre[u] && v != fa) {
e7f8                s.push({u, v});
f115                lowu = min(lowu, pre[v]);
95cf            }
95cf        }
e104        if (fa < 0 && child == 1) iscut[u] = 0;
1160        return lowu;
95cf    }
427e
17be    void find_bcc(int n) {
8c2f        memset(pre, 0, sizeof pre);
e2d2        memset(iscut, 0, sizeof iscut);
40d3        memset(bccno, -1, sizeof bccno);
fae2        dfs_clock = bcc_cnt = 0;
5c63        rep (i, n) if (!pre[i]) dfs(i, -1);
95cf    }
329b };
```

5.3 Cut vertices

If the graph is unconnected, the algorithm should be run on each component. One may run `Rep (i, n)if (!dfn[i])tarjan(i, i)` for unconnected graph.

Usage:

`add_edge(u, v)`	Add an undirected edge (u, v).
`tarjan(u, fa)`	Run Tarjan's algorithm on tree rooted at `fa`. Please call with identical `u` and `fa`.
`cut[v]`	Whether v is a cut vertex.

```
9f60 const int MAXN = 200005;
0b32 vector<int> adj[MAXN];
18e4 int dfn[MAXN], low[MAXN], idx;
d39d bool cut[MAXN];
427e
bfab void add_edge(int u, int v) {
c71a     adj[u].push_back(v);
a717     adj[v].push_back(u);
}                                                                  95cf
                                                                   427e
void tarjan(int u, int fa) {                                       50aa
    dfn[u] = low[u] = ++idx;                                       9891
    int child = 0;                                                 ec14
    for (int v : adj[u]) {                                         18f6
        if (!dfn[v]) {                                             3c64
            tarjan(v, fa); low[u] = min(low[u], low[v]);           9636
            if (low[v] >= dfn[u] && u != fa) cut[u] = true;        f368
            child += u == fa;                                      7923
        }                                                          95cf
        low[u] = min(low[u], dfn[v]);                              769a
    }                                                              95cf
    if (u == fa && child > 1) cut[u] = true;                       7927
}                                                                  95cf
```

5.4 Minimum spanning arborescence, faster

All vertices are 1-based. Clear the fields when reuse the struct.

Usage:

`add_edge(u, v, w)`	Add an edge from u to v with weight w.
`run(n, rt)`	Compute the total weight of MSA rooted at `rt`. If not exist, retun `LLONG_MIN`.

Time Complexity: $O((|E| + |V| \log |V|) \log |V|)$

```
const int MAXN = 300005;                                                      5ece
typedef pair<LL, int> pii;                                                    2fef
struct MDST {                                                                 1495
    priority_queue<pii, vector<pii>, greater<pii>> heap[MAXN];                01b2
    LL shift[MAXN];                                                           321d
    int fa[MAXN], vis[MAXN];                                                  fc06
                                                                              427e
    int find(int x) { return fa[x] == x ? x : fa[x] = find(fa[x]); }          38dd
                                                                              427e
    void unite(int x, int y) {                                                29b0
        x = find(x); y = find(y); fa[y] = x; if (x == y) return;              0c14
        if (heap[x].size() < heap[y].size()) {                                6fa0
            swap(heap[x], heap[y]);                                           9c26
            swap(shift[x], shift[y]);                                         2ffc
        }                                                                     95cf
        while (heap[y].size()) {                                              9959
            auto p = heap[y].top(); heap[y].pop();                            175b
            heap[x].emplace(p.first - shift[y] + shift[x], p.second);         c0c5
```

```
95cf         }
95cf     }
427e
0bbd     void add_edge(int u, int v, LL w) { heap[v].emplace(w, u); }
427e
a526     LL run(int n, int rt) {
f7ff         LL ans = 0;
81f2         iota(fa, fa + n + 1, 0);
19b3         Rep (i, n) if (find(i) != find(rt)) {
a7b1             int u = find(i);
010e             stack<int, vector<int>> s;
eff5             while (find(u) != find(rt)) {
0dda                 if (vis[u]) while (s.top() != u) {
c593                     vis[s.top()] = 0; unite(u, s.top()); s.pop();
83c4                 } else { vis[u] = 1; s.push(u); }
c76e                 while (heap[u].size()) {
b385                     ans += heap[u].top().first - shift[u];
dde2                     shift[u] = heap[u].top().first;
da47                     if (find(heap[u].top().second) != u) break;
9fbb                     heap[u].pop();
95cf                 }
6961                 if (heap[u].empty()) return LLONG_MIN;
87e6                 u = find(heap[u].top().second);
95cf             }
2d46             while (s.size()) { vis[s.top()] = 0; unite(rt, s.top()); s.pop(); }
95cf         }
4206         return ans;
95cf     }
329b };
```

5.5 Maximum flow (Dinic)

Usage:

`add_edge(u, v, c)`	Add an edge from u to v with capacity c.
`max_flow(s, t)`	Compute maximum flow from s to t.

Time Complexity: For general graph, $O(V^2E)$; for network with unit capacity, $O(\min\{V^{2/3}, \sqrt{E}\}E)$; for bipartite network, $O(\sqrt{V}E)$.

```
bcf8 struct edge{
60e2     int from, to;
5e6d     LL cap, flow;
329b };
427e
```

```
const int MAXN = 1005;                                              e2cd
struct Dinic {                                                      9062
    int n, m, s, t;                                                 4dbf
    vector<edge> edges;                                             9f0c
    vector<int> G[MAXN];                                            b891
    bool vis[MAXN];                                                 bbb6
    int d[MAXN];                                                    b40a
    int cur[MAXN];                                                  ddec
                                                                    427e
    void add_edge(int from, int to, LL cap) {                       5973
        edges.push_back(edge{from, to, cap, 0});                    7b55
        edges.push_back(edge{to, from, 0, 0});                      1db7
        m = edges.size();                                           fe77
        G[from].push_back(m-2);                                     dff5
        G[to].push_back(m-1);                                       8f2d
    }                                                               95cf
                                                                    427e
    bool bfs() {                                                    1836
        memset(vis, 0, sizeof(vis));                                3b73
        queue<int> q;                                               93d2
        q.push(s);                                                  5d13
        vis[s] = 1;                                                 2cd2
        d[s] = 0;                                                   721d
        while (!q.empty()) {                                        cc78
            int x = q.front(); q.pop();                             66ba
            for (int i = 0; i < G[x].size(); i++) {                 3b61
                edge& e = edges[G[x][i]];                           b510
                if (!vis[e.to] && e.cap > e.flow) {                 bba9
                    vis[e.to] = 1;                                  cd72
                    d[e.to] = d[x] + 1;                             cf26
                    q.push(e.to);                                   ca93
                }                                                   95cf
            }                                                       95cf
        }                                                           95cf
        return vis[t];                                              b23b
    }                                                               95cf
                                                                    427e
    LL dfs(int x, LL a) {                                           9252
        if (x == t || a == 0) return a;                             6904
        LL flow = 0, f;                                             8bf9
        for (int& i = cur[x]; i < G[x].size(); i++) {               f515
            edge& e = edges[G[x][i]];                               b510
            if(d[x] + 1 == d[e.to] && (f = dfs(e.to, min(a, e.cap-e.flow))) > 0)  2374
                {
```

