南京大学 ACM-ICPC 集训队代码模版库

OS: Linux-4.15.0-45-generic-x86_64-with-Ubuntu-18.04-bionic
python: 2.7.15rc1
2019-02-24 19:35:55.332606, build 0014

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

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

1.5 Template

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

2 Miscellaneous Algorithms

2.1 2-SAT

```
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;
1ce6         for (int i=0; i<G[x].size(); i++)
d942             if (!dfs(G[x][i])) 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);
6835         G[y^1].push_back(x);
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;
f4b9                 if (!dfs(i)){
3f03                     while (c > 0) mark[S[--c]] = false;
86c5                     if (!dfs(i+1)) return false;
95cf                 }
95cf             }
95cf         }
95cf         return true;
95cf     }
427e
5f0a     inline bool value(unsigned i){return mark[2*i+1];}
329b };
```

2.2 Knuth's optimization

```
5c83 int n;
d77c int dp[256][256], dc[256][256];
427e
b7ec template <typename T>
0bc7 void compute(T cost) {
0423   for (int i = 0; i <= n; i++) {
8f5e     dp[i][i] = 0;
9488     dc[i][i] = i;
95cf   }
be8e   rep (i, n) {
95b5     dp[i][i+1] = 0;
aa0f     dc[i][i+1] = i;
95cf   }
ec08   for (int len = 2; len <= n; len++) {
88b8     for (int i = 0; i + len <= n; i++) {
d3da       int j = i + len;
9824       int lbnd = dc[i][j-1], rbnd = dc[i+1][j];
a24a       dp[i][j] = INT_MAX / 2;
f933       int c = cost(i, j);
90d2       for (int k = lbnd; k <= rbnd; k++) {
9bd0         int res = dp[i][k] + dp[k][j] + c;
26b5         if (res < dp[i][j]) {
e6af           dp[i][j] = res;
9c88           dc[i][j] = k;
95cf         }
95cf       }
95cf     }
95cf   }
329b };
```

2.3 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.

Usage:

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

```
5194 constexpr int BLOCK_SZ = 300;
427e
3ec4 struct query { int l, r, id; };
d26a vector<query> queries;
427e
1e30 void add_query(int id, int l, int r) {
54c9   queries.push_back(query{l, r, id});
95cf }
427e
9f6b int l, r;
427e
427e // ----- functions to implement -----
62b4 inline void init();
50e1 inline void yield(int id);
b20d inline void enter(int o);
13af inline void leave(int o);
427e
37f0 void run() {
ab0b   if (queries.empty()) return;
8508   sort(range(queries), [](query lhs, query rhs) {
c7f8     int lb = lhs.l / BLOCK_SZ, rb = rhs.l / BLOCK_SZ;
03e7     if (lb != rb) return lb < rb;
0780     return lhs.r < rhs.r;
b251   });
6196   l = queries[0].l;
9644   r = queries[0].r;
07e2   init();
5bc9   for (query q : queries) {
7bc7     while (l > q.l) enter(l - 1), l--;
d646     while (r < q.r) enter(r + 1), r++;
13f0     while (l < q.l) leave(l), l++;
e1c6     while (r > q.r) leave(r), r--;
82f5     yield(q.id);
95cf   }
95cf }
```

3 String

3.1 Knuth-Morris-Pratt algorithm

```
2836 const int SIZE = 10005;
427e
d02b struct kmp_matcher {
2d81   char p[SIZE];
9847   int fail[SIZE];
57b7   int len;
427e
60cf   void construct(const char* needle) {
aaa1     len = strlen(p);
3a87     strcpy(p, needle);
3dd4     fail[0] = fail[1] = 0;
d8a8     for (int i = 1; i < len; i++) {
147f       int j = fail[i];
3c79       while (j && p[i] != p[j]) j = fail[j];
4643       fail[i + 1] = p[i] == p[j] ? j + 1 : 0;
95cf     }
95cf   }
427e
c464   inline void found(int pos) {
427e     // ! add codes for having found at pos
95cf   }
427e
2daf   void match(const char* haystack) {  // must be called after construct
700f     const char* t = haystack;
8482     int n = strlen(t);
8fd0     int j = 0;
be8e     rep(i, n) {
4e19       while (j && p[j] != t[i]) j = fail[j];
b5d5       if (p[j] == t[i]) j++;
f024       if (j == len) found(i - len + 1);
95cf     }
95cf   }
329b };
```

3.2 Manacher algorithm

```
81d4 struct Manacher {
cd09   int Len;
9255   vector<int> lc;
b301   string s;
427e
ec07   void work() {
c033     lc[1] = 1;
6bef     int k = 1;
427e
491f     for (int i = 2; i <= Len; i++) {
7957       int p = k + lc[k] - 1;
5e04       if (i <= p) {
24a1         lc[i] = min(lc[2 * k - i], p - i + 1);
8e2e       } else {
e0e5         lc[i] = 1;
95cf       }
74ff       while (s[i + lc[i]] == s[i - lc[i]]) lc[i]++;
2b9a       if (i + lc[i] > k + lc[k]) k = i;
95cf     }
95cf   }
427e
bfd5   void init(const char *tt) {
aaaf     int len = strlen(tt);
f701     s.resize(len * 2 + 10);
7045     lc.resize(len * 2 + 10);
8e13     s[0] = '*';
ae54     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];
0ef5         if (!u) {
9d58           tr[r][c] = tr[fail[r]][c];
b333           continue;
95cf         }
3e14         q.push(u);
b3ff         int v = fail[r];
d2ea         while (v && !tr[v][c]) v = fail[v];
c275         fail[u] = tr[v][c];
654c         last[u] = tag[fail[u]] ? fail[u] : last[fail[u]];
95cf       }
95cf     }
95cf   }
427e
7752   void found(int pos, int j) {
```

```
043e      if (j) {
427e        // ! add codes for having found word with tag[j]
4a96        found(pos, last[j]);
95cf      }
95cf    }
427e
9785    void find(const char* text) {  // must be called after construct()
80a4      int p = 0, c, len = strlen(text);
9c94      rep(i, len) {
b3db        c = id(text[i]);
f119        p = tr[p][c];
f08e        if (tag[p])
389b          found(i, p);
1e67        else if (last[p])
299e          found(i, last[p]);
95cf      }
95cf    }
329b  };
```

