南京大学 ACM-ICPC 集训队代码模版库

Linux-4.15.0-65-generic-x86_64-with-Ubuntu-18.04-bionic
XeTeX 3.14159265-2.6-0.99998 (TeX Live 2017/Debian)
CPython 2.7.15+
2019-10-21 19:38:55.515427, build 0053

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 UN-SAT.
`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 Matroid Intersection

Find the maximum cardinality common independent set of two matroids. Matroids are given by independence oracle.

Usage:

`MatroidOracle`	The independence oracle maintaining an independent set. **Note** that the default constructor must properly initialize inner state to an empty set.
`insert(x)`	Insert element labeled x to the independent set.
`test(x)`	Test whether the set is still independent if x is inserted.
`MatroidIntersection<MT1, MT2>(n)`	Construct the matroid intersection solver with n elements labeled from 0 and matroid oracles `MT1` and `MT2`.
`run()`	Run the algorithm and return the matroid intersection.

```
struct MatroidOracle {                                          0935
    MatroidOracle() { /* TODO */ }                              297b
    void insert(int x) { /* TODO */ }                           53e5
    bool test(int x) const { /* TODO */ }                       ff18
};                                                              329b
                                                                427e
const int MAXN = 8192;                                          a015
template <typename MT1, typename MT2>                           94cc
struct MatroidIntersection {                                    3288
    int n;                                                      5c83
    bool in[MAXN] = {}, t[MAXN], vis[MAXN];                     5550
    int pre[MAXN];                                              fe84
    vector<int> adj[MAXN];                                      0b32
    queue<int> q;                                               93d2
                                                                427e
    MatroidIntersection(int n) : n(n) { }                       c152
```

```
427e
2ed1   vector<int> getcur() {
995a     vector<int> ret;
a585     rep (i, n) if (in[i]) ret.push_back(i);
ee0f     return ret;
95cf   }
427e
ca2b   void enqueue(int x, int p) {
e5da     if (vis[x]) return;
f4a6     vis[x] = true; pre[x] = p; q.push(x);
ff59     if (t[x]) throw x;
329b   };
427e
9081   vector<int> run() {
1026     while (true) {
c40f       vector<int> cur = getcur();
6f47       fill(vis, vis + n, 0);
943b       rep (i, n) adj[i].clear();
0e02       MT2 mt2;
3e54       for (int i : cur) mt2.insert(i);
191d       rep (i, n) t[i] = mt2.test(i);
e167       vector<MT1> mt1s(cur.size());
46d2       vector<MT2> mt2s(cur.size());
660b       rep (i, cur.size()) rep (j, cur.size()) if (i != j) {
3cd7         mt1s[i].insert(cur[j]);
9680         mt2s[i].insert(cur[j]);
95cf       }
e8d7       rep (i, n) if (!in[i]) rep (j, cur.size()) {
3fe9         if (mt1s[j].test(i)) adj[cur[j]].push_back(i);
645e         if (mt2s[j].test(i)) adj[i].push_back(cur[j]);
95cf       }
cf76       q = {};
85eb       try {
2f4f         MT1 mt1;
2f34         for (int i : cur) mt1.insert(i);
4053         rep (i, n) if (mt1.test(i)) enqueue(i, -1);
1c7d         while (q.size()) {
c048           int u = q.front(); q.pop();
a697           for (int v : adj[u]) enqueue(v, u);
95cf         }
5a9a       } catch (int v) {
a8f3         while (v >= 0) { in[v] ^= 1; v = pre[v]; }
b333         continue;
95cf       }
```

```
      break;                                       6173
    };                                             329b
    return getcur();                               f2de
  }                                                95cf
};                                                 329b
```

3 String

3.1 Knuth-Morris-Pratt algorithm

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

```
  pair<int, int> maxpal(int l, int r) {          b194
    int center = l + r + 1;                      901a
    int rad = lc[center] / 2;                    ffb2
    int rmid = (l + r + 1) / 2;                  ab54
    int rl = rmid - rad, rr = rmid + rad - 1;    17e4
    if ((r ^ l) & 1) {                           3908
    } else rr++;                                 69f3
    return {max(l, rl), min(r, rr)};             69dc
  }                                              95cf
};                                               329b
```

3.3 Aho-corasick automaton

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

```
    }
  }

  void found(int pos, int j) {
    if (j) {
      // ! add codes for having found word with tag[j]
      found(pos, last[j]);
    }
  }

  void find(const char* text) {  // must be called after construct()
    int p = 0, c, len = strlen(text);
    rep(i, len) {
      c = id(text[i]);
      p = tr[p][c];
      if (tag[p])
        found(i, p);
      else if (last[p])
        found(i, last[p]);
    }
  }
};
```

