南京大学 ACM-ICPC 集训队代码模版库

Linux-4.15.0-48-generic-x86_64-with-Ubuntu-18.04-bionic
XeTeX 3.14159265-2.6-0.99998 (TeX Live 2017/Debian)
CPython 2.7.15rc1
2019-05-14 14:35:43.449888, build 0048

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.

```
const int MAXN = 100005;
struct twoSAT {
    int n;
    vector<int> G[MAXN*2];
    bool mark[MAXN*2];
    int S[MAXN*2], c;

    void init(int n) {
        this->n = n;
        for (int i=0; i < n*2; i++) G[i].clear();
        memset(mark, 0, sizeof(mark));
    }

    bool dfs(int x) {
        if (mark[x^1]) return false;
        if (mark[x]) return true;
        mark[x] = true;
        S[c++] = x;
        for (int u : G[x]) if (!dfs(u)) return false;
        return true;
    }

    void add_clause(int x, bool xval, int y, bool yval) {
        x = x * 2 + xval;
        y = y * 2 + yval;
        G[x^1].push_back(y);
    }

    bool solve() {
        for (int i=0; i<n*2; i+=2) {
            if (!mark[i] && !mark[i+1]) {
                c = 0;
                if (!dfs(i)) {
                    while (c > 0) mark[S[--c]] = false;
                    if (!dfs(i+1)) return false;
                }
            }
        }
        return true;
    }

    bool operator[] (int x) { return mark[2*x+1]; }
};
```

2.2 Mo's algorithm

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

Usage:

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

```
constexpr int BLOCK_SZ = 300;

struct query { int l, r, id; };
vector<query> queries;

void add_query(int id, int l, int r) {
  queries.push_back(query{l, r, id});
}

int l, r;

// ----- functions to implement -----
inline void yield(int id);
inline void enter(int o);
inline void leave(int o);

void run() {
  if (queries.empty()) return;
  sort(range(queries), [](query lhs, query rhs) {
```

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

2.3 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.

```cpp
struct MatroidOracle {
    MatroidOracle() { /* TODO */ }
    void insert(int x) { /* TODO */ }
    bool test(int x) const { /* TODO */ }
};

const int MAXN = 8192;
template <typename MT1, typename MT2>
struct MatroidIntersection {
    int n;
    bool in[MAXN] = {}, t[MAXN], vis[MAXN];
    int pre[MAXN];
    vector<int> adj[MAXN];
    queue<int> q;

    MatroidIntersection(int n) : n(n) { }

    vector<int> getcur() {
        vector<int> ret;
        rep (i, n) if (in[i]) ret.push_back(i);
        return ret;
    }

    void enqueue(int x, int p) {
        if (vis[x]) return;
        vis[x] = true; pre[x] = p; q.push(x);
        if (t[x]) throw x;
    };

    vector<int> run() {
        while (true) {
            vector<int> cur = getcur();
            fill(vis, vis + n, 0);
            rep (i, n) adj[i].clear();
            MT2 mt2;
            for (int i : cur) mt2.insert(i);
            rep (i, n) t[i] = mt2.test(i);
            vector<MT1> mt1s(cur.size());
            vector<MT2> mt2s(cur.size());
            rep (i, cur.size()) rep (j, cur.size()) if (i != j) {
                mt1s[i].insert(cur[j]);
                mt2s[i].insert(cur[j]);
            }
            rep (i, n) if (!in[i]) rep (j, cur.size()) {
                if (mt1s[j].test(i)) adj[cur[j]].push_back(i);
                if (mt2s[j].test(i)) adj[i].push_back(cur[j]);
            }
            q = {};
            try {
                MT1 mt1;
                for (int i : cur) mt1.insert(i);
                rep (i, n) if (mt1.test(i)) enqueue(i, -1);
                while (q.size()) {
                    int u = q.front(); q.pop();
                    for (int v : adj[u]) enqueue(v, u);
                }
```

```
5a9a            } catch (int v) {
a8f3                while (v >= 0) { in[v] ^= 1; v = pre[v]; }
b333                continue;
95cf            }
6173            break;
329b        };
f2de        return getcur();
95cf    }
329b };
```

3 String

3.1 Knuth-Morris-Pratt algorithm

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

```
struct Manacher {                                         81d4
  int Len;                                                cd09
  vector<int> lc;                                         9255
  string s;                                               b301
                                                          427e
  void work() {                                           ec07
    lc[1] = 1;                                            c033
    int k = 1;                                            6bef
                                                          427e
    for (int i = 2; i <= Len; i++) {                      491f
      int p = k + lc[k] - 1;                              7957
      if (i <= p) {                                       5e04
        lc[i] = min(lc[2 * k - i], p - i + 1);            24a1
      } else {                                            8e2e
        lc[i] = 1;                                        e0e5
      }                                                   95cf
      while (s[i + lc[i]] == s[i - lc[i]]) lc[i]++;       74ff
      if (i + lc[i] > k + lc[k]) k = i;                   2b9a
    }                                                     95cf
  }                                                       95cf
                                                          427e
  void init(const char *tt) {                             bfd5
    int len = strlen(tt);                                 aaaf
    s.resize(len * 2 + 10);                               f701
    lc.resize(len * 2 + 10);                              7045
    s[0] = '*';                                           8e13
    s[1] = '#';                                           ae54
    for (int i = 0; i < len; i++) {                       1321
      s[i * 2 + 2] = tt[i];                               e995
      s[i * 2 + 1] = '#';                                 69fd
    }                                                     95cf
    s[len * 2 + 1] = '#';                                 43fd
    s[len * 2 + 2] = '\0';                                75d1
```

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

3.3 Aho-corasick automaton

```
a1ad struct AC : Trie {
9143   int fail[MAXN];
daca   int last[MAXN];
427e
8690   void construct() {
93d2     queue<int> q;
a7a6     fail[0] = 0;
ce3c     rep(c, CHARN) {
b1c6       if (int u = tr[0][c]) {
a506         fail[u] = 0;
3e14         q.push(u);
f689         last[u] = 0;
95cf       }
95cf     }
cc78     while (!q.empty()) {
31f0       int r = q.front();
15dd       q.pop();
ce3c       rep(c, CHARN) {
ab59         int u = tr[r][c];
0ef5         if (!u) {
9d58           tr[r][c] = tr[fail[r]][c];
b333           continue;
95cf         }
3e14         q.push(u);
b3ff         int v = fail[r];
```

