南京大学 ACM-ICPC 集训队代码模版库

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
e458 set statusline=\ %<%F[%1*%M%*%n%R%H]%=\ %y\ %0(%{&fileformat}\ %{&encoding}\ %c
       :%l/%L%)\
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

```cpp
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 i=0; i<G[x].size(); i++)
            if (!dfs(G[x][i])) 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);
        G[y^1].push_back(x);
    }

    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;
    }

    inline bool value(unsigned i){return mark[2*i+1];}
};
```

2.2 Knuth's optimization

```cpp
int n;
int dp[256][256], dc[256][256];

template <typename T>
void compute(T cost) {
  for (int i = 0; i <= n; i++) {
    dp[i][i] = 0;
    dc[i][i] = i;
  }
  rep (i, n) {
    dp[i][i+1] = 0;
    dc[i][i+1] = i;
  }
  for (int len = 2; len <= n; len++) {
    for (int i = 0; i + len <= n; i++) {
      int j = i + len;
      int lbnd = dc[i][j-1], rbnd = dc[i+1][j];
      dp[i][j] = INT_MAX / 2;
      int c = cost(i, j);
      for (int k = lbnd; k <= rbnd; k++) {
        int res = dp[i][k] + dp[k][j] + c;
        if (res < dp[i][j]) {
          dp[i][j] = res;
          dc[i][j] = k;
        }
      }
    }
  }
};
```

2.3 Mo's algorithm

All intervals are closed on both sides. When running functions `enter()` and `leave()`, the global l and r has not changed yet.

Usage:

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

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

```
    yield(q.id);                                                     82f5
  }                                                                  95cf
}                                                                    95cf
```

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
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;
```

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

3.4 Suffix array

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

Usage:

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

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

3.5 Trie

```
e6f1  const int MAXN = 12000;
dd87  const int CHARN = 26;
427e
8ff5  inline int id(char c) { return c - 'a'; }
427e
a281  struct Trie {
5c83    int n;
f4f5    int tr[MAXN][CHARN];  // Trie tree, 0 denotes fail
35a5    int tag[MAXN];
427e
4fee    Trie() {
3ccc      memset(tr[0], 0, sizeof(tr[0]));
4d52      tag[0] = 0;
46bf      n = 1;
```

```
  }

  // 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];
  }
};
```

3.6 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] = pg[i - 1] * g % mod;
}

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;
    }
} ha;
```

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

4.2 Matrix powermod

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

4.3 Linear basis

```
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
```

4.4 Gauss elimination over finite field

```
const LL p = 1000000007;                                        b784
                                                                427e
LL powmod(LL b, LL e) {                                         2a2c
  LL r = 1;                                                     95a2
  while (e) {                                                   3e90
    if (e & 1) r = r * b % p;                                   1783
    b = b * b % p;                                              5549
    e >>= 1;                                                    16fc
  }                                                             95cf
  return r;                                                     547e
}                                                               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;

  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;
}
```

4.5 Berlekamp-Massey algorithm

```
const LL MOD = 1000000007;

LL inverse(LL b) {
  LL e = MOD - 2, r = 1;
  while (e) {
    if (e & 1) r = r * b % MOD;
    b = b * b % MOD;
    e >>= 1;
  }
  return r;
}

struct Poly {
  vector<int> a;

  Poly() { a.clear(); }

  Poly(vector<int> &a) : a(a) {}

  int length() const { return a.size(); }

  Poly move(int d) {
    vector<int> na(d, 0);
    na.insert(na.end(), a.begin(), a.end());
    return Poly(na);
  }

  int calc(vector<int> &d, int pos) {
    int ret = 0;
    for (int i = 0; i < (int)a.size(); ++i) {
      if ((ret += (long long)d[pos - i] * a[i] % MOD) >= MOD) {
        ret -= MOD;
      }
    }
    return ret;
  }

  Poly operator - (const Poly &b) {
    vector<int> na(max(this->length(), b.length()));
    for (int i = 0; i < (int)na.size(); ++i) {
      int aa = i < this->length() ? this->a[i] : 0,
          bb = i < b.length() ? b.a[i] : 0;
      na[i] = (aa + MOD - bb) % MOD;
    }
    return Poly(na);
  }
};
```

```

Poly operator * (const int &c, const Poly &p) {
  vector<int> na(p.length());
  for (int i = 0; i < (int)na.size(); ++i) {
    na[i] = (long long)c * p.a[i] % MOD;
  }
  return na;
}

vector<int> solve(vector<int> a) {
  int n = a.size();
  Poly s, b;
  s.a.push_back(1), b.a.push_back(1);
  for (int i = 1, j = 0, ld = a[0]; i < n; ++i) {
    int d = s.calc(a, i);
    if (d) {
      if ((s.length() - 1) * 2 <= i) {
        Poly ob = b;
        b = s;
        s = s - (long long)d * inverse(ld) % MOD * ob.move(i - j);
        j = i;
        ld = d;
      } else {
        s = s - (long long)d * inverse(ld) % MOD * b.move(i - j);
      }
    }
  }
  // Caution: s.a might be shorter than expected
  return s.a;
}
```