```cpp
                e.flow += f;
                edges[G[x][i]^1].flow -= f;
                flow += f;
                a -= f;
                if(a == 0) break;
            }
        }
        return flow;
    }

    LL max_flow(int s, int t) {
        this->s = s; this->t = t;
        LL flow = 0;
        while (bfs()) {
            memset(cur, 0, sizeof(cur));
            flow += dfs(s, LLONG_MAX);
        }
        return flow;
    }

    vector<int> min_cut() { // call this after maxflow
        vector<int> ans;
        for (int i = 0; i < edges.size(); i++) {
            edge& e = edges[i];
            if(vis[e.from] && !vis[e.to] && e.cap > 0) ans.push_back(i);
        }
        return ans;
    }
};
```

5.6 Maximum cardinality bipartite matching (Hungarian)

```cpp
#include <bits/stdc++.h>
using namespace std;

#define rep(i, n) for (int i = 0; i < (n); i++)
#define Rep(i, n) for (int i = 1; i <= (n); i++)
#define range(x) (x).begin(), (x).end()
typedef long long LL;

struct Hungarian{
    int nx, ny;
    vector<int> mx, my;
    vector<vector<int> > e;
    vector<bool> mark;

    void init(int nx, int ny){
        this->nx = nx;
        this->ny = ny;
        mx.resize(nx); my.resize(ny);
        e.clear(); e.resize(nx);
        mark.resize(nx);
    }

    inline void add(int a, int b){
        e[a].push_back(b);
    }

    bool augment(int i){
        if (!mark[i]) {
            mark[i] = true;
            for (int j : e[i]){
                if (my[j] == -1 || augment(my[j])){
                    mx[i] = j; my[j] = i;
                    return true;
                }
            }
        }
        return false;
    }

    int match(){
        int ret = 0;
        fill(range(mx), -1);
        fill(range(my), -1);
        rep (i, nx){
            fill(range(mark), false);
            if (augment(i)) ret++;
        }
        return ret;
    }
};
```

5.7 Maximum matching of general graph (Edmond's blossom)

Usage:

`init(n)`	Initialize the template with n vertices, numbered from 1.
`add_edge(u, v)`	Add an undirected edge uv.
`solve()`	Find the maximum matching. Return the number of matched edges.
`mate[]`	The mate of a matched vertex. If it is not matched, then the value is 0.

Time Complexity: $O(|V|^3)$, but extremely fast in practice.

```
c041 const int MAXN = 1024;
6ab1 struct Blossom {
0b32     vector<int> adj[MAXN];
93d2     queue<int> q;
5c83     int n;
0de2     int label[MAXN], mate[MAXN], save[MAXN], used[MAXN];
427e
2186     void init(int nv) {
3728         n = nv; for (auto& v : adj) v.clear();
477d         fill(range(label), 0); fill(range(mate), 0);
bb35         fill(range(save), 0); fill(range(used), 0);
95cf     }
427e
c2dd     void add_edge(int u, int v) { adj[u].push_back(v); adj[v].push_back(u); }
427e
2a48     void rematch(int x, int y) {
8af8         int m = mate[x]; mate[x] = y;
1aa4         if (mate[m] == x) {
f4ba             if (label[x] <= n) {
740a                 mate[m] = label[x]; rematch(label[x], m);
8e2e             } else {
3341                 int a = 1 + (label[x] - n - 1) / n;
2885                 int b = 1 + (label[x] - n - 1) % n;
ef33                 rematch(a, b); rematch(b, a);
95cf             }
95cf         }
95cf     }
427e
8a50     void traverse(int x) {
43c0         Rep (i, n) save[i] = mate[i];
2ef7         rematch(x, x);
34d7         Rep (i, n) {
62c5             if (mate[i] != save[i]) used[i] ++;
                 mate[i] = save[i];                                                 97ef
             }                                                                      95cf
         }                                                                          95cf
                                                                                    427e
         void relabel(int x, int y) {                                               8bf8
             Rep (i, n) used[i] = 0;                                                d101
             traverse(x); traverse(y);                                              c4ea
             Rep (i, n) {                                                           34d7
                 if (used[i] == 1 and label[i] < 0) {                               dee9
                     label[i] = n + x + (y - 1) * n;                                1c22
                     q.push(i);                                                     eb31
                 }                                                                  95cf
             }                                                                      95cf
         }                                                                          95cf
                                                                                    427e
         int solve() {                                                              a0ce
             Rep (i, n) {                                                           34d7
                 if (mate[i]) continue;                                             a073
                 Rep (j, n) label[j] = -1;                                          1fc0
                 label[i] = 0; q = queue<int>(); q.push(i);                         7676
                 while (q.size()) {                                                 1c7d
                     int x = q.front(); q.pop();                                    66ba
                     for (int y : adj[x]) {                                         b98c
                         if (mate[y] == 0 and i != y) {                             c07f
                             mate[y] = x; rematch(x, y); q = queue<int>(); break;   7f36
                         }                                                          95cf
                         if (label[y] >= 0) { relabel(x, y); continue; }            d315
                         if (label[mate[y]] < 0) {                                  58ec
                             label[mate[y]] = x; q.push(mate[y]);                   c9c4
                         }                                                          95cf
                     }                                                              95cf
                 }                                                                  95cf
             }                                                                      95cf
             int cnt = 0;                                                           8abb
             Rep (i, n) cnt += (mate[i] > i);                                       b52f
             return cnt;                                                            6808
         }                                                                          95cf
     };                                                                             329b
```

5.8 Minimum cost maximum flow

```
struct edge{                                                                        bcf8
```