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

3.4 Trie

```
const int MAXN = 12000;                                                  e6f1
const int CHARN = 26;                                                    dd87
                                                                         427e
inline int id(char c) { return c - 'a'; }                                8ff5
                                                                         427e
struct Trie {                                                            a281
  int n;                                                                 5c83
  int tr[MAXN][CHARN];  // Trie tree, 0 denotes fail                     f4f5
  int tag[MAXN];                                                         35a5
                                                                         427e
  Trie() {                                                               4fee
    memset(tr[0], 0, sizeof(tr[0]));                                     3ccc
    tag[0] = 0;                                                          4d52
```

```
    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 $[l, r)$.

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

4 Math

4.1 Extended Euclidean algorithm and Chinese remainder theorem

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

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

4.2 Linear basis

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

4.3 Gauss elimination over finite field

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

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

typedef vector<LL> VLL;
typedef vector<VLL> VVLL;

LL gauss(VVLL &a, VVLL &b) {
  const int n = a.size(), m = b[0].size();
  vector<int> irow(n), icol(n), ipiv(n);
  LL det = 1;

  rep (i, n) {
    int pj = -1, pk = -1;
    rep (j, n) if (!ipiv[j])
      rep (k, n) if (!ipiv[k])
        if (pj == -1 || a[j][k] > a[pj][pk]) {
          pj = j;
          pk = k;
        }
    if (a[pj][pk] == 0) return 0;
    ipiv[pk]++;
    swap(a[pj], a[pk]);
    swap(b[pj], b[pk]);
    if (pj != pk) det = (p - det) % p;
    irow[i] = pj;
    icol[i] = pk;

    LL c = powmod(a[pk][pk], p - 2);
    det = det * a[pk][pk] % p;
    a[pk][pk] = 1;
    rep (j, n) a[pk][j] = a[pk][j] * c % p;
    rep (j, m) b[pk][j] = b[pk][j] * c % p;
    rep (j, n) if (j != pk) {
      c = a[j][pk];
      a[j][pk] = 0;
      rep (k, n) a[j][k] = (a[j][k] + p - a[pk][k] * c % p) % p;
      rep (k, m) b[j][k] = (b[j][k] + p - b[pk][k] * c % p) % p;
    }
  }

  for (int j = n - 1; j >= 0; j--) if (irow[j] != icol[j]) {
    for (int k = 0; k < n; k++) swap(a[k][irow[j]], a[k][icol[j]]);
  }
  return det;
}
```

4.4 Berlekamp-Massey algorithm

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

```
LL mod = 1000000007;
vector<LL> berlekamp(const vector<LL>& a) {
    vector<LL> p = {1}, r = {1};
    LL dif = 1;
    rep (i, a.size()) {
        LL u = 0;
        rep (j, p.size()) u = (u + p[j] * a[i-j]) % mod;
        if (u == 0) {
            r.insert(r.begin(), 0);
        } else {
            auto op = p;
            p.resize(max(p.size(), r.size() + 1));
            LL idif = powmod(dif, mod - 2);
            rep (j, r.size())
                p[j+1] = (p[j+1] - r[j] * idif % mod * u % mod + mod) % mod;
            dif = u; r = op;
        }
    }
    return p;
}
```

4.5 Fast Walsh-Hadamard transform

```
void fwt(int* a, int n){
    for (int d = 1; d < n; d <<= 1)
        for (int i = 0; i < n; i += d << 1)
            rep (j, d){
                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
    void dft(cplx* a, cplx* w){                            b941
        rep (i, N) if (i < rev[i]) swap(a[i], a[rev[i]]);  a215
        for (int l = 2; l <= N; l *= 2){                   ac6e
            int m = l/2;                                   2969
            for (cplx* p = a; p != a + N; p += l)          b3cf
                rep (k, m){                                c24f
                    cplx t = w[N/l*k] * p[k+m];            fe06
                    p[k+m] = p[k] - t; p[k] += t;          ecbf
                }                                          95cf
        }                                                  95cf
    }                                                      95cf
                                                           427e
    void fft(cplx* a){dft(a, omega);}                      617b
    void ifft(cplx* a){                                    a123
        dft(a, oinv);                                      3b2f
        rep (i, N) a[i] /= N;                              57fc
    }                                                      95cf
                                                           427e
    void conv(cplx* a, cplx* b){                           bdc0
        fft(a); fft(b);                                    6497
        rep (i, N) a[i] *= b[i];                           12a5
        ifft(a);                                           f84e
    }                                                      95cf
};                                                         329b
```

4.7 Number theoretic transform

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

```
95a2         LL r = 1;
3e90         while (e){
6624             if (e&1) r = r * b % P;
489e             b = b * b % P;
16fc             e >>= 1;
95cf         }
547e         return r;
95cf     }
427e 
f420     NTT(int k){
e209         K = k; N = 1 << k;
7652         g = powmod(G, (P-1)/N);
4b3a         g_inv = powmod(g, N-1);
e04f         omega[0] = oinv[0] = 1;
b393         rep (i, N){
7ba3             rev[i] = (rev[i>>1]>>1) | ((i&1)<<(K-1));
ad4f             if (i){
8d8b                 omega[i] = omega[i-1] * g % P;
9e14                 oinv[i] = oinv[i-1] * g_inv % P;
95cf             }
95cf         }
95cf     }
427e 
9668     void _ntt(LL* a, LL* w){
a215         rep (i, N) if (i < rev[i]) swap(a[i], a[rev[i]]);
ac6e         for (int l = 2; l <= N; l *= 2){
2969             int m = l/2;
7a1d             for (LL* p = a; p != a + N; p += l)
c24f                 rep (k, m){
0ad3                     LL t = w[N/l*k] * p[k+m] % P;
6209                     p[k+m] = (p[k] - t + P) % P;
fa1b                     p[k] = (p[k] + t) % P;
95cf                 }
95cf         }
95cf     }
427e 
92ea     void ntt(LL* a){_ntt(a, omega);}
5daf     void intt(LL* a){
1f2a         LL inv = powmod(N, P-2);
9910         _ntt(a, oinv);
a873         rep (i, N) a[i] = a[i] * inv % P;
95cf     }
427e 
3a5b     void conv(LL* a, LL* b){
```

```
        ntt(a); ntt(b);                          ad16
        rep (i, N) a[i] = a[i] * b[i] % P;       e49e
        intt(a);                                 5748
    }                                            95cf
};                                               329b
```