4.6 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];
                // a[i+j] = x+y, a[i+j+d] = x-y;    // xor
                // a[i+j] = x+y;                    // and
                // a[i+j+d] = x+y;                  // or
            }
}

void ifwt(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];
                // a[i+j] = (x+y)/2, a[i+j+d] = (x-y)/2;    // xor
                // a[i+j] = x-y;                            // and
                // a[i+j+d] = y-x;                          // or
            }
}

void conv(int* a, int* b, int n){
    fwt(a, n);
    fwt(b, n);
    rep(i, n) a[i] *= b[i];
    ifwt(a, n);
}
```

4.7 Fast fourier transform

```
const int NMAX = 1<<20;

typedef complex<double> cplx;

const double PI = 2*acos(0.0);
struct FFT{
    int rev[NMAX];
    cplx omega[NMAX], oinv[NMAX];
    int K, N;

    FFT(int k){
        K = k; N = 1 << k;
        rep (i, N){
            rev[i] = (rev[i>>1]>>1) | ((i&1)<<(K-1));
            omega[i] = polar(1.0, 2.0 * PI / N * i);
            oinv[i] = conj(omega[i]);
        }
    }

    void dft(cplx* a, cplx* w){
```

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

4.8 Number theoretic transform

```
4ab9  const int NMAX = 1<<21;
427e
427e  // 998244353 = 7*17*2^23+1, G = 3
fb9a  const int P = 1004535809, G = 3; // = 479*2^21+1
427e
87ab  struct NTT{
c47c      int rev[NMAX];
0eda      LL omega[NMAX], oinv[NMAX];
81af      int g, g_inv; // g: g_n = G^((P-1)/n)
9827      int K, N;
427e
2a2c      LL powmod(LL b, LL e){
95a2          LL r = 1;
3e90          while (e){
6624              if (e&1) r = r * b % P;
489e              b = b * b % P;
```

```
              e >>= 1;                                            16fc
          }                                                       95cf
          return r;                                               547e
      }                                                           95cf
                                                                  427e
      NTT(int k){                                                 f420
          K = k; N = 1 << k;                                      e209
          g = powmod(G, (P-1)/N);                                 7652
          g_inv = powmod(g, N-1);                                 4b3a
          omega[0] = oinv[0] = 1;                                 e04f
          rep (i, N){                                             b393
              rev[i] = (rev[i>>1]>>1) | ((i&1)<<(K-1));           7ba3
              if (i){                                             ad4f
                  omega[i] = omega[i-1] * g % P;                  8d8b
                  oinv[i] = oinv[i-1] * g_inv % P;                9e14
              }                                                   95cf
          }                                                       95cf
      }                                                           95cf
                                                                  427e
      void _ntt(LL* a, LL* w){                                    9668
          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 (LL* p = a; p != a + N; p += l)                 7a1d
                  rep (k, m){                                     c24f
                      LL t = w[N/l*k] * p[k+m] % P;               0ad3
                      p[k+m] = (p[k] - t + P) % P;                6209
                      p[k] = (p[k] + t) % P;                      fa1b
                  }                                               95cf
          }                                                       95cf
      }                                                           95cf
                                                                  427e
      void ntt(LL* a){_ntt(a, omega);}                            92ea
      void intt(LL* a){                                           5daf
          LL inv = powmod(N, P-2);                                1f2a
          _ntt(a, oinv);                                          9910
          rep (i, N) a[i] = a[i] * inv % P;                       a873
      }                                                           95cf
                                                                  427e
      void conv(LL* a, LL* b){                                    3a5b
          ntt(a); ntt(b);                                         ad16
          rep (i, N) a[i] = a[i] * b[i] % P;                      e49e
          intt(a);                                                5748
      }                                                           95cf
```

