

# 南京大学 ACM-ICPC 集训队代码模版库



## Contents

### 1 General

1.1	Code library checksum	3
1.2	Makefile	3
1.3	.vimrc	3
1.4	Stack	3
1.5	Template	3

### 2 Miscellaneous Algorithms

2.1	2-SAT	4
2.2	Knuth's optimization	4
2.3	Mo's algorithm	5

### 3 String

3.1	Knuth-Morris-Pratt algorithm	5
3.2	Manacher algorithm	6
3.3	Aho-corasick automaton	6
3.4	Suffix array	7
3.5	Trie	7
3.6	Rolling hash	8

### 4 Math

4.1	Extended Euclidean algorithm and Chinese remainder theorem	8
4.2	Matrix powermod	9
4.3	Linear basis	9
4.4	Gauss elimination over finite field	9
4.5	Berlekamp-Massey algorithm	10
4.6	Fast Walsh-Hadamard transform	10
4.7	Fast fourier transform	11
4.8	Number theoretic transform	11
4.9	Sieve of Euler	12
4.10	Sieve of Euler (General)	12
4.11	Miller-Rabin primality test	13
4.12	Pollard's rho algorithm	13

### 5 Graph Theory

5.1	Strongly connected component	13
5.2	Vertex biconnected component	14
5.3	Minimum spanning arborescence (Chu-Liu)	15
5.4	Maximum flow (Dinic)	15
5.5	Maximum cardinality bipartite matching (Hungarian)	16
5.6	Maximum matching of general graph (Edmond's blossom)	17
5.7	Minimum cost maximum flow	18
5.8	Global minimum cut (Stoer-Wagner)	19
5.9	Fast LCA	20
5.10	Heavy-light decomposition	20
5.11	Centroid decomposition	21
5.12	DSU on tree	21

### 6 Data Structures

6.1	Fenwick tree (point update range query)	22
6.2	Fenwick tree (range update point query)	22
6.3	Segment tree	22
6.4	Link/cut tree	23
6.5	Balanced binary search tree from pb_ds	24
6.6	Persistent segment tree, range k-th query	24
6.7	Persistent block list	25
6.8	Sparse table, range extremum query	26

### 7 Geometries

7.1	2D geometric template	26
-----	-----------------------	----

### 8 Appendices

8.1	Primes	28
8.1.1	First primes	28
8.1.2	Arbitrary length primes	28
8.1.3	$\sim 1 \times 10^9$	28
8.1.4	$\sim 1 \times 10^{18}$	29
8.2	Pell's equation	29
8.3	Burnside's lemma and Polya's enumeration theorem	29
8.4	Lagrange's interpolation	29

## 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%\
```

```
set whichwrap=b,s,<,>[,]
```

1c67

### 1.4 Stack

```
const int STK_SZ = 2000000;
char STK[STK_SZ * sizeof(void*)];
void *STK_BAK;

#ifdef __i386__
#define SP "%esp"
#elif defined(__x86_64__)
#define SP "%rsp"
#endif

int main() {
    asm volatile("movl SP, %0; movl 1, SP: =g(STK_BAK):g(STK+sizeof(STK)):");
    ;

    // main program

    asm volatile("movl %0, SP: =g(STK_BAK)");
    return 0;
}
```

bebe  
effc  
4e99  
427e  
7bc9  
0894  
ac7a  
a9ea  
1937  
427e  
3117  
3750  
427e  
427e  
427e  
6856  
7021  
95cf

### 1.5 Template

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

#ifdef __LOCAL_DEBUG__
# define _debug(fmt, ...) fprintf(stderr, "[%s] " fmt "\n", \
    __func__, ##__VA_ARGS__)
#else
# define _debug(...) ((void) 0)
#endif
#define rep(i, n) for (int i=0; i<(n); i++)
#define Rep(i, n) for (int i=1; i<=(n); i++)
#define range(x) begin(x), end(x)
typedef long long LL;
typedef unsigned long long ULL;
```

302f  
421c  
427e  
426f  
3341  
611f  
a8cb  
e6b5  
1937  
0d6c  
cfe3  
3505  
5cad  
b773