4.8 Sieve of Euler

```
const int MAXX = 1e7+5;                                        cfc3
bool p[MAXX];                                                  5861
int prime[MAXX], sz;                                           73ae
                                                               427e
void sieve(){                                                  9bc6
    p[0] = p[1] = 1;                                           9628
    for (int i = 2; i < MAXX; i++){                            1ec8
        if (!p[i]) prime[sz++] = i;                            bf28
        for (int j = 0; j < sz && i*prime[j] < MAXX; j++){     e82c
            p[i*prime[j]] = 1;                                 b6a9
            if (i % prime[j] == 0) break;                      5f51
        }                                                      95cf
    }                                                          95cf
}                                                              95cf
```

4.9 Sieve of Euler (General)

```
namespace sieve {                                  b62e
  constexpr int MAXN = 10000007;                   6589
  bool p[MAXN]; // true if not prime               e982
  int prime[MAXN], sz;                             6ae8
  int pval[MAXN], pcnt[MAXN];                      cbf7
  int f[MAXN];                                     6030
                                                   427e
  void exec(int N = MAXN) {                        76f6
    p[0] = p[1] = 1;                               9628
                                                   427e
    pval[1] = 1;                                   8a8a
    pcnt[1] = 0;                                   bdda
    f[1] = 1;                                      c6b9
                                                   427e
    for (int i = 2; i < N; i++) {                  a643
      if (!p[i]) {                                 01d6
```

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

4.10 Miller-Rabin primality test

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

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

```
f16f bool test(LL n){
59f2     if (n < 3) return n==2;
427e     // ! The array a[] should be modified if the range of x changes.
3f11     const LL a[] = {2LL, 7LL, 61LL, LLONG_MAX};
c320     LL r = 0, d = n-1, x;
f410     while (~d & 1) d >>= 1, r++;
2975     for (int i=0; a[i] < n; i++){
ece1         x = powmod(a[i], d, n); // ! powmod must use for 64bit mulmod
```

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

4.11 Integer factorization (Pollard's rho)

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

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>                                    b7ec
```

```
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);
        }
    }

    void find_scc(){
        time = sccn = 0;
        memset(scc, 0, sizeof scc);
        memset(pre, 0, sizeof pre);
        Rep (i, V){
            if (!pre[i]) dfs(i);
        }
    }

    vector<int> adjc[MAXV];
    void contract(){
        Rep (i, V)
            rep (j, adj[i].size()){
                if (scc[i] != scc[adj[i][j]])
                    adjc[scc[i]].push_back(scc[adj[i][j]]);
            }
    }
};
```

5.2 Vertex biconnected component

```
const int MAXN = 100005;
struct graph {
    int pre[MAXN], iscut[MAXN], bccno[MAXN], dfs_clock, bcc_cnt;
    vector<int> adj[MAXN], bcc[MAXN];
    set<pair<int, int>> bcce[MAXN];

    stack<pair<int, int>> s;

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

```
        adj[v].push_back(u);
    }

    int dfs(int u, int fa) {
        int lowu = pre[u] = ++dfs_clock;
        int child = 0;
        for (int v : adj[u]) {
            if (!pre[v]) {
                s.push({u, v});
                child++;
                int lowv = dfs(v, u);
                lowu = min(lowu, lowv);
                if (lowv >= pre[u]) {
                    iscut[u] = 1;
                    bcc[bcc_cnt].clear();
                    bcce[bcc_cnt].clear();
                    while (1) {
                        int xu, xv;
                        tie(xu, xv) = s.top(); s.pop();
                        bcce[bcc_cnt].insert({min(xu, xv), max(xu, xv)});
                        if (bccno[xu] != bcc_cnt) {
                            bcc[bcc_cnt].push_back(xu);
                            bccno[xu] = bcc_cnt;
                        }
                        if (bccno[xv] != bcc_cnt) {
                            bcc[bcc_cnt].push_back(xv);
                            bccno[xv] = bcc_cnt;
                        }
                        if (xu == u && xv == v) break;
                    }
                    bcc_cnt++;
                }
            } else if (pre[v] < pre[u] && v != fa) {
                s.push({u, v});
                lowu = min(lowu, pre[v]);
            }
        }
        if (fa < 0 && child == 1) iscut[u] = 0;
        return lowu;
    }

    void find_bcc(int n) {
        memset(pre, 0, sizeof pre);
        memset(iscut, 0, sizeof iscut);
        memset(bccno, -1, sizeof bccno);
        dfs_clock = bcc_cnt = 0;
        rep (i, n) if (!pre[i]) dfs(i, -1);
    }
};
```

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.

```
const int MAXN = 200005;
vector<int> adj[MAXN];
int dfn[MAXN], low[MAXN], idx;
bool cut[MAXN];

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

void tarjan(int u, int fa) {
    dfn[u] = low[u] = ++idx;
    int child = 0;
    for (int v : adj[u]) {
        if (!dfn[v]) {
            tarjan(v, fa); low[u] = min(low[u], low[v]);
            if (low[v] >= dfn[u] && u != fa) cut[u] = true;
            child += u == fa;
        }
        low[u] = min(low[u], dfn[v]);
    }
    if (u == fa && child > 1) cut[u] = true;
}
```