```
60e2     int from, to;
d698     int cap, flow;
32cc     LL cost;
329b };
427e
cc3e const LL INF = LLONG_MAX / 2;
2aa8 const int MAXN = 5005;
c6cb struct MCMF {
9ceb     int s, t, n, m;
9f0c     vector<edge> edges;
b891     vector<int> G[MAXN];
f74f     bool inq[MAXN]; // queue
8f67     LL d[MAXN];     // distance
9524     int p[MAXN];    // previous
b330     int a[MAXN];    // improvement
427e
f7f2     void add_edge(int from, int to, int cap, LL cost) {
24f0         edges.push_back(edge{from, to, cap, 0, cost});
95f0         edges.push_back(edge{to, from, 0, 0, -cost});
fe77         m = edges.size();
dff5         G[from].push_back(m-2);
8f2d         G[to].push_back(m-1);
95cf     }
427e
3c52     bool spfa(){
93d2         queue<int> q;
8494         fill(d, d + MAXN, INF); d[s] = 0;
fd48         memset(inq, 0, sizeof(inq));
5e7c         q.push(s); inq[s] = true;
2dae         p[s] = 0; a[s] = INT_MAX;
cc78         while (!q.empty()){
b0aa             int u = q.front(); q.pop(); inq[u] = false;
3bba             for (int i : G[u]) {
56d8                 edge& e = edges[i];
3601                 if (e.cap > e.flow && d[e.to] > d[u] + e.cost){
55bc                     d[e.to] = d[u] + e.cost;
0bea                     p[e.to] = G[u][i];
8249                     a[e.to] = min(a[u], e.cap - e.flow);
e5d3                     if (!inq[e.to]) q.push(e.to), inq[e.to] = true;
95cf                 }
95cf             }
95cf         }
6d7c         return d[t] != INF;
95cf     }
```

```
                                                                427e
    void augment(){                                             71a4
        int u = t;                                              06f1
        while (u != s){                                         b19d
            edges[p[u]].flow += a[t];                           db09
            edges[p[u]^1].flow -= a[t];                         25a9
            u = edges[p[u]].from;                               e6c9
        }                                                       95cf
    }                                                           95cf
                                                                427e
#ifdef GIVEN_FLOW                                               6e20
    bool min_cost(int s, int t, int f, LL& cost) {              5972
        this->s = s; this->t = t;                               590d
        int flow = 0;                                           21d4
        cost = 0;                                               23cb
        while (spfa()) {                                        22dc
            augment();                                          bcdb
            if (flow + a[t] >= f){                              a671
                cost += (f - flow) * d[t]; flow = f;            b14d
                return true;                                    3361
            } else {                                            8e2e
                flow += a[t]; cost += a[t] * d[t];              2a83
            }                                                   95cf
        }                                                       95cf
        return false;                                           438e
    }                                                           95cf
#else                                                           a8cb
    int min_cost(int s, int t, LL& cost) {                      f9a9
        this->s = s; this->t = t;                               590d
        int flow = 0;                                           21d4
        cost = 0;                                               23cb
        while (spfa()) {                                        22dc
            augment();                                          bcdb
            flow += a[t]; cost += a[t] * d[t];                  2a83
        }                                                       95cf
        return flow;                                            84fb
    }                                                           95cf
#endif                                                          1937
};                                                              329b
```

5.9 Fast LCA

All indices of the tree are 1-based.

Usage:

`preprocess(root)`	Initialize with tree rooted at `root`.
`lca(u, v)`	Query the lowest common ancestor of u and v.

```
const int MAXN = 500005;
vector<int> adj[MAXN];
int id[MAXN], nid;
pair<int, int> st[MAXN << 1][33 - __builtin_clz(MAXN)];

void dfs(int u, int p, int d) {
    st[id[u] = nid++][0] = {d, u};
    for (int v : adj[u]) {
        if (v == p) continue;
        dfs(v, u, d + 1);
        st[nid++][0] = {d, u};
    }
}

void preprocess(int root) {
    nid = 0;
    dfs(root, 0, 1);
    int l = 31 - __builtin_clz(nid);
    rep (j, l) rep (i, 1+nid-(1<<j))
        st[i][j+1] = min(st[i][j], st[i+(1<<j)][j]);
}

int lca(int u, int v) {
    tie(u, v) = minmax(id[u], id[v]);
    int k = 31 - __builtin_clz(v-u+1);
    return min(st[u][k], st[v-(1<<k)+1][k]).second;
}
```

5.10 Heavy-light decomposition

Time Complexity: The decomposition itself takes linear time. Each query takes $O(\log n)$ operations.

```
const int MAXN = 100005;
vector<int> adj[MAXN];
int sz[MAXN], top[MAXN], fa[MAXN], son[MAXN], depth[MAXN], id[MAXN];

void dfs1(int x, int dep, int par){
    depth[x] = dep;
    sz[x] = 1;
    fa[x] = par;
    int maxn = 0, s = 0;
    for (int c: adj[x]){
        if (c == par) continue;
        dfs1(c, dep + 1, x);
        sz[x] += sz[c];
        if (sz[c] > maxn){
            maxn = sz[c];
            s = c;
        }
    }
    son[x] = s;
}

int cid = 0;
void dfs2(int x, int t){
    top[x] = t;
    id[x] = ++cid;
    if (son[x]) dfs2(son[x], t);
    for (int c: adj[x]){
        if (c == fa[x]) continue;
        if (c == son[x]) continue;
        else dfs2(c, c);
    }
}

void decomp(int root){
    dfs1(root, 1, 0);
    dfs2(root, root);
}

void query(int u, int v){
    while (top[u] != top[v]){
        if (depth[top[u]] < depth[top[v]]) swap(u, v);
        // id[top[u]] to id[u]
        u = fa[top[u]];
    }
    if (depth[u] > depth[v]) swap(u, v);
    // id[u] to id[v]
}
```

5.11 Centroid decomposition

Note that the centroid here is not the exact centroid of the graph. It only guarantees that the size of each subtree does not exceed half of that of the original tree. This is enough to guarantee the correct time complexity. All vertices are numbered from 1. Call `decomp(root)` to use.

Usage:

`decomp(u, p)`	Decompose the tree rooted at u with parent p.

Time Complexity: The decomposition itself takes $O(n \log n)$ time.

```cpp
vector<int> adj[100005];
int sz[100005], sum;

void getsz(int u, int p) {
  sz[u] = 1; sum++;
  for (int v : adj[u]) {
    if (v == p) continue;
    getsz(v, u);
    sz[u] += sz[v];
  }
}

int getcent(int u, int p) {
  for (int v : adj[u])
    if (v != p and sz[v] > sum / 2)
      return getcent(v, u);
  return u;
}

void decompose(int u) {
  sum = 0; getsz(u, 0);
  u = getcent(u, 0); // update u to the centroid

  for (int v : adj[u]) {
    // get answer for subtree v
  }
  // get answer for the whole tree
  // don't forget to count the centroid itself

  for (int v : adj[u]) { // divide and conquer
    adj[v].erase(find(range(adj[v]), u));
    decompose(v);
    adj[v].push_back(u); // restore deleted edge
  }
}
```

5.12 DSU on tree

This implementation avoids parallel existence of multiple data structures but requires that the data structure is invertible. To use this template, implement `merge`, `enter`, `leave` as needed; first call `decomp(root, 0)`, then call `work(root, 0, false)`. Labels of vertices start from 1.

Usage:

`decomp(u, p)`	Decompose the tree u.
`work(u, p, keep)`	Work for subtree u. When `keep` is set, information is not cleared.

Time Complexity: $O(n \log n)$ times the complexity for `merge`, `enter`, `leave`.