3.4 Trie

```
const int MAXN = 12000;
const int CHARN = 26;

inline int id(char c) { return c - 'a'; }

struct Trie {
  int n;
  int tr[MAXN][CHARN];  // Trie tree, 0 denotes fail
  int tag[MAXN];

  Trie() {
    memset(tr[0], 0, sizeof(tr[0]));
    tag[0] = 0;
    n = 1;
  }

  // tag should not be 0
  void add(const char* s, int t) {
    int p = 0, c, len = strlen(s);
    rep(i, len) {
      c = id(s[i]);
      if (!tr[p][c]) {
        memset(tr[n], 0, sizeof(tr[n]));
        tag[n] = 0;
        tr[p][c] = n++;
      }
      p = tr[p][c];
    }
    tag[p] = t;
  }

  // returns 0 if not found
  // AC automaton does not need this function
  int search(const char* s) {
    int p = 0, c, len = strlen(s);
    rep(i, len) {
      c = id(s[i]);
      if (!tr[p][c]) return 0;
      p = tr[p][c];
    }
    return tag[p];
  }
};
```

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) {
    static int cnt[1000005];    // size > max(n, m)
    fill(cnt, cnt + m, 0);
```

```
    rep (i, n) cnt[x[y[i]]]++;
    partial_sum(cnt, cnt + m, cnt);
    for (int i = n - 1; i >= 0; i--) sa[--cnt[x[y[i]]]] = y[i];
}

void suffix_array(int s[], int sa[], int rk[], int n, int m) {
    static int y[1000005];  // size > n
    copy(s, s + n, rk);
    iota(y, y + n, 0);
    radix_sort(rk, y, sa, n, m);
    for (int j = 1, p = 0; j <= n; j <<= 1, m = p, p = 0) {
        for (int i = n - j; i < n; i++) y[p++] = i;
        rep (i, n) if (sa[i] >= j) y[p++] = sa[i] - j;
        radix_sort(rk, y, sa, n, m + 1);
        swap_ranges(rk, rk + n, y);
        rk[sa[0]] = p = 1;
        for (int i = 1; i < n; i++)
            rk[sa[i]] = ((y[sa[i]] == y[sa[i-1]] and y[sa[i]+j] == y[sa[i-1]+j])
              ? p : ++p);
        if (p == n) break;
    }
    rep (i, n) rk[sa[i]] = i;
}

void calc_height(int s[], int sa[], int rk[], int h[], int n) {
    int k = 0;
    h[0] = 0;
    rep (i, n) {
        k = max(k - 1, 0);
        if (rk[i]) while (s[i+k] == s[sa[rk[i]-1]+k]) ++k;
        h[rk[i]] = k;
    }
}
```

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 $[l, r)$.

```
const LL mod = 1006658951440146419, g = 967;
const int MAXN = 200005;
LL pg[MAXN];

inline LL mul(LL x, LL y) { return __int128_t(x) * y % mod; }

void init_hash() {   // must be called in `int main()`
    pg[0] = 1;
    for (int i = 1; i < MAXN; i++) pg[i] = mul(pg[i-1], g);
}

struct hasher {
    LL val[MAXN];

    void build(const char *str) {   // assume lower-case letter only
        for (int i = 0; str[i]; i++)
            val[i+1] = (mul(val[i], g) + str[i]) % mod;
    }

    LL operator() (int l, int r) { // [l, r)
        return (val[r] - mul(val[l], pg[r-l]) + mod) % mod;
    }
};
```

4 Math

4.1 Extended Euclidean algorithm and Chinese remainder theorem

```
void exgcd(LL a, LL b, LL &g, LL &x, LL &y) {
    if (!b) g = a, x = 1, y = 0;
    else {
        exgcd(b, a % b, g, y, x);
        y -= x * (a / b);
    }
}

LL crt(LL r[], LL p[], int n) {
    LL q = 1, ret = 0;
    rep (i, n) q *= p[i];
    rep (i, n) {
        LL m = q / p[i];
        LL d, x, y;
        exgcd(p[i], m, d, x, y);
```

```
3cd3         ret = (ret + y * m * r[i]) % q;
95cf     }
2e47     return (q + ret) % q;
95cf }
```

4.2 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 };
```