5.4 Minimum spanning arborescence, faster

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

Usage:

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

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

```
5ece const int MAXN = 300005;
2fef typedef pair<LL, int> pii;
1495 struct MDST {
01b2     priority_queue<pii, vector<pii>, greater<pii>> heap[MAXN];
321d     LL shift[MAXN];
fc06     int fa[MAXN], vis[MAXN];
427e 
38dd     int find(int x) { return fa[x] == x ? x : fa[x] = find(fa[x]); }
427e 
29b0     void unite(int x, int y) {
0c14         x = find(x); y = find(y); fa[y] = x; if (x == y) return;
6fa0         if (heap[x].size() < heap[y].size()) {
9c26             swap(heap[x], heap[y]);
2ffc             swap(shift[x], shift[y]);
95cf         }
9959         while (heap[y].size()) {
175b             auto p = heap[y].top(); heap[y].pop();
c0c5             heap[x].emplace(p.first - shift[y] + shift[x], p.second);
95cf         }
95cf     }
427e 
0bbd     void add_edge(int u, int v, LL w) { heap[v].emplace(w, u); }
427e 
a526     LL run(int n, int rt) {
f7ff         LL ans = 0;
81f2         iota(fa, fa + n + 1, 0);
19b3         Rep (i, n) if (find(i) != find(rt)) {
a7b1             int u = find(i);
010e             stack<int, vector<int>> s;
eff5             while (find(u) != find(rt)) {
0dda                 if (vis[u]) while (s.top() != u) {
c593                     vis[s.top()] = 0; unite(u, s.top()); s.pop();
83c4                 } else { vis[u] = 1; s.push(u); }
c76e                 while (heap[u].size()) {
b385                     ans += heap[u].top().first - shift[u];
                         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
             } 95cf
             return ans; 4206
         } 95cf
     }; 329b
```

5.5 Maximum flow (Dinic)

Usage:

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

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

```
struct edge{ bcf8
    int from, to; 60e2
    LL cap, flow; 5e6d
}; 329b
 427e
const int MAXN = 1005; e2cd
struct Dinic { 9062
    int n, m, s, t; 4dbf
    vector<edge> edges; 9f0c
    vector<int> G[MAXN]; b891
    bool vis[MAXN]; bbb6
    int d[MAXN]; b40a
    int cur[MAXN]; ddec
 427e
    void add_edge(int from, int to, LL cap) { 5973
        edges.push_back(edge{from, to, cap, 0}); 7b55
        edges.push_back(edge{to, from, 0, 0}); 1db7
        m = edges.size(); fe77
        G[from].push_back(m-2); dff5
        G[to].push_back(m-1); 8f2d
    } 95cf
 427e
```

```
1836     bool bfs() {
3b73         memset(vis, 0, sizeof(vis));
93d2         queue<int> q;
5d13         q.push(s);
2cd2         vis[s] = 1;
721d         d[s] = 0;
cc78         while (!q.empty()) {
66ba             int x = q.front(); q.pop();
3b61             for (int i = 0; i < G[x].size(); i++) {
b510                 edge& e = edges[G[x][i]];
bba9                 if (!vis[e.to] && e.cap > e.flow) {
cd72                     vis[e.to] = 1;
cf26                     d[e.to] = d[x] + 1;
ca93                     q.push(e.to);
95cf                 }
95cf             }
95cf         }
b23b         return vis[t];
95cf     }
427e
9252     LL dfs(int x, LL a) {
6904         if (x == t || a == 0) return a;
8bf9         LL flow = 0, f;
f515         for (int& i = cur[x]; i < G[x].size(); i++) {
b510             edge& e = edges[G[x][i]];
2374             if(d[x] + 1 == d[e.to] && (f = dfs(e.to, min(a, e.cap-e.flow))) > 0)
                   {
1cce                 e.flow += f;
e16d                 edges[G[x][i]^1].flow -= f;
a74d                 flow += f;
23e5                 a -= f;
97ed                 if(a == 0) break;
95cf             }
95cf         }
84fb         return flow;
95cf     }
427e
5bf2     LL max_flow(int s, int t) {
590d         this->s = s; this->t = t;
62e2         LL flow = 0;
ed58         while (bfs()) {
f326             memset(cur, 0, sizeof(cur));
fb3a             flow += dfs(s, LLONG_MAX);
95cf         }
```

```
        return flow;                                                      84fb
    }                                                                     95cf
                                                                          427e
    vector<int> min_cut() { // call this after maxflow                    c72e
        vector<int> ans;                                                  1df9
        for (int i = 0; i < edges.size(); i++) {                          df9a
            edge& e = edges[i];                                           56d8
            if(vis[e.from] && !vis[e.to] && e.cap > 0) ans.push_back(i);  46a2
        }                                                                 95cf
        return ans;                                                       4206
    }                                                                     95cf
};                                                                        329b
```

5.6 Maximum cardinality bipartite matching (Hungarian)

```
#include <bits/stdc++.h>                                302f
using namespace std;                                    421c
                                                        427e
#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) (x).begin(), (x).end()                 8843
typedef long long LL;                                   5cad
                                                        427e
struct Hungarian{                                       84ee
    int nx, ny;                                         fbf6
    vector<int> mx, my;                                 9ec6
    vector<vector<int> > e;                             9d4c
    vector<bool> mark;                                  edec
                                                        427e
    void init(int nx, int ny){                          8324
        this->nx = nx;                                  c1d1
        this->ny = ny;                                  f9c1
        mx.resize(nx); my.resize(ny);                   ac92
        e.clear(); e.resize(nx);                        3f11
        mark.resize(nx);                                1023
    }                                                   95cf
                                                        427e
    inline void add(int a, int b){                      4589
        e[a].push_back(b);                              486c
    }                                                   95cf
                                                        427e
    bool augment(int i){                                0c2b
```

```
        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;
```

```
1fc0             Rep (j, n) label[j] = -1;
7676             label[i] = 0; q = queue<int>(); q.push(i);
1c7d             while (q.size()) {
66ba                 int x = q.front(); q.pop();
b98c                 for (int y : adj[x]) {
c07f                     if (mate[y] == 0 and i != y) {
7f36                         mate[y] = x; rematch(x, y); q = queue<int>(); break;
95cf                     }
d315                     if (label[y] >= 0) { relabel(x, y); continue; }
58ec                     if (label[mate[y]] < 0) {
c9c4                         label[mate[y]] = x; q.push(mate[y]);
95cf                     }
95cf                 }
95cf             }
95cf         }
8abb         int cnt = 0;
b52f         Rep (i, n) cnt += (mate[i] > i);
6808         return cnt;
95cf     }
329b };
```