## 2 Miscellaneous Algorithms

### 2.1 2-SAT

```

0f42 const int MAXN = 100005;
03a9 struct twoSAT{
5c83     int n;
8f72     vector<int> G[MAXN*2];
d060     bool mark[MAXN*2];
b42d     int S[MAXN*2], c;
427e
d34f     void init(int n){
b985         this->n = n;
f9ec         for (int i=0; i<n*2; i++) G[i].clear();
0609         memset(mark, 0, sizeof(mark));
95cf     }
427e
3bd5     bool dfs(int x){
bd70         if (mark[x^1]) return false;
c96a         if (mark[x]) return true;
fd23         mark[x] = true;
4bea         S[c++] = x;
1ce6         for (int i=0; i<G[x].size(); i++)
d942             if (!dfs(G[x][i])) return false;
3361         return true;
95cf     }
427e
5894     void add_clause(int x, bool xval, int y, bool yval){
6afe         x = x * 2 + xval;
e680         y = y * 2 + yval;
81cc         G[x^1].push_back(y);
6835         G[y^1].push_back(x);
95cf     }
427e
d0cb     bool solve() {
7c39         for (int i=0; i<n*2; i+=2){
e63f             if (!mark[i] && !mark[i+1]){
88fb                 c = 0;
f4b9                 if (!dfs(i)){
3f03                     while (c > 0) mark[S[--c]] = false;
86c5                     if (!dfs(i+1)) return false;
95cf                 }
95cf             }

```

```

    }
    return true;
}

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

```

95cf  
3361  
95cf  
427e  
5f0a  
329b

### 2.2 Knuth's optimization

```

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;
                }
            }
        }
    }
};

```

5c83  
d77c  
427e  
b7ec  
0bc7  
0423  
8f5e  
9488  
95cf  
be8e  
95b5  
aa0f  
95cf  
ec08  
88b8  
d3da  
9824  
a24a  
f933  
90d2  
9bd0  
26b5  
e6af  
9c88  
95cf  
95cf  
95cf  
95cf  
329b

## 2.3 Mo's algorithm

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

### Usage:

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

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

```
        yield(q.id);
    }
}
```

```
82f5
95cf
95cf
```

## 3 String

### 3.1 Knuth-Morris-Pratt algorithm

```
const int SIZE = 10005;

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

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

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

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

```
2836
427e
d02b
2d81
9847
57b7
427e
60cf
aaa1
3a87
3dd4
d8a8
147f
3c79
4643
95cf
95cf
427e
c464
427e
95cf
427e
2daf
700f
8482
8fd0
be8e
4e19
b5d5
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;
if ((r ^ 1) & 1) {
} else rr++;
return {max(1, rl), min(r, rr)};
}
};

```

17e4  
3908  
69f3  
69dc  
95cf  
329b

### 3.3 Aho-corasick automaton

```

struct AC : Trie {
    int fail[MAXN];
    int last[MAXN];

    void construct() {
        queue<int> q;
        fail[0] = 0;
        rep(c, CHARN) {
            if (int u = tr[0][c]) {
                fail[u] = 0;
                q.push(u);
                last[u] = 0;
            }
        }
        while (!q.empty()) {
            int r = q.front();
            q.pop();
            rep(c, CHARN) {
                int u = tr[r][c];
                if (!u) {
                    tr[r][c] = tr[fail[r]][c];
                    continue;
                }
                q.push(u);
                int v = fail[r];
                while (v && !tr[v][c]) v = fail[v];
                fail[u] = tr[v][c];
                last[u] = tag[fail[u]] ? fail[u] : last[fail[u]];
            }
        }
    }

    void found(int pos, int j) {

```

a1ad  
9143  
daca  
427e  
8690  
93d2  
a7a6  
ce3c  
b1c6  
a506  
3e14  
f689  
95cf  
95cf  
cc78  
31f0  
15dd  
ce3c  
ab59  
0ef5  
9d58  
b333  
95cf  
3e14  
b3ff  
d2ea  
c275  
654c  
95cf  
95cf  
95cf  
427e  
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);