3.4 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

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

3.5 Trie

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

3.6 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
e491 LL crt(LL r[], LL p[], int n) {
84e6   LL q = 1, ret = 0;
00d9   rep (i, n) q *= p[i];
be8e   rep (i, n) {
98b4         LL m = q / p[i];
9f4f         LL d, x, y;
b082     exgcd(p[i], m, d, x, y);
3cd3     ret = (ret + y * m * r[i]) % q;
95cf   }
2e47   return (q + ret) % q;
95cf }
```

4.2 Matrix powermod

```
44b4 const int MAXN = 105;
92df const LL modular = 1000000007;
5c83 int n; // order of matrices
427e
8864 struct matrix{
3180     LL m[MAXN][MAXN];
427e
43c5     void operator *=(matrix& a){
e735         static LL t[MAXN][MAXN];
34d7         Rep (i, n){
4c11             Rep (j, n){
ee1e                 t[i][j] = 0;
c4a7                 Rep (k, n){
fcaf                     t[i][j] += (m[i][k] * a.m[k][j]) % modular;
199e                     t[i][j] %= modular;
95cf                 }
95cf             }
95cf         }
dad4         memcpy(m, t, sizeof(t));
95cf     }
329b };
427e
63d8 matrix r;
3ec2 void m_powmod(matrix& b, LL e){
83f0     memset(r.m, 0, sizeof(r.m));
a7c3     Rep(i, n)
de64         r.m[i][i] = 1;
3e90     while (e){
5a0e         if (e & 1) r *= b;
35c5         b *= b;
16fc         e >>= 1;
95cf     }
95cf }
```

4.3 Linear basis

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

4.4 Gauss elimination over finite field

```
b784 const LL p = 1000000007;
427e
2a2c LL powmod(LL b, LL e) {
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
37e1  for (int j = n - 1; j >= 0; j--) if (irow[j] != icol[j]) {
50dc    for (int k = 0; k < n; k++) swap(a[k][irow[j]], a[k][icol[j]]);
95cf  }
f27f  return det;
95cf }
```

4.5 Berlekamp-Massey algorithm

```
d790 vector<int> berlekamp(const vector<int>& a) {
4166     vector<int> p = {1}, r = {1};
baed     int dif = 1;
8bc9     rep (i, a.size()) {
3e58         int u = 0;
ac8e         rep (j, p.size())
a488             u = (u + 1ll * p[j] * a[i-j]) % mod;
eae9         if (u == 0) {
b14c             r.insert(r.begin(), 0);
8e2e         } else {
0c78             auto op = p;
02f6             p.resize(max(p.size(), r.size() + 1));
786b             int idif = inv(dif);
9b57             rep (j, r.size())
793c                 p[j+1] =
1836                     (p[j+1] - 1ll * r[j] * idif % mod * u % mod + mod) % mod;
644c             dif = u;
bc58             r = op;
95cf         }
95cf     }
e149     return p;
95cf }
```

4.6 Fast Walsh-Hadamard transform

```
061e void fwt(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, 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 }
```

4.7 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
b941     void dft(cplx* a, cplx* w){
a215         rep (i, N) if (i < rev[i]) swap(a[i], a[rev[i]]);
ac6e         for (int l = 2; l <= N; l *= 2){
2969             int m = l/2;
b3cf             for (cplx* p = a; p != a + N; p += l)
c24f                 rep (k, m){
fe06                     cplx t = w[N/l*k] * p[k+m];
ecbf                     p[k+m] = p[k] - t; p[k] += t;
95cf                 }
95cf         }
95cf     }
427e
617b     void fft(cplx* a){dft(a, omega);}
a123     void ifft(cplx* a){
3b2f         dft(a, oinv);
57fc         rep (i, N) a[i] /= N;
95cf     }
427e
bdc0     void conv(cplx* a, cplx* b){
6497         fft(a); fft(b);
12a5         rep (i, N) a[i] *= b[i];
f84e         ifft(a);
95cf     }
329b };
```

4.8 Number theoretic transform

```
4ab9 const int NMAX = 1<<21;
427e
427e // 998244353 = 7*17*2^23+1, G = 3
fb9a const int P = 1004535809, G = 3; // = 479*2^21+1
427e
87ab struct NTT{
c47c     int rev[NMAX];
0eda     LL omega[NMAX], oinv[NMAX];
81af     int g, g_inv; // g: g_n = G^((P-1)/n)
9827     int K, N;
427e
2a2c     LL powmod(LL b, LL e){
95a2         LL r = 1;
3e90         while (e){
6624             if (e&1) r = r * b % P;
489e             b = b * b % P;
16fc             e >>= 1;
95cf         }
547e         return r;
95cf     }
427e
f420     NTT(int k){
e209         K = k; N = 1 << k;
7652         g = powmod(G, (P-1)/N);
4b3a         g_inv = powmod(g, N-1);
e04f         omega[0] = oinv[0] = 1;
b393         rep (i, N){
7ba3             rev[i] = (rev[i>>1]>>1) | ((i&1)<<(K-1));
ad4f             if (i){
8d8b                 omega[i] = omega[i-1] * g % P;
9e14                 oinv[i] = oinv[i-1] * g_inv % P;
95cf             }
```

```
95cf        }
95cf    }
427e
9668    void _ntt(LL* a, LL* w){
a215        rep (i, N) if (i < rev[i]) swap(a[i], a[rev[i]]);
ac6e        for (int l = 2; l <= N; l *= 2){
2969            int m = l/2;
7a1d            for (LL* p = a; p != a + N; p += l)
c24f                rep (k, m){
0ad3                    LL t = w[N/l*k] * p[k+m] % P;
6209                    p[k+m] = (p[k] - t + P) % P;
fa1b                    p[k] = (p[k] + t) % P;
95cf                }
95cf        }
95cf    }
427e
92ea    void ntt(LL* a){_ntt(a, omega);}
5daf    void intt(LL* a){
1f2a        LL inv = powmod(N, P-2);
9910        _ntt(a, oinv);
a873        rep (i, N) a[i] = a[i] * inv % P;
95cf    }
427e
3a5b    void conv(LL* a, LL* b){
ad16        ntt(a); ntt(b);
e49e        rep (i, N) a[i] = a[i] * b[i] % P;
5748        intt(a);
95cf    }
329b};
```