5.8 Minimum cost maximum flow

```
bcf8 struct edge{
60e2     int from, to;
d698     int cap, flow;
32cc     LL cost;
329b };
427e
cc3e const LL INF = LLONG_MAX / 2;
2aa8 const int MAXN = 5005;
c6cb struct MCMF {
9ceb     int s, t, n, m;
9f0c     vector<edge> edges;
b891     vector<int> G[MAXN];
f74f     bool inq[MAXN]; // queue
8f67     LL d[MAXN];     // distance
9524     int p[MAXN];    // previous
b330     int a[MAXN];    // improvement
427e
f7f2     void add_edge(int from, int to, int cap, LL cost) {
24f0         edges.push_back(edge{from, to, cap, 0, cost});
             edges.push_back(edge{to, from, 0, 0, -cost});   95f0
             m = edges.size();                                fe77
             G[from].push_back(m-2);                          dff5
             G[to].push_back(m-1);                            8f2d
         }                                                    95cf
                                                              427e
     bool spfa(){                                             3c52
         queue<int> q;                                        93d2
         fill(d, d + MAXN, INF); d[s] = 0;                    8494
         memset(inq, 0, sizeof(inq));                         fd48
         q.push(s); inq[s] = true;                            5e7c
         p[s] = 0; a[s] = INT_MAX;                            2dae
         while (!q.empty()){                                  cc78
             int u = q.front(); q.pop(); inq[u] = false;      b0aa
             for (int i : G[u]) {                             3bba
                 edge& e = edges[i];                          56d8
                 if (e.cap > e.flow && d[e.to] > d[u] + e.cost){    3601
                     d[e.to] = d[u] + e.cost;                 55bc
                     p[e.to] = G[u][i];                       0bea
                     a[e.to] = min(a[u], e.cap - e.flow);     8249
                     if (!inq[e.to]) q.push(e.to), inq[e.to] = true;    e5d3
                 }                                            95cf
             }                                                95cf
         }                                                    95cf
         return d[t] != INF;                                  6d7c
     }                                                        95cf
                                                              427e
     void augment(){                                          71a4
         int u = t;                                           06f1
         while (u != s){                                      b19d
             edges[p[u]].flow += a[t];                        db09
             edges[p[u]^1].flow -= a[t];                      25a9
             u = edges[p[u]].from;                            e6c9
         }                                                    95cf
     }                                                        95cf
                                                              427e
#ifdef GIVEN_FLOW                                             6e20
     bool min_cost(int s, int t, int f, LL& cost) {           5972
         this->s = s; this->t = t;                            590d
         int flow = 0;                                        21d4
         cost = 0;                                            23cb
         while (spfa()) {                                     22dc
             augment();                                       bcdb
             if (flow + a[t] >= f){                           a671
```

```
b14d                cost += (f - flow) * d[t]; flow = f;
3361                return true;
8e2e            } else {
2a83                flow += a[t]; cost += a[t] * d[t];
95cf            }
95cf        }
438e        return false;
95cf    }
a8cb #else
f9a9    int min_cost(int s, int t, LL& cost) {
590d        this->s = s; this->t = t;
21d4        int flow = 0;
23cb        cost = 0;
22dc        while (spfa()) {
bcdb            augment();
2a83            flow += a[t]; cost += a[t] * d[t];
95cf        }
84fb        return flow;
95cf    }
1937 #endif
329b };
```

5.9 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.

```
0e34 const int MAXN = 500005;
0b32 vector<int> adj[MAXN];
fccb int id[MAXN], nid;
1356 pair<int, int> st[MAXN << 1][33 - __builtin_clz(MAXN)];
427e
e16d void dfs(int u, int p, int d) {
0df2     st[id[u] = nid++][0] = {d, u};
18f6     for (int v : adj[u]) {
bd87         if (v == p) continue;
f58c         dfs(v, u, d + 1);
08ad         st[nid++][0] = {d, u};
95cf     }
95cf }
427e
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
int lca(int u, int v) {                                          0f0b
    tie(u, v) = minmax(id[u], id[v]);                            cfc4
    int k = 31 - __builtin_clz(v-u+1);                           be9b
    return min(st[u][k], st[v-(1<<k)+1][k]).second;              8ebc
}                                                                95cf
```

5.10 Heavy-light decomposition

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

```
const int MAXN = 100005;                                                         0f42
vector<int> adj[MAXN];                                                           0b32
int sz[MAXN], top[MAXN], fa[MAXN], son[MAXN], depth[MAXN], id[MAXN];             42f2
                                                                                 427e
void dfs1(int x, int dep, int par){                                              be5c
    depth[x] = dep;                                                              7489
    sz[x] = 1;                                                                   2ee7
    fa[x] = par;                                                                 adb4
    int maxn = 0, s = 0;                                                         b79d
    for (int c: adj[x]){                                                         c861
        if (c == par) continue;                                                  fe45
        dfs1(c, dep + 1, x);                                                     fd2f
        sz[x] += sz[c];                                                          b790
        if (sz[c] > maxn){                                                       f0f1
            maxn = sz[c];                                                        c749
            s = c;                                                               fe19
        }                                                                        95cf
    }                                                                            95cf
    son[x] = s;                                                                  0e08
}                                                                                95cf
                                                                                 427e
int cid = 0;                                                                     ba54
void dfs2(int x, int t){                                                         3644
    top[x] = t;                                                                  8d96
```

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

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

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

5.11 Centroid decomposition

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

Usage:

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

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

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

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

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

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

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

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

5.12 DSU on tree

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

Usage:

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

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

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

```

```
5559 void decomp(int u, int p) {
50c0     sz[u] = 1;
18f6     for (int v : adj[u]) {
bd87         if (v == p) continue;
a851         decomp(v, u);
8449         sz[u] += sz[v];
d28c         if (sz[v] > sz[son[u]]) son[u] = v;
95cf     }
95cf }
427e
b7ec template <typename T>
62f5 void trav(T fn, int u, int p) {
4412     fn(u);
30b3     for (int v : adj[u]) if (v != p) trav(fn, v, u);
95cf }
427e
7467 #define for_light(v) for (int v : adj[u]) if (v != p and v != son[u])
33ff void work(int u, int p, bool keep) {
72a2     for_light(v) work(v, u, 0); // process light children
427e
427e     // process heavy child
427e     // current data structure contains info of heavy child
9866     if (son[u]) work(son[u], u, 1);
427e
18a9     auto merge = [u] (int c) { /* count contribution of c */ };
1ab0     auto enter = [] (int c) { /* add vertex c */ };
f241     auto leave = [] (int c) { /* remove vertex c*/ };
427e
3d3b     for_light(v) {
74c6         trav(merge, v, u);
c13d         trav(enter, v, u);
95cf     }
427e
427e     // count answer for root and add it
427e     // Warning: special check may apply to root!
c54f     merge(u);
9dec     enter(u);
427e
427e     // leave current tree
4e3e     if (!keep) trav(leave, u, p);
95cf }
```

6 Data Structures

6.1 Fenwick tree (point update range query)

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

6.2 Fenwick tree (range update point query)

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

6.3 Segment tree

```
LL p;
const int MAXN = 4 * 100006;
struct segtree {
  int l[MAXN], m[MAXN], r[MAXN];
  LL val[MAXN], tadd[MAXN], tmul[MAXN];

#define lson (o<<1)
#define rson (o<<1|1)

  void pull(int o) {
    val[o] = (val[lson] + val[rson]) % p;
  }

  void push_add(int o, LL x) {
    val[o] = (val[o] + x * (r[o] - l[o])) % p;
    tadd[o] = (tadd[o] + x) % p;
  }

  void push_mul(int o, LL x) {
    val[o] = val[o] * x % p;
    tadd[o] = tadd[o] * x % p;
    tmul[o] = tmul[o] * x % p;
  }

  void push(int o) {
    if (l[o] == m[o]) return;
    if (tmul[o] != 1) {
      push_mul(lson, tmul[o]);
      push_mul(rson, tmul[o]);
      tmul[o] = 1;
    }
    if (tadd[o]) {
      push_add(lson, tadd[o]);
      push_add(rson, tadd[o]);
      tadd[o] = 0;
    }
  }

  void build(int o, int ll, int rr) {
    int mm = (ll + rr) / 2;
    l[o] = ll; r[o] = rr; m[o] = mm;
    tmul[o] = 1;
    if (ll == mm) {
      scanf("%lld", val + o);
      val[o] %= p;
    } else {
      build(lson, ll, mm);
      build(rson, mm, rr);
      pull(o);
    }
  }

  void add(int o, int ll, int rr, LL x) {
    if (ll <= l[o] && r[o] <= rr) {
      push_add(o, x);
    } else {
      push(o);
      if (m[o] > ll) add(lson, ll, rr, x);
      if (m[o] < rr) add(rson, ll, rr, x);
      pull(o);
    }
  }

  void mul(int o, int ll, int rr, LL x) {
    if (ll <= l[o] && r[o] <= rr) {
      push_mul(o, x);
    } else {
      push(o);
      if (ll < m[o]) mul(lson, ll, rr, x);
      if (m[o] < rr) mul(rson, ll, rr, x);
      pull(o);
    }
  }

  LL query(int o, int ll, int rr) {
    if (ll <= l[o] && r[o] <= rr) {
      return val[o];
    } else {
      push(o);
      if (rr <= m[o]) return query(lson, ll, rr);
      if (ll >= m[o]) return query(rson, ll, rr);
      return query(lson, ll, rr) + query(rson, ll, rr);
    }
  }
} seg;
```

6.4 Treap

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

Usage:

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

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

```cpp
const int MAXN = 200005;
mt19937 gen(time(NULL));
struct Treap {
    int ch[MAXN][2];
    int sz[MAXN], key[MAXN], val[MAXN];
    int add[MAXN], rev[MAXN];
    LL sum[MAXN] = {0};
    int maxv[MAXN] = {INT_MIN}, minv[MAXN] = {INT_MAX};

    void Init(int x, int v) {
        ch[x][0] = ch[x][1] = 0;
        key[x] = gen(); val[x] = v; pull(x);
    }

    void pull(int x) {
        sz[x] = 1 + sz[ch[x][0]] + sz[ch[x][1]];
        sum[x] = val[x] + sum[ch[x][0]] + sum[ch[x][1]];
        maxv[x] = max({val[x], maxv[ch[x][0]], maxv[ch[x][1]]});
        minv[x] = min({val[x], minv[ch[x][0]], minv[ch[x][1]]});
    }

    void Add(int x, int a) {
        val[x] += a; add[x] += a;
        sum[x] += LL(sz[x]) * a; maxv[x] += a; minv[x] += a;
    }

    void Reverse(int x) {
        rev[x] ^= 1;
        swap(ch[x][0], ch[x][1]);
    }

    void push(int x) {
        for (int c : ch[x]) if (c) {
            Add(c, add[x]);
            if (rev[x]) Reverse(c);
        }
        add[x] = 0; rev[x] = 0;
    }

    int Merge(int x, int y) {
        if (!x || !y) return x | y;
        push(x); push(y);
        if (key[x] > key[y]) {
            ch[x][1] = Merge(ch[x][1], y); pull(x); return x;
        } else {
            ch[y][0] = Merge(x, ch[y][0]); pull(y); return y;
        }
    }

    void Split(int t, int k, int &x, int &y) {
        if (t == 0) { x = y = 0; return; }
        push(t);
        if (sz[ch[t][0]] < k) {
            x = t; Split(ch[t][1], k - sz[ch[t][0]] - 1, ch[t][1], y);
        } else {
            y = t; Split(ch[t][0], k, x, ch[t][0]);
        }
        if (x) pull(x); if (y) pull(y);
    }
} treap;

int root;

void init(int n) {
    Rep (i, n) {
        int x; scanf("%d", &x);
        treap.Init(i, x);
        root = (i == 1) ? 1 : treap.Merge(root, i);
```