```

```

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;
    }
}

```

### 3.5 Trie

```

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

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

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

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

```

```

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

```

### 3.6 Rolling hash

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

**Usage:**

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

```

1e42 const LL mod = 1006658951440146419, g = 967;
9f60 const int MAXN = 200005;
0291 LL pg[MAXN];
427e
6832 inline LL mul(LL x, LL y) {
c919     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    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;
struct linearbasis {
    ULL b[MAXD] = {};

    bool insert(LL v) {
        for (int j = MAXD - 1; j >= 0; j--) {
            if (!(v & (1ll << j))) continue;
            if (b[j]) v ^= b[j]
            else {
                for (int k = 0; k < j; k++)
                    if (v & (1ll << k)) v ^= b[k];
                for (int k = j + 1; k < MAXD; k++)
                    if (b[k] & (1ll << j)) b[k] ^= v;
                b[j] = v;
                return true;
            }
        }
        return false;
    }
};

```

## 4.4 Gauss elimination over finite field

```

const LL p = 1000000007;

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

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);

```

```

2976 LL det = 1;
427e
be8e rep (i, n) {
d2b5     int pj = -1, pk = -1;
6b4a     rep (j, n) if (!ipiv[j])
e582         rep (k, n) if (!ipiv[k])
6112             if (pj == -1 || a[j][k] > a[pj][pk]) {
a905                 pj = j;
657b                 pk = k;
95cf             }
d480     if (a[pj][pk] == 0) return 0;
0305     ipiv[pk]++;
8dad     swap(a[pj], a[pk]);
aad8     swap(b[pj], b[pk]);
be4d     if (pj != pk) det = (p - det) % p;
d080     irow[i] = pj;
f156     icol[i] = pk;
427e
4ecd     LL c = powmod(a[pk][pk], p - 2);
865b     det = det * a[pk][pk] % p;
c36a     a[pk][pk] = 1;
dd36     rep (j, n) a[pk][j] = a[pk][j] * c % p;
1b23     rep (j, m) b[pk][j] = b[pk][j] * c % p;
f8f3     rep (j, n) if (j != pk) {
e97f         c = a[j][pk];
c449         a[j][pk] = 0;
820b         rep (k, n) a[j][k] = (a[j][k] + p - a[pk][k] * c % p) % p;
f039         rep (k, m) b[j][k] = (b[j][k] + p - b[pk][k] * c % p) % p;
95cf     }
95cf }
427e
37e1 for (int j = n - 1; j >= 0; j--) if (irow[j] != icol[j]) {
50dc     for (int k = 0; k < n; k++) swap(a[k][irow[j]], a[k][icol[j]]);
95cf }
f27f return det;
95cf }

```

#### 4.5 Berlekamp-Massey algorithm

```

d790 vector<int> berlekamp(const vector<int>& a) {
4166     vector<int> p = {1}, r = {1};
baed     int dif = 1;

```

```

rep (i, a.size()) {
    int u = 0;
    rep (j, p.size())
        u = (u + 111 * 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));
        int idif = inv(dif);
        rep (j, r.size())
            p[j+1] =
                (p[j+1] - 111 * r[j] * idif % mod * u % mod + mod) % mod;
        dif = u;
        r = op;
    }
}
return p;
}

```

```

8bc9
3e58
ac8e
a488
eae9
b14c
8e2e
0c78
02f6
786b
9b57
793c
1836
644c
bc58
95cf
95cf
e149
95cf

```

#### 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
            }
}

```

```

061e
5595
05f2
b833
7796
427e
427e
427e
95cf
95cf

```

```

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
            }
}

```

```

427e
4db1
5595
05f2
b833
7796
427e
427e
427e
95cf

```

```

95cf }
427e
2ab6 void conv(int* a, int* b, int n){
950a     fwt(a, n);
e427     fwt(b, n);
8a42     rep(i, n) a[i] *= b[i];
430f     ifwt(a, n);
95cf }

```

## 4.7 Fast fourier transform