```
329b };
```

4.9 Sieve of Euler

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

4.10 Sieve of Euler (General)

```
b62e namespace sieve {
6589   constexpr int MAXN = 10000007;
e982   bool p[MAXN]; // true if not prime
6ae8   int prime[MAXN], sz;
cbf7   int pval[MAXN], pcnt[MAXN];
6030   int f[MAXN];
427e 
76f6   void exec(int N = MAXN) {
9628     p[0] = p[1] = 1;
427e 
8a8a     pval[1] = 1;
bdda     pcnt[1] = 0;
c6b9     f[1] = 1;
427e 
a643     for (int i = 2; i < N; i++) {
01d6       if (!p[i]) {
b2b2         prime[sz++] = i;
37d9         for (LL j = i; j < N; j *= i) {
758c           int b = j / i;
81fd           pval[j] = i * pval[b];
           pcnt[j] = pcnt[b] + 1;                        e0f3
           f[j] = ______; // f[j] = f(i^pcnt[j])         a96c
         }                                               95cf
       }                                                 95cf
       for (int j = 0; i * prime[j] < N; j++) {          34c0
         int x = i * prime[j]; p[x] = 1;                 f87a
         if (i % prime[j] == 0) {                        20cc
           pval[x] = pval[i] * prime[j];                 9985
           pcnt[x] = pcnt[i] + 1;                        3f93
         } else {                                        8e2e
           pval[x] = prime[j];                           cc91
           pcnt[x] = 1;                                  6322
         }                                               95cf
         if (x != pval[x]) {                             6191
           f[x] = f[x / pval[x]] * f[pval[x]]            d614
         }                                               95cf
         if (i % prime[j] == 0) break;                   5f51
       }                                                 95cf
     }                                                   95cf
   }                                                     95cf
 }                                                       95cf
```

4.11 Miller-Rabin primality test

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

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

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

4.12 Pollard's rho algorithm

```
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;
}
```

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

```
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});
fdcf                 child++;
f851                 int lowv = dfs(v, u);
189c                 lowu = min(lowu, lowv);
b687                 if (lowv >= pre[u]) {
6323                     iscut[u] = 1;
57eb                     bcc[bcc_cnt].clear();
90b8                     bcce[bcc_cnt].clear();
a147                     while (1) {
a6a3                         int xu, xv;
a0c3                         tie(xu, xv) = s.top(); s.pop();
0ef5                         bcce[bcc_cnt].insert({min(xu, xv), max(xu, xv)});
3db2                         if (bccno[xu] != bcc_cnt) {
e0db                             bcc[bcc_cnt].push_back(xu);
d27f                             bccno[xu] = bcc_cnt;
95cf                         }
f357                         if (bccno[xv] != bcc_cnt) {
752b                             bcc[bcc_cnt].push_back(xv);
57c9                             bccno[xv] = bcc_cnt;
95cf                         }
```

```
                        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 Minimum spanning arborescence (Chu-Liu)

All vertices are 1-based.

Usage:

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

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

```
struct edge {                                               bcf8
    int u, v;                                               54f1
    LL w;                                                   309c
};                                                          329b
                                                            427e
const int MAXN = 10005;                                     f5a4
LL in[MAXN];                                                7124
int pre[MAXN], vis[MAXN], id[MAXN];                         1c1d
                                                            427e
LL getans(int n, int rt, vector<edge>& edges) {             5a43
    LL ans = 0;                                             f7ff
    int cnt = 0;                                            8abb
    while (1) {                                             a147
641a          Rep (i, n) in[i] = LLONG_MAX, id[i] = vis[i] = 0;
0705          for (auto e : edges) {
073a              if (e.u != e.v and e.w < in[e.v]) {
c1df                  pre[e.v] = e.u;
5fbc                  in[e.v] = e.w;
95cf              }
95cf          }
3fdb          in[rt] = 0;
34d7          Rep (i, n) {
3c97              if (in[i] == LLONG_MAX) return -1;
cf57              ans += in[i];
a763              int u;
4b0e              for (u = i; u != rt && vis[u] != i && !id[u]; u = pre[u])
88a2                  vis[u] = i;
4b22              if (u != rt && !id[u]) {
b66e                  id[u] = ++cnt;
0443                  for (int v = pre[u]; v != u; v = pre[v])
5c22                      id[v] = cnt;
95cf              }
95cf          }
91e9          if (!cnt) return ans;
5e22          Rep (i, n) if (!id[i]) id[i] = ++cnt;
7400          for (auto& e : edges) {
7750              LL laz = in[e.v];
97ae              e.u = id[e.u];
fae6              e.v = id[e.v];
bdd2              if (e.u != e.v) e.w -= laz;
95cf          }
6cc4          n = cnt; rt = id[rt]; cnt = 0;
95cf      }
95cf  }
```

5.4 Maximum flow (Dinic)

Usage:

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

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

```
bcf8  struct edge{
60e2      int from, to;
5e6d      LL cap, flow;
```

```
};                                                               329b
                                                                 427e
const int MAXN = 1005;                                           e2cd
struct Dinic {                                                   9062
    int n, m, s, t;                                              4dbf
    vector<edge> edges;                                          9f0c
    vector<int> G[MAXN];                                         b891
    bool vis[MAXN];                                              bbb6
    int d[MAXN];                                                 b40a
    int cur[MAXN];                                               ddec
                                                                 427e
    void add_edge(int from, int to, LL cap) {                    5973
        edges.push_back(edge{from, to, cap, 0});                 7b55
        edges.push_back(edge{to, from, 0, 0});                   1db7
        m = edges.size();                                        fe77
        G[from].push_back(m-2);                                  dff5
        G[to].push_back(m-1);                                    8f2d
    }                                                            95cf
                                                                 427e
    bool bfs() {                                                 1836
        memset(vis, 0, sizeof(vis));                             3b73
        queue<int> q;                                            93d2
        q.push(s);                                               5d13
        vis[s] = 1;                                              2cd2
        d[s] = 0;                                                721d
        while (!q.empty()) {                                     cc78
            int x = q.front(); q.pop();                          66ba
            for (int i = 0; i < G[x].size(); i++) {              3b61
                edge& e = edges[G[x][i]];                        b510
                if (!vis[e.to] && e.cap > e.flow) {              bba9
                    vis[e.to] = 1;                               cd72
                    d[e.to] = d[x] + 1;                          cf26
                    q.push(e.to);                                ca93
                }                                                95cf
            }                                                    95cf
        }                                                        95cf
        return vis[t];                                           b23b
    }                                                            95cf
                                                                 427e
    LL dfs(int x, LL a) {                                        9252
        if (x == t || a == 0) return a;                          6904
        LL flow = 0, f;                                          8bf9
        for (int& i = cur[x]; i < G[x].size(); i++) {            f515
            edge& e = edges[G[x][i]];                            b510
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         }
84fb         return flow;
95cf     }
427e
c72e     vector<int> min_cut() { // call this after maxflow
1df9         vector<int> ans;
df9a         for (int i = 0; i < edges.size(); i++) {
56d8             edge& e = edges[i];
46a2             if(vis[e.from] && !vis[e.to] && e.cap > 0) ans.push_back(i);
95cf         }
4206         return ans;
95cf     }
329b };
```