```cpp
vector<int> adj[100005];
int sz[100005], son[100005];

void decomp(int u, int p) {
    sz[u] = 1;
    for (int v : adj[u]) {
        if (v == p) continue;
        decomp(v, u);
        sz[u] += sz[v];
        if (sz[v] > sz[son[u]]) son[u] = v;
    }
}

template <typename T>
void trav(T fn, int u, int p) {
    fn(u);
    for (int v : adj[u]) if (v != p) trav(fn, v, u);
}

#define for_light(v) for (int v : adj[u]) if (v != p and v != son[u])
void work(int u, int p, bool keep) {
    for_light(v) work(v, u, 0); // process light children

    // process heavy child
    // current data structure contains info of heavy child
    if (son[u]) work(son[u], u, 1);

    auto merge = [u] (int c) { /* count contribution of c */ };
```

```
1ab0     auto enter = [] (int c) { /* add vertex c */ };
f241     auto leave = [] (int c) { /* remove vertex c*/ };
427e
3d3b     for_light(v) {
74c6         trav(merge, v, u);
c13d         trav(enter, v, u);
95cf     }
427e
427e     // count answer for root and add it
427e     // Warning: special check may apply to root!
c54f     merge(u);
9dec     enter(u);
427e
427e     // leave current tree
4e3e     if (!keep) trav(leave, u, p);
95cf }
```

6 Data Structures

6.1 Fenwick tree (point update range query)

```
9976 struct bit_purq { // point update, range query
d7af     int N;
99ff     vector<LL> tr;
427e
456d     void init(int n) { tr.resize(N = n + 5); }
427e
63d0     LL sum(int n) {
f7ff         LL ans = 0;
6770         while (n) { ans += tr[n]; n &= n - 1; }
4206         return ans;
95cf     }
427e
f4bd     void add(int n, LL x){
968e         while (n < N) { tr[n] += x; n += n & -n; }
95cf     }
329b };
```

6.2 Fenwick tree (range update point query)

```
struct bit_rupq{ // range update, point query                    3d03
    int N;                                                        d7af
    vector<LL> tr;                                                99ff
                                                                  427e
    void init(int n) { tr.resize(N = n + 5);}                     456d
                                                                  427e
    LL query(int n) {                                             38d4
        LL ans = 0;                                               f7ff
        while (n < N) { ans += tr[n]; n += n & -n; }              3667
        return ans;                                               4206
    }                                                             95cf
                                                                  427e
    void add(int n, LL x) {                                       f4bd
        while (n) { tr[n] += x; n &= n - 1; }                     0a2b
    }                                                             95cf
};                                                                329b
```

6.3 Segment tree

```
LL p;                                                             3942
const int MAXN = 4 * 100006;                                      1ebb
struct segtree {                                                  451a
  int l[MAXN], m[MAXN], r[MAXN];                                  27be
  LL val[MAXN], tadd[MAXN], tmul[MAXN];                           4510
                                                                  427e
#define lson (o<<1)                                               ac35
#define rson (o<<1|1)                                             1294
                                                                  427e
  void pull(int o) {                                              1344
    val[o] = (val[lson] + val[rson]) % p;                         bbe9
  }                                                               95cf
                                                                  427e
  void push_add(int o, LL x) {                                    e4bc
    val[o] = (val[o] + x * (r[o] - l[o])) % p;                    5dd6
    tadd[o] = (tadd[o] + x) % p;                                  6eff
  }                                                               95cf
                                                                  427e
  void push_mul(int o, LL x) {                                    d658
    val[o] = val[o] * x % p;                                      b82c
    tadd[o] = tadd[o] * x % p;                                    aa86
    tmul[o] = tmul[o] * x % p;                                    649f
```

```
95cf   }
427e
b149   void push(int o) {
3159     if (l[o] == m[o]) return;
0a90     if (tmul[o] != 1) {
0f4a       push_mul(lson, tmul[o]);
045e       push_mul(rson, tmul[o]);
ac0a       tmul[o] = 1;
95cf     }
1b82     if (tadd[o]) {
9547       push_add(lson, tadd[o]);
0e73       push_add(rson, tadd[o]);
6234       tadd[o] = 0;
95cf     }
95cf   }
427e
471c   void build(int o, int ll, int rr) {
0e87     int mm = (ll + rr) / 2;
9d27     l[o] = ll; r[o] = rr; m[o] = mm;
ac0a     tmul[o] = 1;
5c92     if (ll == mm) {
001f       scanf("%lld", val + o);
e5b6       val[o] %= p;
8e2e     } else {
7293       build(lson, ll, mm);
5e67       build(rson, mm, rr);
ba26       pull(o);
95cf     }
95cf   }
427e
4406   void add(int o, int ll, int rr, LL x) {
3c16     if (ll <= l[o] && r[o] <= rr) {
db32       push_add(o, x);
8e2e     } else {
c4b0       push(o);
4305       if (m[o] > ll) add(lson, ll, rr, x);
d5a6       if (m[o] < rr) add(rson, ll, rr, x);
ba26       pull(o);
95cf     }
95cf   }
427e
48cd   void mul(int o, int ll, int rr, LL x) {
3c16     if (ll <= l[o] && r[o] <= rr) {
e7d0       push_mul(o, x);
```

```
    } else {                                              8e2e
      push(o);                                            c4b0
      if (ll < m[o]) mul(lson, ll, rr, x);                d1ba
      if (m[o] < rr) mul(rson, ll, rr, x);                67f3
      pull(o);                                            ba26
    }                                                     95cf
  }                                                       95cf
                                                          427e
  LL query(int o, int ll, int rr) {                       0f62
    if (ll <= l[o] && r[o] <= rr) {                       3c16
      return val[o];                                      6dfe
    } else {                                              8e2e
      push(o);                                            c4b0
      if (rr <= m[o]) return query(lson, ll, rr);         462a
      if (ll >= m[o]) return query(rson, ll, rr);         5cca
      return query(lson, ll, rr) + query(rson, ll, rr);   bbf9
    }                                                     95cf
  }                                                       95cf
} seg;                                                    4d99
```

6.4 Treap

Self-balanced binary search tree which supports split and merge.

Usage:

`push(x)`	Push lazy tags to children.
`pull(x)`	Update statistics of node x.
`Init(x, v)`	Initialize node x with value v.
`Add(x, v)`	Apply addition to subtree x.
`Reverse(x)`	Apply reversion to subtree x.
`Merge(x, y)`	Merge trees rooted at x and y. Return the root of new tree.
`Split(t, k, x, y)`	Split out the left k elements of tree t. The roots of left part and right part are stored in x an d y, respectively.
`init(n)`	Initialize the treap with array of size n.
`work(op, l, r)`	Range operation over $[l, r)$.

Time Complexity: Expected $O(\log n)$ per operation.