```
    }
}

void work(int op, int l, int r) {
    int tl, tm, tr;
    treap.Split(root, l, tl, tm);
    treap.Split(tm, r - l, tm, tr);
    if (op == 1) {
        int x; scanf("%d", &x); treap.Add(tm, x);
    } else if (op == 2) {
        treap.Reverse(tm);
    } else if (op == 3) {
        printf("%lld %d %d\n",
               treap.sum[tm], treap.minv[tm], treap.maxv[tm]);
    }
    root = treap.Merge(treap.Merge(tl, tm), tr);
}
```

6.5 Link/cut tree

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

Usage:

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

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

```
const int MAXN = 1000005;
struct LCT {
    int fa[MAXN], ch[MAXN][2], val[MAXN], sum[MAXN];
    bool rev[MAXN];

    bool isroot(int x) { return ch[fa[x]][0] == x || ch[fa[x]][1] == x; }
    void pull(int x) { sum[x] = val[x] ^ sum[ch[x][0]] ^ sum[ch[x][1]]; }
    void reverse(int x) { swap(ch[x][0], ch[x][1]); rev[x] ^= 1; }
    void push(int x) {
        if (rev[x]) rep (i, 2) if (ch[x][i]) reverse(ch[x][i]); rev[x] = 0;
    }
    void rotate(int x) {
        int y = fa[x], z = fa[y], k = ch[y][1] == x, w = ch[x][!k];
        if (isroot(y)) ch[z][ch[z][1] == y] = x;
        ch[x][!k] = y; ch[y][k] = w; if (w) fa[w] = y;
        fa[y] = x; fa[x] = z; pull(y);
    }
    void pushall(int x) { if (isroot(x)) pushall(fa[x]); push(x); }
    void splay(int x) {
        int y = x, z = 0;
        for (pushall(y); isroot(x); rotate(x)) {
            y = fa[x]; z = fa[y];
            if (isroot(y)) rotate((ch[y][0] == x) ^ (ch[z][0] == y) ? x : y);
        }
        pull(x);
    }
    void access(int x) {
        int z = x;
        for (int y = 0; x; x = fa[y = x]) { splay(x); ch[x][1] = y; pull(x); }
        splay(z);
    }
    void chroot(int x) { access(x); reverse(x); }
    void split(int x, int y) { chroot(x); access(y); }

    int Root(int x) {
        for (access(x); ch[x][0]; x = ch[x][0]) push(x);
        splay(x); return x;
    }
    void Link(int u, int v) { chroot(u); fa[u] = v; }
    void Cut(int u, int v) { split(u, v); fa[u] = ch[v][0] = 0; pull(v); }
    int Query(int u, int v) { split(u, v); return sum[v]; }
    void Update(int u, int x) { splay(u); val[u] = x; }
    int LCA(int x, int y, int root) {
        chroot(root); access(x); splay(y);
        while (fa[y]) splay(y = fa[y]);
        return y;
    }
};
```

6.6 Balanced binary search tree from `pb_ds`

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

6.7 Persistent segment tree, range k-th query

```
f1a7 struct node {
2ff6   static int n, pos;
427e
7cec   int value;
70e2   node *left, *right;
427e
20b0   void* operator new(size_t size);
427e
3dc0   static node* Build(int l, int r) {
b6c5     node* a = new node;
ce96     if (r > l + 1) {
181e       int mid = (l + r) / 2;
3ba2       a->left = Build(l, mid);
8aaf       a->right = Build(mid, r);
8e2e     } else {
bfc4       a->value = 0;
95cf     }
5ffd     return a;
95cf   }
427e
5a45   static node* init(int size) {
2c46     n = size;
    pos = 0;                                                                   7ee3
    return Build(0, n);                                                        be52
  }                                                                            95cf
                                                                               427e
  static int Query(node* lt, node *rt, int l, int r, int k) {                  93c0
    if (r == l + 1) return l;                                                  d30c
    int mid = (l + r) / 2;                                                     181e
    if (rt->left->value - lt->left->value < k) {                               cb5a
      k -= rt->left->value - lt->left->value;                                  8edb
      return Query(lt->right, rt->right, mid, r, k);                           2412
    } else {                                                                   8e2e
      return Query(lt->left, rt->left, l, mid, k);                             0119
    }                                                                          95cf
  }                                                                            95cf
                                                                               427e
  static int query(node* lt, node *rt, int k) {                                c9ad
    return Query(lt, rt, 0, n, k);                                             9e27
  }                                                                            95cf
                                                                               427e
  node *Inc(int l, int r, int pos) const {                                     b19c
    node* a = new node(*this);                                                 5794
    if (r > l + 1) {                                                           ce96
      int mid = (l + r) / 2;                                                   181e
      if (pos < mid)                                                           203d
        a->left = left->Inc(l, mid, pos);                                      f44a
      else                                                                     649a
        a->right = right->Inc(mid, r, pos);                                    1024
    }                                                                          95cf
    a->value++;                                                                2b3e
    return a;                                                                  5ffd
  }                                                                            95cf
                                                                               427e
  node *inc(int index) {                                                       e80f
    return Inc(0, n, index);                                                   c246
  }                                                                            95cf
} nodes[8000000];                                                              865a
                                                                               427e
int node::n, node::pos;                                                        99ce
inline void* node::operator new(size_t size) {                                 1987
  return nodes + (pos++);                                                      bb3c
}                                                                              95cf
```

6.8 Block list

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

Usage:

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

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

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

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

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

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

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

typedef list<block>::iterator lit;

struct blocklist {
    list<block> blk;

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

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

        }
    }

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

    void Reverse(int l, int r) {
        lit it = split(l), it2 = split(r);
        reverse(it, it2);
        while (it != it2) {
            it->rev ^= 1;
            it++;
        }
        maintain();
    }

    void Add(int l, int r, int x) {
```

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

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

6.9 Persistent block list

Block list that supports persistence. All indices are 0-based. All ranges are left-closed right-open. `std::shared_ptr` is used to ease memory management. One should modify the constructor of `block` to maintain extra information. Here we use this policy that the size of each block does not exceed `BLOCK`, while the sum of sizes of two adjacent blocks does not less than `BLOCK`.
When some operation that breaks block list property, please call `maintain` in time to restore the property.