5.5 Maximum cardinality bipartite matching (Hungarian)

```
302f #include <bits/stdc++.h>
421c using namespace std;
427e
0d6c #define rep(i, n) for (int i = 0; i < (n); i++)
cfe3 #define Rep(i, n) for (int i = 1; i <= (n); i++)
8843 #define range(x) (x).begin(), (x).end()
5cad typedef long long LL;
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]) {                                    207c
            mark[i] = true;                                dae4
            for (int j : e[i]){                            6a1e
                if (my[j] == -1 || augment(my[j])){        0892
                    mx[i] = j; my[j] = i;                  9ca3
                    return true;                           3361
                }                                          95cf
            }                                              95cf
        }                                                  95cf
        return false;                                      438e
    }                                                      95cf
                                                           427e
    int match(){                                           3fac
        int ret = 0;                                       5b57
        fill(range(mx), -1);                               b0f1
        fill(range(my), -1);                               b957
        rep (i, nx){                                       4ed1
            fill(range(mark), false);                      13a5
            if (augment(i)) ret++;                         cc89
        }                                                  95cf
        return ret;                                        ee0f
    }                                                      95cf
};                                                         329b
```

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

Usage:

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

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

```
c041 const int MAXN = 1024;
6ab1 struct Blossom {
0b32     vector<int> adj[MAXN];
93d2     queue<int> q;
5c83     int n;  // set n to number of vertices before use
0de2     int label[MAXN], mate[MAXN], save[MAXN], used[MAXN];
427e 
427e 
2186     void init(int nv) {
6646         n = nv;
e962         Rep (i, n) adj[i].clear();
f7e2         memset(label, 0, sizeof label);
5f6a         memset(mate, 0, sizeof mate);
c4b9         memset(save, 0, sizeof save);
ee13         memset(used, 0, sizeof used);
95cf     }
427e 
bfab     void add_edge(int u, int v) {
c71a         adj[u].push_back(v);
a717         adj[v].push_back(u);
95cf     }
427e 
2a48     void rematch(int x, int y){
8af8         int m = mate[x]; mate[x] = y;
1aa4         if (mate[m] == x) {
f4ba             if (label[x] <= n) {
e7ce                 mate[m] = label[x];
bec9                 rematch(label[x], m);
8e2e             } else {
3341                 int a = 1 + (label[x] - n - 1) / n;
2885                 int b = 1 + (label[x] - n - 1) % n;
ef33                 rematch(a, b); rematch(b, a);
95cf             }
95cf         }
95cf     }
427e 
8a50     void traverse(int x) {
43c0         Rep (i, n) save[i] = mate[i];
2ef7         rematch(x, x);
34d7         Rep (i, n) {
62c5             if (mate[i] != save[i]) used[i]++;
97ef             mate[i] = save[i];
95cf         }
95cf     }
427e 
8bf8     void relabel(int x, int y) {
d101         Rep (i, n) used[i] = 0;
c4ea         traverse(x); traverse(y);
34d7         Rep (i, n) {
dee9             if (used[i] == 1 and label[i] < 0) {
1c22                 label[i] = n + x + (y - 1) * n;
eb31                 q.push(i);
95cf             }
95cf         }
95cf     }
427e 
a0ce     int solve() {
34d7         Rep (i, n) {
a073             if (mate[i]) continue;
1fc0             Rep (j, n) label[j] = -1;
7676             label[i] = 0; q = queue<int>(); q.push(i);
1c7d             while (q.size()) {
66ba                 int x = q.front(); q.pop();
b98c                 for (int y : adj[x]) {
c07f                     if (mate[y] == 0 and i != y) {
0593                         mate[y] = x;
2b14                         rematch(x, y);
8ea8                         q = queue<int>();
6173                         break;
95cf                     }
9079                     if (label[y] >= 0) {
a72e                         relabel(x, y);
b333                         continue;
95cf                     }
58ec                     if (label[mate[y]] < 0) {
9773                         label[mate[y]] = x;
086d                         q.push(mate[y]);
```

```cpp
                    }
                }
            }
        }
        int cnt = 0;
        Rep (i, n) if (mate[i] > i) cnt++;
        return cnt;
    }
};
```