```
const int MAXN = 200005;                                  9f60
mt19937 gen(time(NULL));                                  a7c5
struct Treap {                                            9542
    int ch[MAXN][2];                                      6d61
    int sz[MAXN], key[MAXN], val[MAXN];                   3948
    int add[MAXN], rev[MAXN];                             5d9a
```

```
2b1b     LL sum[MAXN] = {0};
a773     int maxv[MAXN] = {INT_MIN}, minv[MAXN] = {INT_MAX};
427e
a629     void Init(int x, int v) {
5a00         ch[x][0] = ch[x][1] = 0;
d8cd         key[x] = gen(); val[x] = v; pull(x);
95cf     }
427e
3bf9     void pull(int x) {
e1c3         sz[x] = 1 + sz[ch[x][0]] + sz[ch[x][1]];
99f8         sum[x] = val[x] + sum[ch[x][0]] + sum[ch[x][1]];
94e9         maxv[x] = max({val[x], maxv[ch[x][0]], maxv[ch[x][1]]});
6bb9         minv[x] = min({val[x], minv[ch[x][0]], minv[ch[x][1]]});
95cf     }
427e
8c8e     void Add(int x, int a) {
a7b1         val[x] += a; add[x] += a;
832a         sum[x] += LL(sz[x]) * a; maxv[x] += a; minv[x] += a;
95cf     }
427e
aaf6     void Reverse(int x) {
52c6         rev[x] ^= 1;
7850         swap(ch[x][0], ch[x][1]);
95cf     }
427e
1a53     void push(int x) {
5fe5         for (int c : ch[x]) if (c) {
fd76             Add(c, add[x]);
7a53             if (rev[x]) Reverse(c);
95cf         }
49ee         add[x] = 0; rev[x] = 0;
95cf     }
427e
9d2c     int Merge(int x, int y) {
1b09         if (!x || !y) return x | y;
cd7e         push(x); push(y);
bffa         if (key[x] > key[y]) {
a3df             ch[x][1] = Merge(ch[x][1], y); pull(x); return x;
8e2e         } else {
bf9e             ch[y][0] = Merge(x, ch[y][0]); pull(y); return y;
95cf         }
95cf     }
427e
dc7e     void Split(int t, int k, int &x, int &y) {
```

```
        if (t == 0) { x = y = 0; return; }                                  6303
        push(t);                                                            f26b
        if (sz[ch[t][0]] < k) {                                             3465
            x = t; Split(ch[t][1], k - sz[ch[t][0]] - 1, ch[t][1], y);      ffd8
        } else {                                                            8e2e
            y = t; Split(ch[t][0], k, x, ch[t][0]);                         8a23
        }                                                                   95cf
        if (x) pull(x); if (y) pull(y);                                     89e3
    }                                                                       95cf
} treap;                                                                    b1f4
                                                                            427e
int root;                                                                   24b6
                                                                            427e
void init(int n) {                                                          d34f
    Rep (i, n) {                                                            34d7
        int x; scanf("%d", &x);                                             7681
        treap.Init(i, x);                                                   0ed8
        root = (i == 1) ? 1 : treap.Merge(root, i);                         bcc8
    }                                                                       95cf
}                                                                           95cf
                                                                            427e
void work(int op, int l, int r) {                                           d030
    int tl, tm, tr;                                                         6639
    treap.Split(root, l, tl, tm);                                           b6c4
    treap.Split(tm, r - l, tm, tr);                                         8de3
    if (op == 1) {                                                          3658
        int x; scanf("%d", &x); treap.Add(tm, x);                           c039
    } else if (op == 2) {                                                   1dcb
        treap.Reverse(tm);                                                  ae78
    } else if (op == 3) {                                                   581d
        printf("%lld␣%d␣%d\n",                                              e092
               treap.sum[tm], treap.minv[tm], treap.maxv[tm]);              867f
    }                                                                       95cf
    root = treap.Merge(treap.Merge(tl, tm), tr);                            6188
}                                                                           95cf
```

6.5 Link/cut tree

Dynamic connectivity of undirected acyclic graph. Support single-vertex update, path aggregation and relative LCA query. Vertices are numbered from 1. Zero initialization is enough except for the statistic information.

Usage:

`pull(x)`	Update statistics of node x.
`Root(u)`	Get the root of tree where vertex u is in.
`Link(u, v)`	Link two unconnected trees.
`Cut(u, v)`	Cut an existent edge.
`Query(u, v)`	Path aggregation.
`Update(u, x)`	Single point modification.
`LCA(u, v, root)`	Get the lowest common ancestor of u and v in tree rooted at `root`.

Time Complexity: $O(\log n)$ per operation

```
2e73 const int MAXN = 1000005;
ca06 struct LCT {
6a6d     int fa[MAXN], ch[MAXN][2], val[MAXN], sum[MAXN];
c6e1     bool rev[MAXN];
427e
eba3     bool isroot(int x) { return ch[fa[x]][0] == x || ch[fa[x]][1] == x; }
f19f     void pull(int x) { sum[x] = val[x] ^ sum[ch[x][0]] ^ sum[ch[x][1]]; }
1c4d     void reverse(int x) { swap(ch[x][0], ch[x][1]); rev[x] ^= 1; }
1a53     void push(int x) {
89a0         if (rev[x]) rep (i, 2) if (ch[x][i]) reverse(ch[x][i]); rev[x] = 0;
95cf     }
425f     void rotate(int x) {
51af         int y = fa[x], z = fa[y], k = ch[y][1] == x, w = ch[x][!k];
e1fe         if (isroot(y)) ch[z][ch[z][1] == y] = x;
1e6f         ch[x][!k] = y; ch[y][k] = w; if (w) fa[w] = y;
6d09         fa[y] = x; fa[x] = z; pull(y);
95cf     }
52c6     void pushall(int x) { if (isroot(x)) pushall(fa[x]); push(x); }
f69c     void splay(int x) {
d095         int y = x, z = 0;
c494         for (pushall(y); isroot(x); rotate(x)) {
ceef             y = fa[x]; z = fa[y];
4449             if (isroot(y)) rotate((ch[y][0] == x) ^ (ch[z][0] == y) ? x : y);
95cf         }
78a0         pull(x);
95cf     }
6229     void access(int x) {
1548         int z = x;
8854         for (int y = 0; x; x = fa[y = x]) { splay(x); ch[x][1] = y; pull(x); }
7afd         splay(z);
95cf     }
a067     void chroot(int x) { access(x); reverse(x); }
126d     void split(int x, int y) { chroot(x); access(y); }
427e
```