4.3 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
typedef vector<LL> VLL;                                         c130
typedef vector<VLL> VVLL;                                       42ac
                                                                427e
LL gauss(VVLL &a, VVLL &b) {                                    2c62
  const int n = a.size(), m = b[0].size();                      561b
  vector<int> irow(n), icol(n), ipiv(n);                        a25e
  LL det = 1;                                                   2976
                                                                427e
  rep (i, n) {                                                  be8e
    int pj = -1, pk = -1;                                       d2b5
    rep (j, n) if (!ipiv[j])                                    6b4a
      rep (k, n) if (!ipiv[k])                                  e582
        if (pj == -1 || a[j][k] > a[pj][pk]) {                  6112
          pj = j;                                               a905
          pk = k;                                               657b
        }                                                       95cf
    if (a[pj][pk] == 0) return 0;                               d480
    ipiv[pk]++;                                                 0305
    swap(a[pj], a[pk]);                                         8dad
    swap(b[pj], b[pk]);                                         aad8
    if (pj != pk) det = (p - det) % p;                          be4d
    irow[i] = pj;                                               d080
    icol[i] = pk;                                               f156
                                                                427e
    LL c = powmod(a[pk][pk], p - 2);                            4ecd
    det = det * a[pk][pk] % p;                                  865b
    a[pk][pk] = 1;                                              c36a
    rep (j, n) a[pk][j] = a[pk][j] * c % p;                     dd36
    rep (j, m) b[pk][j] = b[pk][j] * c % p;                     1b23
    rep (j, n) if (j != pk) {                                   f8f3
      c = a[j][pk];                                             e97f
      a[j][pk] = 0;                                             c449
      rep (k, n) a[j][k] = (a[j][k] + p - a[pk][k] * c % p) % p;    820b
      rep (k, m) b[j][k] = (b[j][k] + p - b[pk][k] * c % p) % p;    f039
    }                                                           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.

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

4.5 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 }
```

4.6 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
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];
```

```
                    p[k+m] = p[k] - t; p[k] += t;
                }
            }
        }

    void fft(cplx* a){dft(a, omega);}
    void ifft(cplx* a){
        dft(a, oinv);
        rep (i, N) a[i] /= N;
    }

    void conv(cplx* a, cplx* b){
        fft(a); fft(b);
        rep (i, N) a[i] *= b[i];
        ifft(a);
    }
};
```

4.7 Number theoretic transform

```
const int NMAX = 1<<21;

// 998244353 = 7*17*2^23+1, G = 3
const int P = 1004535809, G = 3; // = 479*2^21+1

struct NTT{
    int rev[NMAX];
    LL omega[NMAX], oinv[NMAX];
    int g, g_inv; // g: g_n = G^((P-1)/n)
    int K, N;

    LL powmod(LL b, LL e){
        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]) {
        prime[sz++] = i;
        for (LL j = i; j < N; j *= i) {
          int b = j / i;
          pval[j] = i * pval[b];
          pcnt[j] = pcnt[b] + 1;
          f[j] = ______; // f[j] = f(i^pcnt[j])
        }
      }
      for (int j = 0; i * prime[j] < N; j++) {
        int x = i * prime[j]; p[x] = 1;
        if (i % prime[j] == 0) {
          pval[x] = pval[i] * prime[j];
          pcnt[x] = pcnt[i] + 1;
        } else {
          pval[x] = prime[j];
          pcnt[x] = 1;
        }
        if (x != pval[x]) {
          f[x] = f[x / pval[x]] * f[pval[x]]
        }
        if (i % prime[j] == 0) break;
      }
    }
  }
}
```

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 $n < 2^{64}$.

```
bool test(LL n){
    if (n < 3) return n==2;
    // ! The array a[] should be modified if the range of x changes.
    const LL a[] = {2LL, 7LL, 61LL, LLONG_MAX};
    LL r = 0, d = n-1, x;
    while (~d & 1) d >>= 1, r++;
    for (int i=0; a[i] < n; i++){
        x = powmod(a[i], d, n); // ! powmod must use for 64bit mulmod
        if (x == 1 || x == n-1) goto next;
        rep (i, r) {
            x = mulmod(x, x, n);
            if (x == n-1) goto next;
        }
        return false;
next:;
    }
    return true;
```

```
}
```

4.11 Integer factorization (Pollard's rho)

```
ULL gcd(ULL a, ULL b) {return b ? gcd(b, a % b) : a;}

ULL PollardRho(ULL n){
    ULL c, x, y, d = n;
    if (~n&1) return 2;
    while (d == n){
        x = y = 2;
        d = 1;
        c = rand() % (n - 1) + 1;
        while (d == 1){
            x = (mulmod(x, x, n) + c) % n;
            y = (mulmod(y, y, n) + c) % n;
            y = (mulmod(y, y, n) + c) % n;
            d = gcd(x>y ? x-y : y-x, n);
        }
    }
    return d;
}
```

4.12 Adaptive Simpson's Method

The Simpson's formula has order 3 algebraic precision.