```

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

```

```

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

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

```

```

617b
a123
3b2f
57fc
95cf
427e
bdc0
6497
12a5
f84e
95cf
329b

```

## 4.8 Number theoretic transform

```

const int NMAX = 1<<21;

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

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

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

    NTT(int k){
        K = k; N = 1 << k;
        g = powmod(G, (P-1)/N);
        g_inv = powmod(g, N-1);
        omega[0] = oinv[0] = 1;
        rep (i, N){

```

```

4ab9
427e
427e
fb9a
427e
87ab
c47c
0eda
81af
9827
427e
2a2c
95a2
3e90
6624
489e
16fc
95cf
547e
95cf
427e
f420
e209
7652
4b3a
e04f
b393

```

```

7ba3         rev[i] = (rev[i>>1]>>1) | ((i&1)<<(K-1));
ad4f         if (i){
8d8b             omega[i] = omega[i-1] * g % P;
9e14             oinv[i] = oinv[i-1] * g_inv % P;
95cf         }
95cf     }
95cf }
427e
9668 void _ntt(LL* a, LL* w){
a215     rep (i, N) if (i < rev[i]) swap(a[i], a[rev[i]]);
ac6e     for (int l = 2; l <= N; l *= 2){
2969         int m = l/2;
7a1d         for (LL* p = a; p != a + N; p += l)
c24f             rep (k, m){
0ad3                 LL t = w[N/l*k] * p[k+m] % P;
6209                 p[k+m] = (p[k] - t + P) % P;
fa1b                 p[k] = (p[k] + t) % P;
95cf             }
95cf         }
95cf     }
427e
92ea void ntt(LL* a){_ntt(a, omega);}
5daf void intt(LL* a){
1f2a     LL inv = powmod(N, P-2);
9910     _ntt(a, oinv);
a873     rep (i, N) a[i] = a[i] * inv % P;
95cf }
427e
3a5b void conv(LL* a, LL* b){
ad16     ntt(a); ntt(b);
e49e     rep (i, N) a[i] = a[i] * b[i] % P;
5748     intt(a);
95cf }
329b };

```

## 4.9 Sieve of Euler

```

cfc3 const int MAXX = 1e7+5;
5861 bool p[MAXX];
73ae int prime[MAXX], sz;
427e
9bc6 void sieve(){

```

```

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

```

```

9628
1ec8
bf28
e82c
b6a9
5f51
95cf
95cf
95cf

```

## 4.10 Sieve of Euler (General)

```

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

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

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

    for (int i = 2; i < N; i++) {
        if (!p[i]) {
            prime[sz++] = i;
            for (LL j = i; j < N; j *= i) {
                int b = j / i;
                pval[j] = i * pval[b];
                pcnt[j] = pcnt[b] + 1;
                f[j] = _____; // f[j] = f(i^pcnt[j])
            }
        }
        for (int j = 0; i * prime[j] < N; j++) {
            int x = i * prime[j]; p[x] = 1;
            if (i % prime[j] == 0) {
                pval[x] = pval[i] * prime[j];
                pcnt[x] = pcnt[i] + 1;
            } else {

```

```

b62e
6589
e982
6ae8
cbf7
6030
427e
76f6
9628
427e
8a8a
bdda
c6b9
427e
a643
01d6
b2b2
37d9
758c
81fd
e0f3
a96c
95cf
95cf
34c0
f87a
20cc
9985
3f93
8e2e

```

```

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     }
95cf     }

```

#### 4.11 Miller-Rabin primality test

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

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

```

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

```

#### 4.12 Pollard's rho algorithm

```

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;
}

```

2e6b  
427e  
54a5  
45eb  
d3e5  
3c69  
0964  
4753  
5952  
9e5b  
33d5  
e1bf  
e1bf  
a313  
95cf  
95cf  
5d89  
95cf

## 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]){