```
    int Root(int x) {                                                          d87a
        for (access(x); ch[x][0]; x = ch[x][0]) push(x);                       f4f1
        splay(x); return x;                                                    0d77
    }                                                                          95cf
    void Link(int u, int v) { chroot(u); fa[u] = v; }                          9e46
    void Cut(int u, int v) { split(u, v); fa[u] = ch[v][0] = 0; pull(v); }     7c10
    int Query(int u, int v) { split(u, v); return sum[v]; }                    0691
    void Update(int u, int x) { splay(u); val[u] = x; }                        a999
    int LCA(int x, int y, int root) {                                          1f42
        chroot(root); access(x); splay(y);                                     6cb2
        while (fa[y]) splay(y = fa[y]);                                        02e5
        return y;                                                              c218
    }                                                                          95cf
};                                                                             329b
```

6.6 Balanced binary search tree from `pb_ds`

```
#include <ext/pb_ds/assoc_container.hpp>                                       0475
using namespace __gnu_pbds;                                                    332d
                                                                               427e
tree<int, null_type, less<int>, rb_tree_tag, tree_order_statistics_node_update>  43a7
  rkt;
// null_tree_node_update                                                       427e
                                                                               427e
// SAMPLE USAGE                                                                427e
rkt.insert(x);          // insert element                                      190e
rkt.erase(x);           // erase element                                       05d4
rkt.order_of_key(x);    // obtain the number of elements less than x           add5
rkt.find_by_order(i);   // iterator to i-th (numbered from 0) smallest element  b064
rkt.lower_bound(x);                                                            c103
rkt.upper_bound(x);                                                            4ff4
rkt.join(rkt2);         // merge tree (only if their ranges do not intersect)  b19b
rkt.split(x, rkt2);     // split all elements greater than x to rkt2           cb47
```

6.7 Persistent segment tree, range k-th query

```
struct node {                                                                  f1a7
  static int n, pos;                                                           2ff6
                                                                               427e
  int value;                                                                   7cec
  node *left, *right;                                                          70e2
```

```
427e 
20b0   void* operator new(size_t size);
427e 
3dc0   static node* Build(int l, int r) {
b6c5     node* a = new node;
ce96     if (r > l + 1) {
181e       int mid = (l + r) / 2;
3ba2       a->left = Build(l, mid);
8aaf       a->right = Build(mid, r);
8e2e     } else {
bfc4       a->value = 0;
95cf     }
5ffd     return a;
95cf   }
427e 
5a45   static node* init(int size) {
2c46     n = size;
7ee3     pos = 0;
be52     return Build(0, n);
95cf   }
427e 
93c0   static int Query(node* lt, node *rt, int l, int r, int k) {
d30c     if (r == l + 1) return l;
181e     int mid = (l + r) / 2;
cb5a     if (rt->left->value - lt->left->value < k) {
8edb       k -= rt->left->value - lt->left->value;
2412       return Query(lt->right, rt->right, mid, r, k);
8e2e     } else {
0119       return Query(lt->left, rt->left, l, mid, k);
95cf     }
95cf   }
427e 
c9ad   static int query(node* lt, node *rt, int k) {
9e27     return Query(lt, rt, 0, n, k);
95cf   }
427e 
b19c   node *Inc(int l, int r, int pos) const {
5794     node* a = new node(*this);
ce96     if (r > l + 1) {
181e       int mid = (l + r) / 2;
203d       if (pos < mid)
f44a         a->left = left->Inc(l, mid, pos);
649a       else
1024         a->right = right->Inc(mid, r, pos);
95cf     }
2b3e     a->value++;
5ffd     return a;
95cf   }
427e 
e80f   node *inc(int index) {
c246     return Inc(0, n, index);
95cf   }
865a } nodes[8000000];
427e 
99ce int node::n, node::pos;
1987 inline void* node::operator new(size_t size) {
bb3c   return nodes + (pos++);
95cf }
```

6.8 Block list

All indices are 0-based. All ranges are left-closed right-open.

Usage:

`block::fix()`	Apply tags to the current block.
`Init(l, r)`	Range initializer.
`Reverse(l, r)`	Reverse the range.
`Add(l, r, x)`	Add x to the range.
`Query(l, r)`	Range aggregation.

```
fd9e const int BLOCK = 800;
76b3 typedef vector<int> vi;
427e 
a771 struct block {
8fbc     vi data;
e3b5     LL sum; int minv, maxv;
41db     int add; bool rev;
427e 
d7eb     block(vi&& vec) : data(move(vec)),
1f0c         sum(accumulate(range(data), 0ll)),
8216         minv(*min_element(range(data))),
527d         maxv(*max_element(range(data))),
6437         add(0), rev(0) { }
427e 
b919     void fix() {
0694         if (rev) reverse(range(data));          rev = 0;
0527         if (add) for (int& x : data) x += add;  add = 0;
95cf     }
```

```
427e
8bc4     void merge(block& another) {
b895         fix(); another.fix();
f516         vi temp(move(data));
d02c         temp.insert(temp.end(), range(another.data));
88ea         *this = block(move(temp));
95cf     }
427e
42e8     block split(int pos) {
3e79         fix();
ccab         block result(vi(data.begin() + pos, data.end()));
861a         data.resize(pos); *this = block(move(data));
56b0         return result;
95cf     }
329b };
427e
2a18 typedef list<block>::iterator lit;
427e
ce14 struct blocklist {
5540     list<block> blk;
427e
7b8e     void maintain() {
3131         lit it = blk.begin();
4628         while (it != blk.end() && next(it) != blk.end()) {
852d             lit it2 = it;
188c             while (next(it2) != blk.end() &&
3600                     it2->data.size() + next(it2)->data.size() <= BLOCK) {
93e1                 it2->merge(*next(it2));
e1fa                 blk.erase(next(it2));
95cf             }
5771             ++it;
95cf         }
95cf     }
427e
b7b3     lit split(int pos) {
2273         for (lit it = blk.begin(); ; it++) {
5502             if (pos == 0) return it;
8e85             while (it->data.size() > pos)
2099                 blk.insert(next(it), it->split(pos));
a5a1             pos -= it->data.size();
427e
95cf         }
95cf     }
427e
    void Init(int *l, int *r) {                                      1c7b
        for (int *cur = l; cur < r; cur += BLOCK)                    9919
            blk.emplace_back(vi(cur, min(cur + BLOCK, r)));          8950
    }                                                                95cf
                                                                     427e
    void Reverse(int l, int r) {                                     a22f
        lit it = split(l), it2 = split(r);                           997b
        reverse(it, it2);                                            dfd0
        while (it != it2) {                                          8f89
            it->rev ^= 1;                                            6a06
            it++;                                                    5283
        }                                                            95cf
        maintain();                                                  b204
    }                                                                95cf
                                                                     427e
    void Add(int l, int r, int x) {                                  3cce
        lit it = split(l), it2 = split(r);                           997b
        while (it != it2) {                                          8f89
            it->sum += LL(x) * it->data.size();                      e927
            it->minv += x; it->maxv += x;                            03d3
            it->add += x; it++;                                      4511
        }                                                            95cf
        maintain();                                                  b204
    }                                                                95cf
                                                                     427e
    void Query(int l, int r) {                                       3ad3
        lit it = split(l), it2 = split(r);                           997b
        LL sum = 0; int minv = INT_MAX, maxv = INT_MIN;              c33d
        while (it != it2) {                                          8f89
            sum += it->sum;                                          e472
            minv = min(minv, it->minv);                              72c4
            maxv = max(maxv, it->maxv);                              e1c4
            it++;                                                    5283
        }                                                            95cf
        maintain();                                                  b204
        printf("%lld %d %d\n", sum, minv, maxv);                     8792
    }                                                                95cf
} lst;                                                               958e
```

6.9 Persistent block list

Block list that supports persistence. All indices are 0-based. All ranges are left-closed right-open. `std::shared_ptr` is used to ease memory management. One should modify

the constructor of `block` to maintain extra information. Here we use this policy that the size of each block does not exceed `BLOCK`, while the sum of sizes of two adjacent blocks does not less than `BLOCK`.
When some operation that breaks block list property, please call `maintain` in time to restore the property.
Usage:

`maintain()`	Maintain the block list property.
`split(pos)`	Split the block list at position `pos`. Returns an iterator to a block starting at `pos`.
`sum(l, r)`	An example function of list traversal between $[l, r)$.

Time Complexity: When `BLOCK` is properly selected, the time complexity is $O(\sqrt{n})$ per operation.