Usage:

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

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

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

struct block {
    pcvi data;
    LL sum;

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

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

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

typedef list<block>::iterator lit;

struct blocklist {
    list<block> blk;

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

```

```
b7b3     lit split(int pos) {
2273         for (lit it = blk.begin(); ; it++) {
5502             if (pos == 0) return it;
d480             while (it->data->size() > pos) {
2099                 blk.insert(next(it), it->split(pos));
95cf             }
a1c8             pos -= it->data->size();
95cf         }
95cf     }
427e
fd38     LL sum(int l, int r) { // traverse
48b4         lit it1 = split(l), it2 = split(r);
ac09         LL res = 0;
9f1d         while (it1 != it2) {
8284             res += it1->sum;
61fd             it1++;
95cf         }
b204         maintain();
244d         return res;
95cf     }
329b };
```

6.10 Sparse table, range 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

```
#include <bits/stdc++.h>                                                302f
using namespace std;                                                    421c
                                                                        427e
typedef int T;                                                          4553
typedef struct pt {                                                     c0ae
    T x, y;                                                             7a9d
    T operator , (pt a) { return x*a.x + y*a.y; } // inner product      ffaa
    T operator * (pt a) { return x*a.y - y*a.x; } // outer product      3ec7
    pt operator + (pt a) { return {x+a.x, y+a.y}; }                     221a
    pt operator - (pt a) { return {x-a.x, y-a.y}; }                     8b34
                                                                        427e
    pt operator * (T k) { return {x*k, y*k}; }                          368b
    pt operator - () { return {-x, -y};}                                90f4
} vec;                                                                  ba8c
                                                                        427e
typedef pair<pt, pt> seg;                                               0ea6
                                                                        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
427e // >0 in order
427e // <0 out of order
427e // =0 not standard
7538 inline double rotOrder(vec a, vec b, vec c){return double(a*b)*(b*c);}
427e
31ed inline bool intersect(seg a, seg b){
427e     // ! if (nIntRectRect(a, b)) return false; // if commented, assume that a
           and b are non-collinear
cb52     return rotOrder(b.first-a.first, a.second-a.first, b.second-a.first) >= 0 &&
059e            rotOrder(a.first-b.first, b.second-b.first, a.second-b.first) >= 0;
95cf }
427e
427e // 0 not insersect
427e // 1 standard intersection
427e // 2 vertex-line intersection
427e // 3 vertex-vertex intersection
427e // 4 collinear and have common point(s)
4d19 int intersect2(seg& a, seg& b){
5dc4     if (nIntRectRect(a, b)) return 0;
42c0     vec va = a.second - a.first, vb = b.second - b.first;
2096     double j1 = rotOrder(b.first-a.first, va, b.second-a.first),
72fe            j2 = rotOrder(a.first-b.first, vb, a.second-b.first);
5ac6     if (j1 < 0 || j2 < 0) return 0;
9400     if (j1 != 0 && j2 != 0) return 1;
83db     if (j1 == 0 && j2 == 0){
6b0c         if (va * vb == 0) return 4; else return 3;
fb17     } else return 2;
95cf }
427e
2c68 template <typename Tp = T>
5894 inline pt getIntersection(pt P, vec v, pt Q, vec w){
6850     static_assert(is_same<Tp, double>::value, "must␣be␣double!");
7c9a     return P + v * (w*(P-Q)/(v*w));
95cf }
427e
427e // -1 outside the polygon
427e // 0  on the border of the polygon
427e // 1  inside the polygon
cbdd int ptOnPoly(pt p, pt* poly, int n){
5fb4     int wn = 0;
1294     for (int i = 0; i < n; i++) {
427e
3cae         T k, d1 = poly[i].y - p.y, d2 = poly[(i+1)%n].y - p.y;
```

```
        if (k = (poly[(i+1)%n] - poly[i])*(p - poly[i])){     b957
            if (k > 0 && d1 <= 0 && d2 > 0) wn++;             8c40
            if (k < 0 && d2 <= 0 && d1 > 0) wn--;             3c4d
        } else return 0;                                      aad3
    }                                                         95cf
    return wn ? 1 : -1;                                       0a5f
}                                                             95cf
                                                              427e
istream& operator >> (istream& lhs, pt& rhs){                 d4a3
    lhs >> rhs.x >> rhs.y;                                    fa86
    return lhs;                                               331a
}                                                             95cf
                                                              427e
istream& operator >> (istream& lhs, seg& rhs){                07ae
    lhs >> rhs.first >> rhs.second;                           5cab
    return lhs;                                               331a
}                                                             95cf
```

8 Appendices

8.1 Primes

8.1.1 First primes

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

8.1.2 Arbitrary length primes

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

8.1.3 $\sim 1 \times 10^9$

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

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

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

8.2 Pell's equation

$x^2 - ny^2 = 1$, where n is a positive nonsquare integer.

Let (x_0, y_0) be the smallest positive solution of the equation, then the k-th solution is:

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

Some smallest solutions to Pell's equation:

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

8.3 Burnside's lemma and Polya's enumeration theorem

The Burnside's lemma says that

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

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

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

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

8.4 Lagrange's interpolation

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

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

then the Lagrange polynomial is

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

To use the script below, type two lines

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

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

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