```

837c  
427e  
2ea0  
88e3  
9cad  
3d02  
8b6c  
27ee  
427e  
bfab  
c71a  
95cf  
427e  
d714  
7e41  
80f6  
18f6  
173e

```

5f3c         dfs(v);
002c         lnk[u] = min(lnk[u], lnk[v]);
6068     } else if (!scc[v]){
d5df         lnk[u] = min(lnk[u], pre[v]);
95cf     }
95cf     }
8de2     if (lnk[u] == pre[u]){
660f         sccn++;
3c9e         int x;
a69f         do {
3834             x = s.top(); s.pop();
b0e9             scc[x] = sccn;
6757         } while (x != u);
95cf     }
95cf }
427e
4c88 void find_scc(){
f4a2     time = sccn = 0;
8de7     memset(scc, 0, sizeof scc);
8c2f     memset(pre, 0, sizeof pre);
6901     Rep (i, V){
56d1         if (!pre[i]) dfs(i);
95cf     }
95cf }
427e
27ce vector<int> adjc[MAXV];
364d void contract(){
1a1e     Rep (i, V)
21a2         rep (j, adj[i].size()){
b730             if (scc[i] != scc[adj[i][j]])
b46e                 adjc[scc[i]].push_back(scc[adj[i][j]]);
95cf         }
95cf     }
329b };

```

## 5.2 Vertex biconnected component

```

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];

```

```

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;
}

```

427e  
76f7  
427e  
bfab  
c71a  
a717  
95cf  
427e  
7d3c  
9fe6  
ec14  
18f6  
173e  
e7f8  
fdcf  
f851  
189c  
b687  
6323  
57eb  
90b8  
a147  
a6a3  
a0c3  
0ef5  
3db2  
e0db  
d27f  
95cf  
f357  
752b  
57c9  
95cf  
7096  
95cf  
03f5  
95cf  
7470  
e7f8  
f115  
95cf  
95cf  
e104  
1160

```

95cf    }
427e
17be    void find_bcc(int n) {
8c2f        memset(pre, 0, sizeof pre);
e2d2        memset(iscut, 0, sizeof iscut);
40d3        memset(bccno, -1, sizeof bccno);
fae2        dfs_clock = bcc_cnt = 0;
5c63        rep (i, n) if (!pre[i]) dfs(i, -1);
95cf    }
329b    };

```

### 5.3 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:**  $O(|V||E|)$

```

bcf8    struct edge {
54f1        int u, v;
309c        LL w;
329b    };
427e
f5a4    const int MAXN = 10005;
7124    LL in[MAXN];
1c1d    int pre[MAXN], vis[MAXN], id[MAXN];
427e
5a43    LL getans(int n, int rt, vector<edge>& edges) {
f7ff        LL ans = 0;
8abb        int cnt = 0;
a147        while (1) {
641a            Rep (i, n) in[i] = LLONG_MAX, id[i] = vis[i] = 0;
0705            for (auto e : edges) {
073a                if (e.u != e.v and e.w < in[e.v]) {
c1df                    pre[e.v] = e.u;
5fbc                    in[e.v] = e.w;
95cf                }
95cf            }
3fdb            in[rt] = 0;
34d7            Rep (i, n) {
3c97                if (in[i] == LLONG_MAX) return -1;
cf57                ans += in[i];

```

```

        int u;
        for (u = i; u != rt && vis[u] != i && !id[u]; u = pre[u])
            vis[u] = i;
        if (u != rt && !id[u]) {
            id[u] = ++cnt;
            for (int v = pre[u]; v != u; v = pre[v])
                id[v] = cnt;
        }
    }
    if (!cnt) return ans;
    Rep (i, n) if (!id[i]) id[i] = ++cnt;
    for (auto& e : edges) {
        LL laz = in[e.v];
        e.u = id[e.u];
        e.v = id[e.v];
        if (e.u != e.v) e.w -= laz;
    }
    n = cnt; rt = id[rt]; cnt = 0;
}

```

a763  
4b0e  
88a2  
4b22  
b66e  
0443  
5c22  
95cf  
95cf  
91e9  
5e22  
7400  
7750  
97ae  
fae6  
bdd2  
95cf  
6cc4  
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,  $O(\sqrt{VE})$ .

```

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

const int MAXN = 10005;
struct Dinic {
    int n, m, s, t;
    vector<edge> edges;
    vector<int> G[MAXN];
    bool vis[MAXN];
    int d[MAXN];
    int cur[MAXN];