4.9 Sieve of Euler

```
cfc3const int MAXX = 1e7+5;
5861bool p[MAXX];
73aeint prime[MAXX], sz;
427e
9bc6void sieve(){
9628    p[0] = p[1] = 1;
1ec8    for (int i = 2; i < MAXX; i++){
bf28        if (!p[i]) prime[sz++] = i;
e82c        for (int j = 0; j < sz && i*prime[j] < MAXX; j++){
b6a9            p[i*prime[j]] = 1;
5f51            if (i % prime[j] == 0) break;
95cf        }
95cf    }
95cf}
```

4.10 Sieve of Euler (General)

```
b62enamespace sieve {
6589  constexpr int MAXN = 10000007;
e982  bool p[MAXN]; // true if not prime
6ae8  int prime[MAXN], sz;
cbf7  int pval[MAXN], pcnt[MAXN];
6030  int f[MAXN];
427e
76f6  void exec(int N = MAXN) {
9628    p[0] = p[1] = 1;
427e
8a8a    pval[1] = 1;
bdda    pcnt[1] = 0;
c6b9    f[1] = 1;
427e
a643    for (int i = 2; i < N; i++) {
01d6      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 }
```

4.11 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
7f99         if (x == 1 || x == n-1) goto next;
e257         rep (i, r) {
d7ff             x = mulmod(x, x, n);
8d2e             if (x == n-1) goto next;
95cf         }
438e         return false;
d490 next:;
95cf     }
3361     return true;
95cf }
```

4.12 Pollard's rho algorithm

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

5 Graph Theory

5.1 Strongly connected component

```
837c const int MAXV = 100005;
427e
2ea0 struct graph{
88e3     vector<int> adj[MAXV];
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]])
b46e                    adjc[scc[i]].push_back(scc[adj[i][j]]);
95cf            }
95cf    }
329b};
```

5.2 Vertex biconnected component

```
0f42const int MAXN = 100005;
2ea0struct graph {
33ae    int pre[MAXN], iscut[MAXN], bccno[MAXN], dfs_clock, bcc_cnt;
848f    vector<int> adj[MAXN], bcc[MAXN];
6b06    set<pair<int, int>> bcce[MAXN];
427e
76f7    stack<pair<int, int>> s;
427e
bfab    void add_edge(int u, int v) {
c71a        adj[u].push_back(v);
a717        adj[v].push_back(u);
95cf    }
427e
7d3cint dfs(int u, int fa) {
9fe6    int lowu = pre[u] = ++dfs_clock;
ec14    int child = 0;
18f6    for (int v : adj[u]) {
173e        if (!pre[v]) {
e7f8            s.push({u, v});
fdcf            child++;
f851            int lowv = dfs(v, u);
189c            lowu = min(lowu, lowv);
b687            if (lowv >= pre[u]) {
6323                iscut[u] = 1;
57eb                bcc[bcc_cnt].clear();
90b8                bcce[bcc_cnt].clear();
a147                while (1) {
a6a3                    int xu, xv;
a0c3                    tie(xu, xv) = s.top(); s.pop();
0ef5                    bcce[bcc_cnt].insert({min(xu, xv), max(xu, xv)});
3db2                    if (bccno[xu] != bcc_cnt) {
e0db                        bcc[bcc_cnt].push_back(xu);
d27f                        bccno[xu] = bcc_cnt;
95cf                    }
f357                    if (bccno[xv] != bcc_cnt) {
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
17bevoid 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.

Usage:

`tarjan(u, fa)`	Run Tarjan's algorithm on tree rooted at `fa`. Please call with identical `u` and `fa`.

```
9f60const int MAXN = 200005;
0b32vector<int> adj[MAXN];
18e4int dfn[MAXN], low[MAXN], idx;
d39dbool cut[MAXN];
427e
bfabvoid add_edge(int u, int v) {
c71a    adj[u].push_back(v);
a717    adj[v].push_back(u);
95cf}
427e
50aavoid tarjan(int u, int fa) {
9891    dfn[u] = low[u] = ++idx;
ec14    int child = 0;
18f6    for (int v : adj[u]) {
3c64        if (!dfn[v]) {
9636            tarjan(v, fa); low[u] = min(low[u], low[v]);
f368            if (low[v] >= dfn[u] && u != fa) cut[u] = true;
7923            child += u == fa;
95cf        }
769a        low[u] = min(low[u], dfn[v]);
95cf    }
7927    if (u == fa && child > 1) cut[u] = true;
95cf}
```

5.4 Minimum spanning arborescence (Chu-Liu)

All vertices are 1-based.