Usage:

`integrate(l, r, eps, est, fn)`	Integrate the function `fn` on interval $[l, r]$. `eps` is the estimated precision, while `est` is the current estimation, which can be set to arbitrary value initially.

```
template <typename T>
double simpson(double l, double r, T&& f) {
    double mid = (l + r) / 2;
    return (f(l) + 4 * f(mid) + f(r)) * (r - l) / 6.0;
}

template <typename T>
double integrate(double l, double r, double eps, double est, T&& f) {
    double mid = (l + r) / 2;
    double lv = simpson(l, mid, f), rv = simpson(mid, r, f);
    if (fabs(lv + rv - est) <= 15.0 * eps)
        return lv + rv + (lv + rv - est) / 15.0;
    return integrate(l, mid, eps, lv, f) + integrate(mid, r, eps, rv, f);
}
```

5 Graph Theory

5.1 Strongly connected component

```
const int MAXV = 100005;

struct graph{
    vector<int> adj[MAXV];
    stack<int> s;
    int V; // number of vertices
    int pre[MAXV], lnk[MAXV], scc[MAXV];
    int time, sccn;

    void add_edge(int u, int v){
        adj[u].push_back(v);
    }

    void dfs(int u){
        pre[u] = lnk[u] = ++time;
        s.push(u);
        for (int v : adj[u]){
            if (!pre[v]){
                dfs(v);
                lnk[u] = min(lnk[u], lnk[v]);
            } else if (!scc[v]){
                lnk[u] = min(lnk[u], pre[v]);
            }
        }
        if (lnk[u] == pre[u]){
            sccn++;
            int x;
            do {
                x = s.top(); s.pop();
                scc[x] = sccn;
            } while (x != u);
        }
```

```
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

```
0f42 const int MAXN = 100005;
2ea0 struct 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
7d3c     int 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});
                     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
                                 bcc[bcc_cnt].push_back(xv);                   752b
                                 bccno[xv] = bcc_cnt;                          57c9
                             }                                                 95cf
                             if (xu == u && xv == v) break;                    7096
                         }                                                     95cf
                         bcc_cnt++;                                            03f5
                     }                                                         95cf
                 } else if (pre[v] < pre[u] && v != fa) {                      7470
                     s.push({u, v});                                           e7f8
                     lowu = min(lowu, pre[v]);                                 f115
                 }                                                             95cf
             }                                                                 95cf
             if (fa < 0 && child == 1) iscut[u] = 0;                           e104
             return lowu;                                                      1160
         }                                                                     95cf
                                                                               427e
         void find_bcc(int n) {                                                17be
             memset(pre, 0, sizeof pre);                                       8c2f
             memset(iscut, 0, sizeof iscut);                                   e2d2
             memset(bccno, -1, sizeof bccno);                                  40d3
             dfs_clock = bcc_cnt = 0;                                          fae2
             rep (i, n) if (!pre[i]) dfs(i, -1);                               5c63
         }                                                                     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
50aa void 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, 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: $O(|E| \log^2 |V|)$

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

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

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

```
struct edge{
    int from, to;
    LL cap, flow;
};

const int MAXN = 1005;
struct Dinic {
    int n, m, s, t;
    vector<edge> edges;
    vector<int> G[MAXN];
    bool vis[MAXN];
    int d[MAXN];
    int cur[MAXN];

    void add_edge(int from, int to, LL cap) {
        edges.push_back(edge{from, to, cap, 0});
        edges.push_back(edge{to, from, 0, 0});
        m = edges.size();
        G[from].push_back(m-2);
        G[to].push_back(m-1);
    }

    bool bfs() {
        memset(vis, 0, sizeof(vis));
        queue<int> q;
        q.push(s);
        vis[s] = 1;
        d[s] = 0;
        while (!q.empty()) {
            int x = q.front(); q.pop();
            for (int i = 0; i < G[x].size(); i++) {
                edge& e = edges[G[x][i]];
                if (!vis[e.to] && e.cap > e.flow) {
                    vis[e.to] = 1;
                    d[e.to] = d[x] + 1;
                    q.push(e.to);
                }
            }
        }
        return vis[t];
    }

    LL dfs(int x, LL a) {
        if (x == t || a == 0) return a;
        LL flow = 0, f;
        for (int& i = cur[x]; i < G[x].size(); i++) {
            edge& e = edges[G[x][i]];
            if(d[x] + 1 == d[e.to] && (f = dfs(e.to, min(a, e.cap-e.flow))) > 0)
              {
                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)