5.7 Minimum cost maximum flow

```cpp
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;
        fill(d, d + MAXN, INF); d[s] = 0;
        memset(inq, 0, sizeof(inq));
        q.push(s); inq[s] = true;
        p[s] = 0; a[s] = INT_MAX;
        while (!q.empty()){
            int u = q.front(); q.pop(); inq[u] = false;
            for (int i : G[u]) {
                edge& e = edges[i];
                if (e.cap > e.flow && d[e.to] > d[u] + e.cost){
                    d[e.to] = d[u] + e.cost;
                    p[e.to] = G[u][i];
                    a[e.to] = min(a[u], e.cap - e.flow);
                    if (!inq[e.to]) q.push(e.to), inq[e.to] = true;
                }
            }
        }
        return d[t] != INF;
    }

    void augment(){
        int u = t;
        while (u != s){
            edges[p[u]].flow += a[t];
            edges[p[u]^1].flow -= a[t];
            u = edges[p[u]].from;
        }
    }

#ifdef GIVEN_FLOW
    bool min_cost(int s, int t, int f, LL& cost) {
        this->s = s; this->t = t;
        int flow = 0;
        cost = 0;
        while (spfa()) {
            augment();
            if (flow + a[t] >= f){
                cost += (f - flow) * d[t]; flow = f;
                return true;
            } else {
                flow += a[t]; cost += a[t] * d[t];
            }
        }
        return false;
    }
#else
    int min_cost(int s, int t, LL& cost) {
        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  };
```

5.8 Global minimum cut (Stoer-Wagner)

Usage:

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

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

```
f9d7  typedef vector<LL> VI;
045e  typedef vector<VI> VVI;
427e
f012  pair<LL, VI> stoer(VVI &w) {
66f7      int n = w.size();
4d98      VI used(n), c, bestc;
329d      LL bestw = -1;
427e
cd21      for (int ph = n - 1; ph >= 0; ph--) {
ec6e          VI wt = w[0], added = used;
f20e          int prev, last = 0;
4b32          rep (i, ph) {
8bfc              prev = last;
0706              last = -1;
4942              for (int j = 1; j < n; j++)
c4b9                  if (!added[j] && (last == -1 || wt[j] > wt[last]))
887d                      last = j;
71bc              if (i == ph - 1) {
9cfa                  rep (j, n) w[prev][j] += w[last][j];
1f25                  rep (j, n) w[j][prev] = w[prev][j];
5613                  used[last] = true;
8e11                  c.push_back(last);
bb8e                  if (bestw == -1 || wt[last] < bestw) {
```

```
                      bestc = c;                                bab6
                      bestw = wt[last];                         372e
                  }                                             95cf
              } else {                                          8e2e
                  rep (j, n) wt[j] += w[last][j];               caeb
                  added[last] = true;                           8b92
              }                                                 95cf
          }                                                     95cf
      }                                                         95cf
      return {bestw, bestc};                                    038c
  }                                                             95cf
```

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
int lca(int u, int v) {                                         0f0b
    tie(u, v) = minmax(id[u], id[v]);                           cfc4
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 
0f04 void decomp(int root){
         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
}                                         95cf
                                          427e
```

```
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;

    void init(int n) { // fill the array with 0
```

```
        tr.resize(N = n + 5);
    }

    LL sum(int n) {
        LL ans = 0;
        while (n) {
            ans += tr[n];
            n &= n - 1;
        }
        return ans;
    }

    void add(int n, LL x){
        while (n < N) {
            tr[n] += x;
            n += n & -n;
        }
    }
};
```

6.2 Fenwick tree (range update point query)

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

    void init(int n) { // fill the array with 0
        tr.resize(N = n + 5);
    }

    LL query(int n) {
        LL ans = 0;
        while (n < N) {
            ans += tr[n];
            n += n & -n;
        }
        return ans;
    }

    void add(int n, LL x) {
        while (n){
            tr[n] += x;
            n &= n - 1;
        }
    }
};
```

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;
```

```
95cf      }
95cf    }
427e
471c    void build(int o, int ll, int rr) {
0e87      int mm = (ll + rr) / 2;
9d27      l[o] = ll; r[o] = rr; m[o] = mm;
ac0a      tmul[o] = 1;
5c92      if (ll == mm) {
001f        scanf("%lld", val + o);
e5b6        val[o] %= p;
8e2e      } else {
7293        build(lson, ll, mm);
5e67        build(rson, mm, rr);
ba26        pull(o);
95cf      }
95cf    }
427e
4406    void add(int o, int ll, int rr, LL x) {
3c16      if (ll <= l[o] && r[o] <= rr) {
db32        push_add(o, x);
8e2e      } else {
c4b0        push(o);
4305        if (m[o] > ll) add(lson, ll, rr, x);
d5a6        if (m[o] < rr) add(rson, ll, rr, x);
ba26        pull(o);
95cf      }
95cf    }
427e
48cd    void mul(int o, int ll, int rr, LL x) {
3c16      if (ll <= l[o] && r[o] <= rr) {
e7d0        push_mul(o, x);
8e2e      } else {
c4b0        push(o);
d1ba        if (ll < m[o]) mul(lson, ll, rr, x);
67f3        if (m[o] < rr) mul(rson, ll, rr, x);
ba26        pull(o);
95cf      }
95cf    }
427e
0f62    LL query(int o, int ll, int rr) {
3c16      if (ll <= l[o] && r[o] <= rr) {
6dfe        return val[o];
8e2e      } else {
c4b0        push(o);
```