Usage:

`getans(n, root, edges)`	Compute the total size of MSA rooted at `root`.

Time Complexity: $O(|V||E|)$

```
bcf8struct edge {
54f1    int u, v;
309c    LL w;
329b};
427e
f5a4const int MAXN = 10005;
7124LL in[MAXN];
1c1dint pre[MAXN], vis[MAXN], id[MAXN];
427e
5a43LL getans(int n, int rt, vector<edge>& edges) {
f7ff    LL ans = 0;
8abb    int cnt = 0;
a147    while (1) {
641a        Rep (i, n) in[i] = LLONG_MAX, id[i] = vis[i] = 0;
0705        for (auto e : edges) {
073a            if (e.u != e.v and e.w < in[e.v]) {
c1df                pre[e.v] = e.u;
5fbc                in[e.v] = e.w;
95cf            }
95cf        }
3fdb        in[rt] = 0;
34d7        Rep (i, n) {
3c97            if (in[i] == LLONG_MAX) return -1;
cf57            ans += in[i];
a763            int u;
4b0e            for (u = i; u != rt && vis[u] != i && !id[u]; u = pre[u])
88a2                vis[u] = i;
4b22            if (u != rt && !id[u]) {
b66e                id[u] = ++cnt;
0443                for (int v = pre[u]; v != u; v = pre[v])
5c22                    id[v] = cnt;
95cf            }
95cf        }
91e9        if (!cnt) return ans;
5e22        Rep (i, n) if (!id[i]) id[i] = ++cnt;
7400        for (auto& e : edges) {
7750            LL laz = in[e.v];
97ae            e.u = id[e.u];
fae6            e.v = id[e.v];
bdd2            if (e.u != e.v) e.w -= laz;
```

```
95cf        }
6cc4        n = cnt; rt = id[rt]; cnt = 0;
95cf    }
95cf}
```

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, $O(\sqrt{V}E)$.

```
bcf8struct edge{
60e2    int from, to;
5e6d    LL cap, flow;
329b};
427e
e2cdconst int MAXN = 1005;
9062struct Dinic {
4dbf    int n, m, s, t;
9f0c    vector<edge> edges;
b891    vector<int> G[MAXN];
bbb6    bool vis[MAXN];
b40a    int d[MAXN];
ddec    int cur[MAXN];
427e
5973    void add_edge(int from, int to, LL cap) {
7b55        edges.push_back(edge{from, to, cap, 0});
1db7        edges.push_back(edge{to, from, 0, 0});
fe77        m = edges.size();
dff5        G[from].push_back(m-2);
8f2d        G[to].push_back(m-1);
95cf    }
427e
1836    bool bfs() {
3b73        memset(vis, 0, sizeof(vis));
93d2        queue<int> q;
5d13        q.push(s);
2cd2        vis[s] = 1;
721d        d[s] = 0;
cc78        while (!q.empty()) {
66ba            int x = q.front(); q.pop();
3b61            for (int i = 0; i < G[x].size(); i++) {
b510                edge& e = edges[G[x][i]];
bba9                if (!vis[e.to] && e.cap > e.flow) {
cd72                    vis[e.to] = 1;
cf26                    d[e.to] = d[x] + 1;
ca93                    q.push(e.to);
95cf                }
95cf            }
95cf        }
b23b        return vis[t];
95cf}
427e
9252LL dfs(int x, LL a) {
6904    if (x == t || a == 0) return a;
8bf9    LL flow = 0, f;
f515    for (int& i = cur[x]; i < G[x].size(); i++) {
b510        edge& e = edges[G[x][i]];
2374        if(d[x] + 1 == d[e.to] && (f = dfs(e.to, min(a, e.cap-e.flow))) > 0)
               {
1cce            e.flow += f;
e16d            edges[G[x][i]^1].flow -= f;
a74d            flow += f;
23e5            a -= f;
97ed            if(a == 0) break;
95cf        }
95cf    }
84fb    return flow;
95cf}
427e
5bf2LL max_flow(int s, int t) {
590d    this->s = s; this->t = t;
62e2    LL flow = 0;
ed58    while (bfs()) {
f326        memset(cur, 0, sizeof(cur));
fb3a        flow += dfs(s, LLONG_MAX);
95cf    }
84fb    return flow;
95cf}
427ec72evector<int> min_cut() { // call this after maxflow
1df9    vector<int> ans;
df9a    for (int i = 0; i < edges.size(); i++) {
56d8        edge& e = edges[i];
46a2        if(vis[e.from] && !vis[e.to] && e.cap > 0) ans.push_back(i);
```

```
        }
        return ans;
    }
};
```

5.6 Maximum cardinality bipartite matching (Hungarian)

```
#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.

```
const int MAXN = 1024;
struct Blossom {

    vector<int> adj[MAXN];
    queue<int> q;
    int n;
    int label[MAXN], mate[MAXN], save[MAXN], used[MAXN];

    void init(int nv) {
        n = nv; for (auto& v : adj) v.clear();
        fill(range(label), 0); fill(range(mate), 0);
        fill(range(save), 0); fill(range(used), 0);
    }