```
a19e constexpr int BLOCK = 800;
76b3 typedef vector<int> vi;
0563 typedef shared_ptr<vi> pvi;
013b typedef shared_ptr<const vi> pcvi;
427e
a771 struct block {
2989     pcvi data;
8fd0     LL sum;
427e
427e     // add information to maintain
a613     block(pcvi ptr) :
24b5         data(ptr),
0cf0         sum(accumulate(ptr->begin(), ptr->end(), 0ll))
e93b     { }
427e
5c0f     void merge(const block& another) {
0b18         pvi temp = make_shared<vi>(data->begin(), data->end());
ac21         temp->insert(temp->end(), another.data->begin(), another.data->end());
6467         *this = block(temp);
95cf     }
427e
42e8     block split(int pos) {
dac1         block result(make_shared<vi>(data->begin() + pos, data->end()));
01db         *this = block(make_shared<vi>(data->begin(), data->begin() + pos));
56b0         return result;
95cf     }
329b };
427e
2a18 typedef list<block>::iterator lit;
427e
```

```
struct blocklist {                                                          ce14
    list<block> blk;                                                        5540
                                                                            427e
    void maintain() {                                                       7b8e
        lit it = blk.begin();                                               3131
        while (it != blk.end() and next(it) != blk.end()) {                 5e44
            lit it2 = it;                                                   852d
            while (next(it2) != blk.end() and                               0b03
                    it2->data->size() + next(it2)->data->size() <= BLOCK) { 029f
                it2->merge(*next(it2));                                     93e1
                blk.erase(next(it2));                                       e1fa
            }                                                               95cf
            ++it;                                                           5771
        }                                                                   95cf
    }                                                                       95cf
                                                                            427e
    lit split(int pos) {                                                    b7b3
        for (lit it = blk.begin(); ; it++) {                                2273
            if (pos == 0) return it;                                        5502
            while (it->data->size() > pos) {                                d480
                blk.insert(next(it), it->split(pos));                       2099
            }                                                               95cf
            pos -= it->data->size();                                        a1c8
        }                                                                   95cf
    }                                                                       95cf
                                                                            427e
    LL sum(int l, int r) { // traverse                                      fd38
        lit it1 = split(l), it2 = split(r);                                 48b4
        LL res = 0;                                                         ac09
        while (it1 != it2) {                                                9f1d
            res += it1->sum;                                                8284
            it1++;                                                          61fd
        }                                                                   95cf
        maintain();                                                         b204
        return res;                                                         244d
    }                                                                       95cf
};                                                                          329b
```

6.10 Sparse table, range minimum query

The array is 0-based and the range is left-closed right-open.

```
const int MAXN = 100007;                                                    db63
```

```cpp
int a[MAXN], st[MAXN][30];

void init(int n){
    int l = log2(n);
    rep (i, n) st[i][0] = a[i];
    rep (j, l) rep (i, 1+n-(1<<j))
        st[i][j+1] = min(st[i][j], st[i+(1<<j)][j]);
}

int rmq(int l, int r){
    int k = log2(r - l);
    return min(st[l][k], st[r-(1<<k)][k]);
}
```

7 Geometrics

7.1 2D geometric template

```cpp
#include <bits/stdc++.h>
using namespace std;

typedef int T;
typedef struct pt {
    T x, y;
    T operator , (pt a) { return x*a.x + y*a.y; } // inner product
    T operator * (pt a) { return x*a.y - y*a.x; } // outer product
    pt operator + (pt a) { return {x+a.x, y+a.y}; }
    pt operator - (pt a) { return {x-a.x, y-a.y}; }

    pt operator * (T k) { return {x*k, y*k}; }
    pt operator - () { return {-x, -y};}
} vec;

typedef pair<pt, pt> seg;

bool ptOnSeg(pt& p, seg& s){
    vec v1 = s.first - p, v2 = s.second - p;
    return (v1, v2) <= 0 && v1 * v2 == 0;
}

// 0 not on segment
// 1 on segment except vertices
// 2 on vertices
int ptOnSeg2(pt& p, seg& s){
    vec v1 = s.first - p, v2 = s.second - p;
    T ip = (v1, v2);
    if (v1 * v2 != 0 || ip > 0) return 0;
    return (v1, v2) ? 1 : 2;
}

// if two orthogonal rectangles do not touch, return true
inline bool nIntRectRect(seg a, seg b){
    return min(a.first.x, a.second.x) > max(b.first.x, b.second.x) ||
           min(a.first.y, a.second.y) > max(b.first.y, b.second.y) ||
           min(b.first.x, b.second.x) > max(a.first.x, a.second.x) ||
           min(b.first.y, b.second.y) > max(a.first.y, a.second.y);
}

// >0 in order
// <0 out of order
// =0 not standard
inline double rotOrder(vec a, vec b, vec c){return double(a*b)*(b*c);}

inline bool intersect(seg a, seg b){
    // ! if (nIntRectRect(a, b)) return false; // if commented, assume that a
      and b are non-collinear
    return rotOrder(b.first-a.first, a.second-a.first, b.second-a.first) >= 0 &&
           rotOrder(a.first-b.first, b.second-b.first, a.second-b.first) >= 0;
}