```
#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); }

```

```
    void rematch(int x, int y) {
        int m = mate[x]; mate[x] = y;
        if (mate[m] == x) {
            if (label[x] <= n) {
                mate[m] = label[x]; rematch(label[x], m);
            } else {
                int a = 1 + (label[x] - n - 1) / n;
                int b = 1 + (label[x] - n - 1) % n;
                rematch(a, b); rematch(b, a);
            }
        }
    }

    void traverse(int x) {
        Rep (i, n) save[i] = mate[i];
        rematch(x, x);
        Rep (i, n) {
            if (mate[i] != save[i]) used[i] ++;
            mate[i] = save[i];
        }
    }

    void relabel(int x, int y) {
        Rep (i, n) used[i] = 0;
        traverse(x); traverse(y);
        Rep (i, n) {
            if (used[i] == 1 and label[i] < 0) {
                label[i] = n + x + (y - 1) * n;
                q.push(i);
            }
        }
    }

    int solve() {
        Rep (i, n) {
            if (mate[i]) continue;
            Rep (j, n) label[j] = -1;
            label[i] = 0; q = queue<int>(); q.push(i);
            while (q.size()) {
                int x = q.front(); q.pop();
                for (int y : adj[x]) {
                    if (mate[y] == 0 and i != y) {
                        mate[y] = x; rematch(x, y); q = queue<int>(); break;
                    }
                    if (label[y] >= 0) { relabel(x, y); continue; }
                    if (label[mate[y]] < 0) {
                        label[mate[y]] = x; q.push(mate[y]);
                    }
                }
            }
        }
        int cnt = 0;
        Rep (i, n) cnt += (mate[i] > i);
        return cnt;
    }
};
```

5.8 Minimum cost maximum flow

```
struct edge{
    int from, to;
    int cap, flow;
    LL cost;
};

const LL INF = LLONG_MAX / 2;
const int MAXN = 5005;
struct MCMF {
    int s, t, n, m;
    vector<edge> edges;
    vector<int> G[MAXN];
    bool inq[MAXN]; // queue
    LL d[MAXN];     // distance
    int p[MAXN];    // previous
    int a[MAXN];    // improvement

    void add_edge(int from, int to, int cap, LL cost) {
        edges.push_back(edge{from, to, cap, 0, cost});
        edges.push_back(edge{to, from, 0, 0, -cost});
        m = edges.size();
        G[from].push_back(m-2);
        G[to].push_back(m-1);
    }

    bool spfa(){
        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     }
```

```
#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;                                      0e34
vector<int> adj[MAXN];                                        0b32
int id[MAXN], nid;                                            fccb
pair<int, int> st[MAXN << 1][33 - __builtin_clz(MAXN)];       1356
                                                              427e
void dfs(int u, int p, int d) {                               e16d
    st[id[u] = nid++][0] = {d, u};                            0df2
    for (int v : adj[u]) {                                    18f6
        if (v == p) continue;                                 bd87
        dfs(v, u, d + 1);                                     f58c
        st[nid++][0] = {d, u};                                08ad
    }                                                         95cf
}                                                             95cf
                                                              427e
void preprocess(int root) {                                   3d1b
    nid = 0;                                                  3269
    dfs(root, 0, 1);                                          91e1
    int l = 31 - __builtin_clz(nid);                          5e98
    rep (j, l) rep (i, 1+nid-(1<<j))                          213b
        st[i][j+1] = min(st[i][j], st[i+(1<<j)][j]);          1131
}                                                             95cf
                                                              427e
```

```
0f0b int 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 }
```

5.10 Heavy-light decomposition

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

```
0f42 const int MAXN = 100005;
0b32 vector<int> adj[MAXN];
42f2 int sz[MAXN], top[MAXN], fa[MAXN], son[MAXN], depth[MAXN], id[MAXN];
427e
be5c void 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
ba54 int cid = 0;
3644 void 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
void decomp(int root){                                            0f04
    dfs1(root, 1, 0);                                             9fa4
    dfs2(root, root);                                             1c88
}                                                                 95cf
                                                                  427e
void query(int u, int v){                                         2c98
    while (top[u] != top[v]){                                     03a1
        if (depth[top[u]] < depth[top[v]]) swap(u, v);            45ec
        // id[top[u]] to id[u]                                    427e
        u = fa[top[u]];                                           005b
    }                                                             95cf
    if (depth[u] > depth[v]) swap(u, v);                          6083
    // id[u] to id[v]                                             427e
}                                                                 95cf
```

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.