    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] ++;
97ef            mate[i] = save[i];
95cf        }
95cf    }
427e
8bf8    void relabel(int x, int y) {
d101        Rep (i, n) used[i] = 0;
c4ea        traverse(x); traverse(y);
34d7        Rep (i, n) {
dee9            if (used[i] == 1 and label[i] < 0) {
1c22                label[i] = n + x + (y - 1) * n;
eb31                q.push(i);
95cf            }
95cf        }
95cf    }
427e
a0ce    int solve() {
34d7        Rep (i, n) {
a073            if (mate[i]) continue;
1fc0            Rep (j, n) label[j] = -1;
7676            label[i] = 0; q = queue<int>(); q.push(i);
1c7d            while (q.size()) {
66ba                int x = q.front(); q.pop();
b98c                for (int y : adj[x]) {
c07f                    if (mate[y] == 0 and i != y) {
7f36                        mate[y] = x; rematch(x, y); q = queue<int>(); break;
95cf                    }
d315                    if (label[y] >= 0) { relabel(x, y); continue; }
58ec                    if (label[mate[y]] < 0) {
c9c4                        label[mate[y]] = x; q.push(mate[y]);
95cf                    }
95cf                }
95cf            }
95cf        }
8abb        int cnt = 0;
b52f        Rep (i, n) cnt += (mate[i] > i);
6808        return cnt;
95cf    }
329b};
```

5.8 Minimum cost maximum flow

```
bcf8struct edge{
60e2    int from, to;
d698    int cap, flow;
32cc    LL cost;
329b};
427e
cc3econst LL INF = LLONG_MAX / 2;
2aa8const int MAXN = 5005;
c6cbstruct 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
71a4     void augment(){
06f1         int u = t;
b19d         while (u != s){
db09             edges[p[u]].flow += a[t];
25a9             edges[p[u]^1].flow -= a[t];
e6c9             u = edges[p[u]].from;
95cf         }
95cf     }
427e
6e20 #ifdef GIVEN_FLOW
5972     bool min_cost(int s, int t, int f, LL& cost) {
590d         this->s = s; this->t = t;
21d4         int flow = 0;
23cb         cost = 0;
22dc         while (spfa()) {
bcdb             augment();
a671             if (flow + a[t] >= f){
b14d                 cost += (f - flow) * d[t]; flow = f;
3361                 return true;
8e2e             } else {
2a83                 flow += a[t]; cost += a[t] * d[t];
95cf             }
95cf         }
438e         return false;
95cf     }
a8cb #else
f9a9     int min_cost(int s, int t, LL& cost) {
590d         this->s = s; this->t = t;
21d4         int flow = 0;
23cb         cost = 0;
22dc         while (spfa()) {
bcdb             augment();
2a83             flow += a[t]; cost += a[t] * d[t];
95cf         }
84fb         return flow;
95cf     }
1937 #endif
329b };
```

5.9 Global minimum cut (Stoer-Wagner)

Usage:

`stoer(w)`	Compute the global minimum cut of the graph specified by the **symmetric** adjacenct matrix `w` (0-based). Return the capaity of the cut and the indices of one part of the cut.

Time Complexity: $O(|V|^3)$

```
f9d7 typedef vector<LL> VI;
045e typedef vector<VI> VVI;
427e
f012 pair<LL, VI> stoer(VVI &w) {
66f7     int n = w.size();
4d98     VI used(n), c, bestc;
329d     LL bestw = -1;
427e
cd21     for (int ph = n - 1; ph >= 0; ph--) {
ec6e         VI wt = w[0], added = used;
f20e         int prev, last = 0;
4b32         rep (i, ph) {
8bfc             prev = last;
0706             last = -1;
4942             for (int j = 1; j < n; j++)
c4b9                 if (!added[j] && (last == -1 || wt[j] > wt[last]))
887d                     last = j;
71bc             if (i == ph - 1) {
9cfa                 rep (j, n) w[prev][j] += w[last][j];
```

```
1f25                rep (j, n) w[j][prev] = w[prev][j];
5613                used[last] = true;
8e11                c.push_back(last);
bb8e                if (bestw == -1 || wt[last] < bestw) {
bab6                    bestc = c;
372e                    bestw = wt[last];
95cf                }
8e2e            } else {
caeb                rep (j, n) wt[j] += w[last][j];
8b92                added[last] = true;
95cf            }
95cf        }
95cf    }
038c    return {bestw, bestc};
95cf}
```

5.10 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.

```
0e34const int MAXN = 500005;
0b32vector<int> adj[MAXN];
fccbint id[MAXN], nid;
1356pair<int, int> st[MAXN << 1][33 - __builtin_clz(MAXN)];
427e
e16dvoid dfs(int u, int p, int d) {
0df2    st[id[u] = nid++][0] = {d, u};
18f6    for (int v : adj[u]) {
bd87        if (v == p) continue;
f58c        dfs(v, u, d + 1);
08ad        st[nid++][0] = {d, u};
95cf    }
95cf}
427e
3d1bvoid preprocess(int root) {
3269    nid = 0;
91e1    dfs(root, 0, 1);
5e98    int l = 31 - __builtin_clz(nid);
213b    rep (j, l) rep (i, 1+nid-(1<<j))
1131        st[i][j+1] = min(st[i][j], st[i+(1<<j)][j]);
95cf}
427e
0f0bint lca(int u, int v) {
cfc4    tie(u, v) = minmax(id[u], id[v]);
be9b    int k = 31 - __builtin_clz(v-u+1);
8ebc    return min(st[u][k], st[v-(1<<k)+1][k]).second;
95cf}
```

5.11 Heavy-light decomposition

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

```
0f42const int MAXN = 100005;
0b32vector<int> adj[MAXN];
42f2int sz[MAXN], top[MAXN], fa[MAXN], son[MAXN], depth[MAXN], id[MAXN];
427e
be5cvoid dfs1(int x, int dep, int par){
7489    depth[x] = dep;
2ee7    sz[x] = 1;
adb4    fa[x] = par;
b79d    int maxn = 0, s = 0;
c861    for (int c: adj[x]){
fe45        if (c == par) continue;
fd2f        dfs1(c, dep + 1, x);
b790        sz[x] += sz[c];
f0f1        if (sz[c] > maxn){
c749            maxn = sz[c];
fe19            s = c;
95cf        }
95cf    }
0e08    son[x] = s;
95cf}
427e
ba54int cid = 0;
3644void dfs2(int x, int t){
8d96    top[x] = t;
d314    id[x] = ++cid;
c4a1    if (son[x]) dfs2(son[x], t);
c861    for (int c: adj[x]){
9881        if (c == fa[x]) continue;
5518        if (c == son[x]) continue;
13f9        else dfs2(c, c);
```