```
        if (rr <= m[o]) return query(lson, ll, rr);           462a
        if (ll >= m[o]) return query(rson, ll, rr);           5cca
        return query(lson, ll, rr) + query(rson, ll, rr);     bbf9
      }                                                       95cf
    }                                                         95cf
} seg;                                                        4d99
```

6.4 Link/cut tree

Usage:

`pull(x)`	Collect information of subtrees.
`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.

```
// about 0.13s per 100k ops @Luogu.org                        427e
                                                              427e
namespace LCT {                                               ed4d
  const int MAXN = 300005;                                    5ece
  int fa[MAXN], ch[MAXN][2], val[MAXN], sum[MAXN];            6a6d
  bool rev[MAXN];                                             c6e1
                                                              427e
  bool isroot(int x) {                                        7839
    return ch[fa[x]][0] == x || ch[fa[x]][1] == x;            45a9
  }                                                           95cf
                                                              427e
  void pull(int x) {                                          3bf9
    sum[x] = val[x] ^ sum[ch[x][0]] ^ sum[ch[x][1]];          6664
  }                                                           95cf
                                                              427e
  void reverse(int x) {                                       3698
    swap(ch[x][0], ch[x][1]);                                 7850
    rev[x] ^= 1;                                              52c6
  }                                                           95cf
                                                              427e
  void push(int x) {                                          1a53
    if (rev[x]) {                                             8f1f
      if (ch[x][0]) reverse(ch[x][0]);                        ebf3
      if (ch[x][1]) reverse(ch[x][1]);                        6eb0
      rev[x] = 0;                                             8fc1
    }                                                         95cf
  }                                                           95cf
427e
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;
af46      ch[x][!k] = y; ch[y][k] = w;
fa6f      if (w) fa[w] = y;
3540      fa[y] = x; fa[x] = z;
72ef      pull(y);
95cf    }
427e
bc1b    void pushall(int x) {
a316      if (isroot(x)) pushall(fa[x]);
a97b      push(x);
95cf    }
427e
f69c    void splay(int x) {
d095      int y = x, z = 0;
8ab3      pushall(y);
f244      while (isroot(x)) {
ceef        y = fa[x]; z = fa[y];
4449        if (isroot(y)) rotate((ch[y][0] == x) ^ (ch[z][0] == y) ? x : y);
cf90        rotate(x);
95cf      }
78a0      pull(x);
95cf    }
427e
6229    void access(int x) {
1548      int z = x;
ba78      for (int y = 0; x; x = fa[y = x]) {
8fec        splay(x);
b05d        ch[x][1] = y;
78a0        pull(x);
95cf      }
7afd      splay(z);
95cf    }
427e
502e    void chroot(int x) {
766a      access(x);
cb0d      reverse(x);
95cf    }
427e
471a    void split(int x, int y) {
3015      chroot(x);
29b5      access(y);
95cf      }
427e
d87a      int Root(int x) {
766a        access(x);
874d        while (ch[x][0]) {
a97b          push(x);
b83a          x = ch[x][0];
95cf        }
8fec        splay(x);
d074        return x;
95cf      }
427e
70d3      void Link(int u, int v) {  // assume unconnected before
b8a5        chroot(u);
2448        fa[u] = v;
95cf      }
427e
c2f4      void Cut(int u, int v) {  // assume connected before
e8ce        split(u, v);
fd95        fa[u] = ch[v][0] = 0;
743b        pull(v);
95cf      }
427e
6ca2      int Query(int u, int v) {
e8ce        split(u, v);
a5ba        return sum[v];
95cf      }
427e
eaba      void Update(int u, int x) {
46ce        splay(u);
1d62        val[u] = x;
95cf      }
329b    };
```

6.5 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
```