```
vector<int> adj[100005];                                          1fb6
int sz[100005], sum;                                              88e0
                                                                  427e
void getsz(int u, int p) {                                        f93d
  sz[u] = 1; sum++;                                               5b36
  for (int v : adj[u]) {                                          18f6
    if (v == p) continue;                                         bd87
    getsz(v, u);                                                  e3cb
    sz[u] += sz[v];                                               8449
  }                                                               95cf
}                                                                 95cf
                                                                  427e
int getcent(int u, int p) {                                       67f9
  for (int v : adj[u])                                            d51f
    if (v != p and sz[v] > sum / 2)                               76e4
      return getcent(v, u);                                       18e3
  return u;                                                       81b0
```

```
}

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`.

```
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 */ };
    auto enter = [] (int c) { /* add vertex c */ };
    auto leave = [] (int c) { /* remove vertex c*/ };

    for_light(v) {
        trav(merge, v, u);
        trav(enter, v, u);
    }

    // count answer for root and add it
    // Warning: special check may apply to root!
    merge(u);
    enter(u);

    // leave current tree
    if (!keep) trav(leave, u, p);
}
```

6 Data Structures

6.1 Fenwick tree (point update range query)

```
struct bit_purq { // point update, range query
    int N;
    vector<LL> tr;
```

```
427e 
2d99     void init(int n) { tr.assign(N = n + 5, 0); }
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)

```
3d03 struct bit_rupq{ // range update, point query
d7af     int N;
99ff     vector<LL> tr;
427e 
2d99     void init(int n) { tr.assign(N = n + 5, 0);}
427e 
38d4     LL query(int n) {
f7ff         LL ans = 0;
3667         while (n < N) { ans += tr[n]; n += n & -n; }
4206         return ans;
95cf     }
427e 
f4bd     void add(int n, LL x) {
0a2b         while (n) { tr[n] += x; n &= n - 1; }
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 
#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
  void push(int o) { b149
    if (l[o] == m[o]) return; 3159
    if (tmul[o] != 1) { 0a90
      push_mul(lson, tmul[o]); 0f4a
      push_mul(rson, tmul[o]); 045e
      tmul[o] = 1; ac0a
    } 95cf
    if (tadd[o]) { 1b82
      push_add(lson, tadd[o]); 9547
      push_add(rson, tadd[o]); 0e73
      tadd[o] = 0; 6234
    } 95cf
  } 95cf
 427e
  void build(int o, int ll, int rr) { 471c
    int mm = (ll + rr) / 2; 0e87
    l[o] = ll; r[o] = rr; m[o] = mm; 9d27
    tmul[o] = 1; ac0a
    if (ll == mm) { 5c92
      scanf("%lld", val + o); 001f
      val[o] %= p; e5b6
    } else { 8e2e
      build(lson, ll, mm); 7293
      build(rson, mm, rr); 5e67
      pull(o); ba26
    } 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 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;            c7f8
        if (lb != rb) return lb < rb;                                03e7
        return lhs.r < rhs.r;                                        0780
    });                                                              b251
    l = queries[0].l;                                                6196
    r = queries[0].r;                                                9644
    for (int i = l; i <= r; i++) enter(i);                           38e6
    for (query q : queries) {                                        5bc9
        while (l > q.l) enter(--l);                                  f422
        while (r < q.r) enter(++r);                                  39fb
        while (l < q.l) leave(l++);                                  46b3
        while (r > q.r) leave(r--);                                  6234
        yield(q.id);                                                 82f5
    }                                                                95cf
}                                                                    95cf
```

6.5 Mo's algorithm on tree

Numbers of vertices are 1-based. Implement `deal(int u)` and `query::yield()`.