```
95cf     }
95cf }
427e
0f04 void decomp(int root){
9fa4     dfs1(root, 1, 0);
1c88     dfs2(root, root);
95cf }
427e
2c98 void query(int u, int v){
03a1     while (top[u] != top[v]){
45ec         if (depth[top[u]] < depth[top[v]]) swap(u, v);
427e         // id[top[u]] to id[u]
005b         u = fa[top[u]];
95cf     }
6083     if (depth[u] > depth[v]) swap(u, v);
427e     // id[u] to id[v]
95cf }
```

5.12 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.

```
1fb6 vector<int> adj[100005];
88e0 int sz[100005], sum;
427e
f93d void getsz(int u, int p) {
5b36   sz[u] = 1; sum++;
18f6   for (int v : adj[u]) {
bd87     if (v == p) continue;
e3cb     getsz(v, u);
8449     sz[u] += sz[v];
95cf   }
95cf }
427e
67f9 int getcent(int u, int p) {
d51f   for (int v : adj[u])
76e4     if (v != p and sz[v] > sum / 2)
18e3       return getcent(v, u);
81b0   return u;
95cf }
427e
4662 void decompose(int u) {
618e   sum = 0; getsz(u, 0);
303c   u = getcent(u, 0); // update u to the centroid
427e
18f6   for (int v : adj[u]) {
427e     // get answer for subtree v
95cf   }
427e   // get answer for the whole tree
427e   // don't forget to count the centroid itself
427e
18f6   for (int v : adj[u]) { // divide and conquer
c375     adj[v].erase(find(range(adj[v]), u));
fa6b     decompose(v);
a717     adj[v].push_back(u); // restore deleted edge
95cf   }
95cf }
```

5.13 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`.

```
1fb6 vector<int> adj[100005];
901d int sz[100005], son[100005];
427e
5559 void decomp(int u, int p) {
50c0     sz[u] = 1;
18f6     for (int v : adj[u]) {
bd87         if (v == p) continue;
a851         decomp(v, u);
8449         sz[u] += sz[v];
```

```
d28c         if (sz[v] > sz[son[u]]) son[u] = v;
95cf     }
95cf }
427e
b7ec template <typename T>
62f5 void trav(T fn, int u, int p) {
4412     fn(u);
30b3     for (int v : adj[u]) if (v != p) trav(fn, v, u);
95cf }
427e
7467 #define for_light(v) for (int v : adj[u]) if (v != p and v != son[u])
33ff void work(int u, int p, bool keep) {
72a2     for_light(v) work(v, u, 0); // process light children
427e
427e     // process heavy child
427e     // current data structure contains info of heavy child
9866     if (son[u]) work(son[u], u, 1);
427e
18a9     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
     int N;                                                    d7af
     vector<LL> tr;                                            99ff
                                                               427e
     void init(int n) { // fill the array with 0               d34f
         tr.resize(N = n + 5);                                 1010
     }                                                         95cf
                                                               427e
     LL sum(int n) {                                           63d0
         LL ans = 0;                                           f7ff
         while (n) {                                           e290
             ans += tr[n];                                     0715
             n &= n - 1;                                       c0d4
         }                                                     95cf
         return ans;                                           4206
     }                                                         95cf
                                                               427e
     void add(int n, LL x){                                    f4bd
         while (n < N) {                                       ad20
             tr[n] += x;                                       6c81
             n += n & -n;                                      0af5
         }                                                     95cf
     }                                                         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) { // fill the array with 0               d34f
         tr.resize(N = n + 5);                                 1010
     }                                                         95cf
                                                               427e
     LL query(int n) {                                         38d4
         LL ans = 0;                                           f7ff
         while (n < N) {                                       ad20
             ans += tr[n];                                     0715
             n += n & -n;                                      0af5
         }                                                     95cf
         return ans;                                           4206
     }                                                         95cf
```

```
427e
f4bd      void add(int n, LL x) {
e290          while (n){
6c81              tr[n] += x;
c0d4              n &= n - 1;
95cf          }
95cf      }
329b  };
```

6.3 Segment tree

```
3942  LL p;
1ebb  const int MAXN = 4 * 100006;
451a  struct segtree {
27be    int l[MAXN], m[MAXN], r[MAXN];
4510    LL val[MAXN], tadd[MAXN], tmul[MAXN];
427e
ac35  #define lson (o<<1)
1294  #define rson (o<<1|1)
427e
1344    void pull(int o) {
bbe9      val[o] = (val[lson] + val[rson]) % p;
95cf    }
427e
e4bc    void push_add(int o, LL x) {
5dd6      val[o] = (val[o] + x * (r[o] - l[o])) % p;
6eff      tadd[o] = (tadd[o] + x) % p;
95cf    }
427e
d658    void push_mul(int o, LL x) {
b82c      val[o] = val[o] * x % p;
aa86      tadd[o] = tadd[o] * x % p;
649f      tmul[o] = tmul[o] * x % p;
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);
8e2e    } else {
c4b0      push(o);
d1ba      if (ll < m[o]) mul(lson, ll, rr, x);
67f3      if (m[o] < rr) mul(rson, ll, rr, x);
ba26      pull(o);
95cf    }
95cf  }
427e
0f62  LL query(int o, int ll, int rr) {
```

```
3c16     if (ll <= l[o] && r[o] <= rr) {
6dfe       return val[o];
8e2e     } else {
c4b0       push(o);
462a       if (rr <= m[o]) return query(lson, ll, rr);
5cca       if (ll >= m[o]) return query(rson, ll, rr);
bbf9       return query(lson, ll, rr) + query(rson, ll, rr);
95cf     }
95cf   }
4d99 } seg;
```

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.