6.6 Persistent segment tree, range k-th query

```
f1a7 struct node {
2ff6   static int n, pos;
427e
7cec   int value;
70e2   node *left, *right;
427e
20b0   void* operator new(size_t size);
427e
3dc0   static node* Build(int l, int r) {
b6c5     node* a = new node;
ce96     if (r > l + 1) {
181e       int mid = (l + r) / 2;
3ba2       a->left = Build(l, mid);
8aaf       a->right = Build(mid, r);
8e2e     } else {
bfc4       a->value = 0;
95cf     }
5ffd     return a;
95cf   }
427e
5a45   static node* init(int size) {
2c46     n = size;
7ee3     pos = 0;
be52     return Build(0, n);
95cf   }
427e
93c0   static int Query(node* lt, node *rt, int l, int r, int k) {
d30c     if (r == l + 1) return l;
181e     int mid = (l + r) / 2;
```

```
    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
```

6.7 Persistent block list

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

the property.

Usage:

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

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

```
a19e constexpr int BLOCK = 800;
76b3 typedef vector<int> vi;
0563 typedef shared_ptr<vi> pvi;
013b typedef shared_ptr<const vi> pcvi;
427e
a771 struct block {
2989     pcvi data;
8fd0     LL sum;
427e
427e     // add information to maintain
a613     block(pcvi ptr) :
24b5         data(ptr),
0cf0         sum(accumulate(ptr->begin(), ptr->end(), 0ll))
e93b     { }
427e
5c0f     void merge(const block& another) {
0b18         pvi temp = make_shared<vi>(data->begin(), data->end());
ac21         temp->insert(temp->end(), another.data->begin(), another.data->end());
6467         *this = block(temp);
95cf     }
427e
42e8     block split(int pos) {
dac1         block result(make_shared<vi>(data->begin() + pos, data->end()));
01db         *this = block(make_shared<vi>(data->begin(), data->begin() + pos));
56b0         return result;
95cf     }
329b };
427e
2a18 typedef list<block>::iterator lit;
427e
ce14 struct blocklist {
5540     list<block> blk;
427e
7b8e     void maintain() {
```

```
        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
```

6.8 Sparse table, range extremum query

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

```
const int MAXN = 100007;                                                             db63
int a[MAXN];                                                                         b330
int st[MAXN][32 - __builtin_clz(MAXN)];                                              69ae
                                                                                     427e
inline int ext(int x, int y){return x>y?x:y;} // ! max                               8041
```

```
427e
d34f  void init(int n){
ce01      int l = 31 - __builtin_clz(n);
cf75      rep (i, n) st[i][0] = a[i];
b811      rep (j, l)
6937          rep (i, 1+n-(1<<j))
082a              st[i][j+1] = ext(st[i][j], st[i+(1<<j)][j]);
95cf  }
427e
c863  int rmq(int l, int r){
92f5      int k = 31 - __builtin_clz(r-l+1);
baa2      return ext(st[l][k], st[r-(1<<k)+1][k]);
95cf  }
```

7 Geometrics

7.1 2D geometric template

```
302f  #include <bits/stdc++.h>
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
8d6e  bool ptOnSeg(pt& p, seg& s){
ce77      vec v1 = s.first - p, v2 = s.second - p;
de97      return (v1, v2) <= 0 && v1 * v2 == 0;
95cf  }
427e
427e  // 0 not on segment
```

```
// 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),                 2096
           j2 = rotOrder(a.first-b.first, vb, a.second-b.first);                 72fe
    if (j1 < 0 || j2 < 0) return 0;                                              5ac6
    if (j1 != 0 && j2 != 0) return 1;                                            9400
    if (j1 == 0 && j2 == 0){                                                     83db
        if (va * vb == 0) return 4; else return 3;                               6b0c
    } else return 2;                                                             fb17
```

```cpp
}

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 Lagrange's interpolation

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

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

then the Lagrange polynomial is

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