```
const int MAXN = 200005, BLOCK = 300;
int n, m;
vector<int> adj[MAXN];
int en[MAXN], edx;
int dep[MAXN], fa[MAXN];
bool in[MAXN];

inline void deal(int u) {
    if (in[u] ^= 1) {
        // enter
    } else {
        // leave
    }
}

void moveto(int a, int b) {
    if (a == b) return;
    int cross = in[b] ? b : 0;
    auto moveup = [&] (int &x) {
        if (!cross) {
            if (in[x] and !in[fa[x]]) cross = x;
            else if (in[fa[x]] and !in[x]) cross = fa[x];
        }
        deal(x); x = fa[x];
    };
    while (dep[a] > dep[b]) moveup(a);
    while (dep[b] > dep[a]) moveup(b);
    while (a != b) moveup(a), moveup(b);
    deal(a); if (cross) deal(cross);
}

void dfs(int u, int p) {
    en[u] = edx++; fa[u] = p;
    for (int v : adj[u]) if (v != p) {
        dep[v] = dep[u] + 1;
        dfs(v, u); edx++;
    }
}

struct query {
    int l, r, id;
    void yield() { /* TODO */}
};
vector<query> qs;

void run() {
    dfs(1, 0);

    sort(range(qs), [] (query lhs, query rhs) {
        int u0 = en[lhs.l], v0 = en[rhs.l];
        int bl = u0 / BLOCK, br = v0 / BLOCK;
        if (bl != br) return bl < br;
        int u1 = en[lhs.r], v1 = en[rhs.r];
        return bl & 1 ? u1 < v1 : u1 > v1;
    });

    int l = 1, r = 1; deal(1);
    for (auto& q : qs) {
        moveto(l, q.l); l = q.l;
        moveto(r, q.r); r = q.r;
        q.yield();
    }
}
```

6.6 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;
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
    void Split(int t, int k, int &x, int &y) {                         dc7e
        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
```

6.7 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) {
             int z = x;                                                                    1548
             for (int y = 0; x; x = fa[y = x]) { splay(x); ch[x][1] = y; pull(x); }        8854
             splay(z);                                                                     7afd
         }                                                                                 95cf
         void chroot(int x) { access(x); reverse(x); }                                     a067
         void split(int x, int y) { chroot(x); access(y); }                                126d
                                                                                           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
```

6.8 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
```

6.9 Persistent segment tree, range k-th query

```
struct node {
  static int n, pos;

  int value;
  node *left, *right;

  void* operator new(size_t size);

  static node* Build(int l, int r) {
    node* a = new node;
    if (r > l + 1) {
      int mid = (l + r) / 2;
      a->left = Build(l, mid);
      a->right = Build(mid, r);
    } else {
      a->value = 0;
    }
    return a;
  }

  static node* init(int size) {
    n = size;
    pos = 0;
    return Build(0, n);
  }

  static int Query(node* lt, node *rt, int l, int r, int k) {
    if (r == l + 1) return l;
    int mid = (l + r) / 2;
    if (rt->left->value - lt->left->value < k) {
      k -= rt->left->value - lt->left->value;
      return Query(lt->right, rt->right, mid, r, k);
    } else {
      return Query(lt->left, rt->left, l, mid, k);
    }
  }

  static int query(node* lt, node *rt, int k) {
    return Query(lt, rt, 0, n, k);
  }

  node *Inc(int l, int r, int pos) const {
    node* a = new node(*this);
    if (r > l + 1) {
      int mid = (l + r) / 2;
      if (pos < mid)
        a->left = left->Inc(l, mid, pos);
      else
        a->right = right->Inc(mid, r, pos);
    }
    a->value++;
    return a;
  }

  node *inc(int index) {
    return Inc(0, n, index);
  }
} nodes[8000000];

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

6.10 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)),
```

```
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++) {
```

```
            if (pos == 0) return it;                           5502
            while (it->data.size() > pos)                      8e85
                blk.insert(next(it), it->split(pos));          2099
            pos -= it->data.size();                            a5a1
                                                               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     }
958e } lst;
```

6.11 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) {
         block result(make_shared<vi>(data->begin() + pos, data->end())); dac1
         *this = block(make_shared<vi>(data->begin(), data->begin() + pos)); 01db
         return result; 56b0
     } 95cf
}; 329b
 427e
typedef list<block>::iterator lit; 2a18
 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 };
```

6.12 Sparse table, range minimum query

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