```
9f60 const int MAXN = 200005;
a7c5 mt19937 gen(time(NULL));
9542 struct Treap {
6d61     int ch[MAXN][2];
3948     int sz[MAXN], key[MAXN], val[MAXN];
5d9a     int add[MAXN], rev[MAXN];
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) {
6303         if (t == 0) { x = y = 0; return; }
f26b         push(t);
3465         if (sz[ch[t][0]] < k) {
ffd8             x = t; Split(ch[t][1], k - sz[ch[t][0]] - 1, ch[t][1], y);
8e2e         } else {
8a23             y = t; Split(ch[t][0], k, x, ch[t][0]);
95cf         }
89e3         if (x) pull(x); if (y) pull(y);
95cf     }
```

```
b1f4 } treap;
427e
24b6 int root;
427e
d34f void init(int n) {
34d7     Rep (i, n) {
7681         int x; scanf("%d", &x);
0ed8         treap.Init(i, x);
bcc8         root = (i == 1) ? 1 : treap.Merge(root, i);
95cf     }
95cf }
427e
d030 void work(int op, int l, int r) {
6639     int tl, tm, tr;
b6c4     treap.Split(root, l, tl, tm);
8de3     treap.Split(tm, r - l, tm, tr);
3658     if (op == 1) {
c039         int x; scanf("%d", &x); treap.Add(tm, x);
1dcb     } else if (op == 2) {
ae78         treap.Reverse(tm);
581d     } else if (op == 3) {
e092         printf("%lld␣%d␣%d\n",
867f                treap.sum[tm], treap.minv[tm], treap.maxv[tm]);
95cf     }
6188     root = treap.Merge(treap.Merge(tl, tm), tr);
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
d87a     int Root(int x) {
f4f1         for (access(x); ch[x][0]; x = ch[x][0]) push(x);
0d77         splay(x); return x;
95cf     }
9e46     void Link(int u, int v) { chroot(u); fa[u] = v; }
7c10     void Cut(int u, int v) { split(u, v); fa[u] = ch[v][0] = 0; pull(v); }
0691     int Query(int u, int v) { split(u, v); return sum[v]; }
a999     void Update(int u, int x) { splay(u); val[u] = x; }
1f42     int LCA(int x, int y, int root) {
```

```
6cb2        chroot(root); access(x); splay(y);
02e5        while (fa[y]) splay(y = fa[y]);
c218        return y;
95cf    }
329b };
```

6.6 Balanced binary search tree from pb_ds

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

6.7 Persistent segment tree, range k-th query

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

```
int node::n, node::pos;
inline void* node::operator new(size_t size) {
  return nodes + (pos++);
}
```

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.

```
const int BLOCK = 800;
typedef vector<int> vi;

struct block {
    vi data;
    LL sum; int minv, maxv;
    int add; bool rev;

    block(vi&& vec) : data(move(vec)),
        sum(accumulate(range(data), 0ll)),
        minv(*min_element(range(data))),
        maxv(*max_element(range(data))),
        add(0), rev(0) { }

    void fix() {
        if (rev) reverse(range(data));          rev = 0;
        if (add) for (int& x : data) x += add;  add = 0;
    }

    void merge(block& another) {
        fix(); another.fix();
        vi temp(move(data));
        temp.insert(temp.end(), range(another.data));
        *this = block(move(temp));
    }

    block split(int pos) {
        fix();
        block result(vi(data.begin() + pos, data.end()));
        data.resize(pos); *this = block(move(data));
        return result;
    }
};

typedef list<block>::iterator lit;

struct blocklist {
    list<block> blk;

    void maintain() {
        lit it = blk.begin();
        while (it != blk.end() && next(it) != blk.end()) {
            lit it2 = it;
            while (next(it2) != blk.end() &&
                    it2->data.size() + next(it2)->data.size() <= BLOCK) {
                it2->merge(*next(it2));
                blk.erase(next(it2));
            }
            ++it;
        }
    }

    lit split(int pos) {
        for (lit it = blk.begin(); ; it++) {
            if (pos == 0) return it;
            while (it->data.size() > pos)
                blk.insert(next(it), it->split(pos));
            pos -= it->data.size();

        }
    }

    void Init(int *l, int *r) {
        for (int *cur = l; cur < r; cur += BLOCK)
            blk.emplace_back(vi(cur, min(cur + BLOCK, r)));
    }

    void Reverse(int l, int r) {
        lit it = split(l), it2 = split(r);
        reverse(it, it2);
        while (it != it2) {
```

```
            it->rev ^= 1;
            it++;
        }
        maintain();
    }

    void Add(int l, int r, int x) {
        lit it = split(l), it2 = split(r);
        while (it != it2) {
            it->sum += LL(x) * it->data.size();
            it->minv += x; it->maxv += x;
            it->add += x; it++;
        }
        maintain();
    }

    void Query(int l, int r) {
        lit it = split(l), it2 = split(r);
        LL sum = 0; int minv = INT_MAX, maxv = INT_MIN;
        while (it != it2) {
            sum += it->sum;
            minv = min(minv, it->minv);
            maxv = max(maxv, it->maxv);
            it++;
        }
        maintain();
        printf("%lld %d %d\n", sum, minv, maxv);
    }
} lst;
```

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.