// 0 not insersect
// 1 standard intersection
// 2 vertex-line intersection
// 3 vertex-vertex intersection
// 4 collinear and have common point(s)
int intersect2(seg& a, seg& b){
    if (nIntRectRect(a, b)) return 0;
    vec va = a.second - a.first, vb = b.second - b.first;
    double j1 = rotOrder(b.first-a.first, va, b.second-a.first),
           j2 = rotOrder(a.first-b.first, vb, a.second-b.first);
    if (j1 < 0 || j2 < 0) return 0;
    if (j1 != 0 && j2 != 0) return 1;
    if (j1 == 0 && j2 == 0){
        if (va * vb == 0) return 4; else return 3;
    } else return 2;
```

```
95cf }
427e
2c68 template <typename Tp = T>
5894 inline pt getIntersection(pt P, vec v, pt Q, vec w){
6850     static_assert(is_same<Tp, double>::value, "must␣be␣double!");
7c9a     return P + v * (w*(P-Q)/(v*w));
95cf }
427e
427e // -1 outside the polygon
427e // 0  on the border of the polygon
427e // 1  inside the polygon
cbdd int ptOnPoly(pt p, pt* poly, int n){
5fb4     int wn = 0;
1294     for (int i = 0; i < n; i++) {
427e
3cae         T k, d1 = poly[i].y - p.y, d2 = poly[(i+1)%n].y - p.y;
b957         if (k = (poly[(i+1)%n] - poly[i])*(p - poly[i])){
8c40             if (k > 0 && d1 <= 0 && d2 > 0) wn++;
3c4d             if (k < 0 && d2 <= 0 && d1 > 0) wn--;
aad3         } else return 0;
95cf     }
0a5f     return wn ? 1 : -1;
95cf }
427e
d4a3 istream& operator >> (istream& lhs, pt& rhs){
fa86     lhs >> rhs.x >> rhs.y;
331a     return lhs;
95cf }
427e
07ae istream& operator >> (istream& lhs, seg& rhs){
5cab     lhs >> rhs.first >> rhs.second;
331a     return lhs;
95cf }
```

8 Appendices

8.1 Primes

8.1.1 First primes

p	$g(p)$	p	$g(p)$	p	$g(p)$	p	$g(p)$	p	$g(p)$
2	1	3	2	5	2	7	3	11	2
13	2	17	3	19	2	23	5	29	2
31	3	37	2	41	6	43	3	47	5
53	2	59	2	61	2	67	2	71	7
73	5	79	3	83	2	89	3	97	5
101	2	103	5	107	2	109	6	113	3
127	3	131	2	137	3	139	2	149	2
151	6	157	5	163	2	167	5	173	2
179	2	181	2	191	19	193	5	197	2
199	3	211	2	223	3	227	2	229	6

8.1.2 Arbitrary length primes

$\lg p$	p	$g(p)$	p	$g(p)$
3	967	5	1031	14
4	9859	2	10273	10
5	96331	10	102931	3
6	958543	6	1031137	5
7	9594539	2	10169651	2
8	96243449	3	103211039	7
9	980483981	2	1042484357	2
10	9858935453	2	10261276009	7
11	95748666809	3	101759940101	2
12	950781833849	3	1012797784423	5
13	9739822952371	7	10037217092377	7
14	96181051140397	5	104974966380359	11
15	981030138360889	13	1029038416465403	2
16	9655206098080843	3	10116299875820773	2
17	97687777921994419	3	101506415998163437	2

8.1.3 $\sim 1 \times 10^9$

p	$g(p)$	p	$g(p)$	p	$g(p)$
954854573	3	967607731	2	973215833	3
975831713	3	978949117	2	980766497	3
983879921	3	985918807	3	986608921	29
991136977	5	991752599	13	997137961	11
1003911991	3	1009775293	2	1012423549	6
1021000537	5	1023976897	7	1024153643	2
1037027287	3	1038812881	11	1044754639	3
1045125617	3	1047411427	3	1047753349	6

8.1.4 $\sim 1 \times 10^{18}$

p	$g(p)$	p	$g(p)$
951970612352230049	3	963284339889659609	3
967495386904694119	3	969751761517096213	2
983238274281901499	2	984647442475101409	23
989286107138674069	11	1002507954383424641	3
1006658951440146419	2	1020152326159075903	3
1034876265966119449	7	1042753851435034019	2
1043609016597371563	2	1045571042176595707	2
1048364250160580293	2	1049495624119026949	2

8.2 Pell's equation

$x^2 - ny^2 = 1$, where n is a positive nonsquare integer.
Let (x_0, y_0) be the smallest positive solution of the equation, then the k-th solution is:

$$\begin{pmatrix} x_k \\ y_k \end{pmatrix} = \begin{pmatrix} x_0 & ny_0 \\ y_0 & x_0 \end{pmatrix}^k \begin{pmatrix} x_0 \\ y_0 \end{pmatrix}$$

Some smallest solutions to Pell's equation:

n	2	3	5	6	7	8	10	11	12	13	14	15	17	18	19	20
x	3	2	9	5	8	3	19	10	7	649	15	4	33	17	170	9
y	2	1	4	2	3	1	6	3	2	180	4	1	8	4	39	2

8.3 Burnside's lemma and Polya's enumeration theorem

The Burnside's lemma says that

$$|X/G| = \frac{1}{|G|}\sum_{g\in G}|X^g|$$

where G is a group acting on X, X^g is the set of elements in X that are fixed by g, i.e. $X^g = \{x \in X : gx = x\}$.
The unweighted version of Pólya enumeration theorem says that

$$|Y^X/G| = \frac{1}{|G|}\sum_{g\in G} m^{c_g}$$

where $m = |X|$ is the number of colors, c_g is the number of the cycles of permutation g.

8.4 Lagrange's interpolation

For sample points $(x_0, y_0), \cdots, (x_k, y_k)$, define

$$l_j(x) = \prod_{0\le m\le k, m\ne j} \frac{x - x_m}{x_j - x_m}$$

then the Lagrange polynomial is

$$L(x) = \sum_{j=0}^{k} y_j l_j(x).$$

To use the script below, type two lines

```
x0 x1 x2 ... xn
y0 y1 y2 ... yn
```

the script will print the fractional coefficient of the polynomial in ascending exponent order.

```
#!/usr/bin/python2                                                          6dc9
from fractions import *                                                     4b2b
                                                                            427e
def polymul(a, b) :                                                         796b
    p = [0] * (len(a)+len(b)-1)                                             83e4
    for e1, c1 in enumerate(a) :                                            f697
        for e2, c2 in enumerate(b) :                                        156c
            p[e1+e2] += c1*c2                                               dfce
    return p                                                                5849
                                                                            427e
x, y = [map(Fraction, raw_input().split()) for _ in 0,0]                    f06d
n = len(x)                                                                  e80a
lj = [reduce(polymul, [[-x[m]/(x[j]-x[m]), 1/(x[j]-x[m])]                   a649
    for m in range(n) if m != j]) for j in range(n)]                        9dfa
print '␣'.join(map(str, map(sum, zip(*map(                                  3cae
    lambda a, b : [x*a for x in b], y, lj)))))                              7c0d
```