```
db63 const int MAXN = 100007;
cefd int a[MAXN], st[MAXN][30];
427e
d34f void init(int n){
c73d     int l = log2(n);
cf75     rep (i, n) st[i][0] = a[i];
426b     rep (j, l) rep (i, 1+n-(1<<j))
1131         st[i][j+1] = min(st[i][j], st[i+(1<<j)][j]);
95cf }
427e
c863 int rmq(int l, int r){
f089     int k = log2(r - l);
6117     return min(st[l][k], st[r-(1<<k)][k]);
95cf }
```

7 Geometrics

7.1 2D geometric template

```
302f #include <bits/stdc++.h>
421c using namespace std;
427e
4553 typedef int T;
c0ae typedef 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
0ea6 typedef pair<pt, pt> seg;
```

```
                                                                                     427e
bool ptOnSeg(pt& p, seg& s){                                                         8d6e
    vec v1 = s.first - p, v2 = s.second - p;                                         ce77
    return (v1, v2) <= 0 && v1 * v2 == 0;                                            de97
}                                                                                    95cf
                                                                                     427e
// 0 not on segment                                                                  427e
// 1 on segment except vertices                                                      427e
// 2 on vertices                                                                     427e
int ptOnSeg2(pt& p, seg& s){                                                         8421
    vec v1 = s.first - p, v2 = s.second - p;                                         ce77
    T ip = (v1, v2);                                                                 70ca
    if (v1 * v2 != 0 || ip > 0) return 0;                                            8b14
    return (v1, v2) ? 1 : 2;                                                         0847
}                                                                                    95cf
                                                                                     427e
// if two orthogonal rectangles do not touch, return true                            427e
inline bool nIntRectRect(seg a, seg b){                                              72bb
    return min(a.first.x, a.second.x) > max(b.first.x, b.second.x) ||                f9ac
           min(a.first.y, a.second.y) > max(b.first.y, b.second.y) ||                f486
           min(b.first.x, b.second.x) > max(a.first.x, a.second.x) ||                39ce
           min(b.first.y, b.second.y) > max(a.first.y, a.second.y);                  80c7
}                                                                                    95cf
                                                                                     427e
// >0 in order                                                                       427e
// <0 out of order                                                                   427e
// =0 not standard                                                                   427e
inline double rotOrder(vec a, vec b, vec c){return double(a*b)*(b*c);}               7538
                                                                                     427e
inline bool intersect(seg a, seg b){                                                 31ed
    // ! if (nIntRectRect(a, b)) return false; // if commented, assume that a        427e
      and b are non-collinear
    return rotOrder(b.first-a.first, a.second-a.first, b.second-a.first) >= 0 &&     cb52
           rotOrder(a.first-b.first, b.second-b.first, a.second-b.first) >= 0;       059e
}                                                                                    95cf
                                                                                     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)                                              427e
int intersect2(seg& a, seg& b){                                                      4d19
    if (nIntRectRect(a, b)) return 0;                                                5dc4
    vec va = a.second - a.first, vb = b.second - b.first;                            42c0
```

```
    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;
}

template <typename Tp = T>
inline pt getIntersection(pt P, vec v, pt Q, vec w){
    static_assert(is_same<Tp, double>::value, "must␣be␣double!");
    return P + v * (w*(P-Q)/(v*w));
}

// -1 outside the polygon
// 0  on the border of the polygon
// 1  inside the polygon
int ptOnPoly(pt p, pt* poly, int n){
    int wn = 0;
    for (int i = 0; i < n; i++) {

        T k, d1 = poly[i].y - p.y, d2 = poly[(i+1)%n].y - p.y;
        if (k = (poly[(i+1)%n] - poly[i])*(p - poly[i])){
            if (k > 0 && d1 <= 0 && d2 > 0) wn++;
            if (k < 0 && d2 <= 0 && d1 > 0) wn--;
        } else return 0;
    }
    return wn ? 1 : -1;
}

istream& operator >> (istream& lhs, pt& rhs){
    lhs >> rhs.x >> rhs.y;
    return lhs;
}

istream& operator >> (istream& lhs, seg& rhs){
    lhs >> rhs.first >> rhs.second;
    return lhs;
}
```

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 Supnick TSP

Given f and $x_1 \le x_2 \le \cdots \le x_n$, if f is Supnick, then

$$\sum_{i=1}^{n} f(x_{\pi(i)}, x_{\pi(i+1)})$$

1. is minimized when $\pi = (1, 3, 5, 7, \cdots, 8, 6, 4, 2)$.
2. is maximized when $\pi = (n, 2, n-2, 4, \cdots, 5, n-3, 3, n-1, 1)$.

8.5 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
```