```
constexpr int BLOCK = 800;
typedef vector<int> vi;
typedef shared_ptr<vi> pvi;
typedef shared_ptr<const vi> pcvi;

struct block {
    pcvi data;
    LL sum;

    // add information to maintain
    block(pcvi ptr) :
        data(ptr),
        sum(accumulate(ptr->begin(), ptr->end(), 0ll))
    { }

    void merge(const block& another) {
        pvi temp = make_shared<vi>(data->begin(), data->end());
        temp->insert(temp->end(), another.data->begin(), another.data->end());
        *this = block(temp);
    }

    block split(int pos) {
        block result(make_shared<vi>(data->begin() + pos, data->end()));
        *this = block(make_shared<vi>(data->begin(), data->begin() + pos));
        return result;
    }
};

typedef list<block>::iterator lit;

struct blocklist {
    list<block> blk;

    void maintain() {
        lit it = blk.begin();
        while (it != blk.end() and next(it) != blk.end()) {
            lit it2 = it;
```

```
0b03            while (next(it2) != blk.end() and
029f                    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;
d480            while (it->data->size() > pos) {
2099                blk.insert(next(it), it->split(pos));
95cf            }
a1c8            pos -= it->data->size();
95cf        }
95cf    }
427e
fd38    LL sum(int l, int r) { // traverse
48b4        lit it1 = split(l), it2 = split(r);
ac09        LL res = 0;
9f1d        while (it1 != it2) {
8284            res += it1->sum;
61fd            it1++;
95cf        }
b204        maintain();
244d        return res;
95cf    }
329b};
```

6.10 Sparse table, range extremum query

The array is 0-based and the range is closed.

```
db63const int MAXN = 100007;
b330int a[MAXN];
69aeint st[MAXN][32 - __builtin_clz(MAXN)];
427e
8041inline int ext(int x, int y){return x>y?x:y;} // ! max
427e
d34fvoid init(int n){
ce01    int l = 31 - __builtin_clz(n);
cf75    rep (i, n) st[i][0] = a[i];
b811    rep (j, l)
6937        rep (i, 1+n-(1<<j))
082a            st[i][j+1] = ext(st[i][j], st[i+(1<<j)][j]);
95cf}
427e
c863int rmq(int l, int r){
92f5    int k = 31 - __builtin_clz(r-l+1);
baa2    return ext(st[l][k], st[r-(1<<k)+1][k]);
95cf}
```

7 Geometrics

7.1 2D geometric template

```
302f#include <bits/stdc++.h>
421cusing namespace std;
427e
4553typedef int T;
c0aetypedef struct pt {
7a9d    T x, y;
ffaa    T operator , (pt a) { return x*a.x + y*a.y; } // inner product
3ec7    T operator * (pt a) { return x*a.y - y*a.x; } // outer product
221a    pt operator + (pt a) { return {x+a.x, y+a.y}; }
8b34    pt operator - (pt a) { return {x-a.x, y-a.y}; }
427e
368b    pt operator * (T k) { return {x*k, y*k}; }
90f4    pt operator - () { return {-x, -y};}
ba8c} vec;
427e
0ea6typedef pair<pt, pt> seg;
427e
8d6ebool ptOnSeg(pt& p, seg& s){
ce77    vec v1 = s.first - p, v2 = s.second - p;
de97    return (v1, v2) <= 0 && v1 * v2 == 0;
95cf}
427e
427e// 0 not on segment
427e// 1 on segment except vertices
427e// 2 on vertices
8421int ptOnSeg2(pt& p, seg& s){
```

```
ce77     vec v1 = s.first - p, v2 = s.second - p;
70ca     T ip = (v1, v2);
8b14     if (v1 * v2 != 0 || ip > 0) return 0;
0847     return (v1, v2) ? 1 : 2;
95cf }
427e
427e // if two orthogonal rectangles do not touch, return true
72bb inline bool nIntRectRect(seg a, seg b){
f9ac     return min(a.first.x, a.second.x) > max(b.first.x, b.second.x) ||
f486            min(a.first.y, a.second.y) > max(b.first.y, b.second.y) ||
39ce            min(b.first.x, b.second.x) > max(a.first.x, a.second.x) ||
80c7            min(b.first.y, b.second.y) > max(a.first.y, a.second.y);
95cf }
427e
427e // >0 in order
427e // <0 out of order
427e // =0 not standard
7538 inline double rotOrder(vec a, vec b, vec c){return double(a*b)*(b*c);}
427e
31ed inline bool intersect(seg a, seg b){
427e     // ! if (nIntRectRect(a, b)) return false; // if commented, assume that a
           and b are non-collinear
cb52     return rotOrder(b.first-a.first, a.second-a.first, b.second-a.first) >= 0 &&
059e            rotOrder(a.first-b.first, b.second-b.first, a.second-b.first) >= 0;
95cf }
427e
427e // 0 not insersect
427e // 1 standard intersection
427e // 2 vertex-line intersection
427e // 3 vertex-vertex intersection
427e // 4 collinear and have common point(s)
4d19 int intersect2(seg& a, seg& b){
5dc4     if (nIntRectRect(a, b)) return 0;
42c0     vec va = a.second - a.first, vb = b.second - b.first;
2096     double j1 = rotOrder(b.first-a.first, va, b.second-a.first),
72fe            j2 = rotOrder(a.first-b.first, vb, a.second-b.first);
5ac6     if (j1 < 0 || j2 < 0) return 0;
9400     if (j1 != 0 && j2 != 0) return 1;
83db     if (j1 == 0 && j2 == 0){
6b0c         if (va * vb == 0) return 4; else return 3;
fb17     } 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

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

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 polyadd(a, b) : return map(lambda x, y : (x or 0) + (y or 0), a, b)     bbbe
                                                                            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, reduce(polyadd,                                     46f9
    map(lambda a, b : [x * a for x in b], y, lj))))                         d754
```