```

bcf8  
60e2  
5e6d  
329b  
427e  
e2cd  
9062  
4dbf  
9f0c  
b891  
bbb6  
b40a  
ddec  
427e

```

5973 void add_edge(int from, int to, LL cap) {
7b55     edges.push_back(edge{from, to, cap, 0});
1db7     edges.push_back(edge{to, from, 0, 0});
fe77     m = edges.size();
dff5     G[from].push_back(m-2);
8f2d     G[to].push_back(m-1);
95cf }
427e
1836 bool bfs() {
3b73     memset(vis, 0, sizeof(vis));
93d2     queue<int> q;
5d13     q.push(s);
2cd2     vis[s] = 1;
721d     d[s] = 0;
cc78     while (!q.empty()) {
66ba         int x = q.front(); q.pop();
3b61         for (int i = 0; i < G[x].size(); i++) {
b510             edge& e = edges[G[x][i]];
bba9             if (!vis[e.to] && e.cap > e.flow) {
cd72                 vis[e.to] = 1;
cf26                 d[e.to] = d[x] + 1;
ca93                 q.push(e.to);
95cf             }
95cf         }
95cf     }
b23b     return vis[t];
95cf }
427e
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 }

```

```

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

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

```

427e  
5bf2  
590d  
62e2  
ed58  
f326  
fb3a  
95cf  
84fb  
95cf  
427e  
c72e  
1df9  
df9a  
56d8  
46a2  
95cf  
4206  
95cf  
329b

## 5.5 Maximum cardinality bipartite matching (Hungarian)

```

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

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

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

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

```

302f  
421c  
427e  
0d6c  
cfe3  
8843  
5cad  
427e  
84ee  
fbf6  
9ec6  
9d4c  
edec  
427e  
8324  
c1d1  
f9c1  
ac92  
3f11



```

1023     mark.resize(nx);
95cf   }
427e
4589   inline void add(int a, int b){
486c       e[a].push_back(b);
95cf   }
427e
0c2b   bool augment(int i){
207c       if (!mark[i]) {
dae4           mark[i] = true;
6a1e           for (int j : e[i]){
0892               if (my[j] == -1 || augment(my[j])){
9ca3                   mx[i] = j; my[j] = i;
3361                   return true;
95cf               }
95cf           }
95cf       }
438e       return false;
95cf   }
427e
3fac   int match(){
5b57       int ret = 0;
b0f1       fill(range(mx), -1);
b957       fill(range(my), -1);
4ed1       rep (i, nx){
13a5           fill(range(mark), false);
cc89           if (augment(i)) ret++;
95cf       }
ee0f       return ret;
95cf   }
329b };

```

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

### Usage:

<code>init(n)</code>	Initialize the template with $n$ vertices, numbered from 1.
<code>add_edge(u, v)</code>	Add an undirected edge $uv$ .
<code>solve()</code>	Find the maximum matching. Return the number of matched edges.
<code>mate[]</code>	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; // set n to number of vertices before use
    int label[MAXN], mate[MAXN], save[MAXN], used[MAXN];

    void init(int nv) {
        n = nv;
        Rep (i, n) adj[i].clear();
        memset(label, 0, sizeof label);
        memset(mate, 0, sizeof mate);
        memset(save, 0, sizeof save);
        memset(used, 0, sizeof used);
    }

    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];
        }
    }
};

```

c041  
6ab1  
0b32  
93d2  
5c83  
0de2  
427e  
427e  
2186  
6646  
e962  
f7e2  
5f6a  
c4b9  
ee13  
95cf  
427e  
bfab  
c71a  
a717  
95cf  
427e  
2a48  
8af8  
1aa4  
f4ba  
e7ce  
bec9  
8e2e  
3341  
2885  
ef33  
95cf  
95cf  
95cf  
427e  
8a50  
43c0  
2ef7  
34d7  
62c5  
97ef  
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]);
95cf                    }
95cf                }
95cf            }
95cf        }
8abb        int cnt = 0;
c816        Rep (i, n) if (mate[i] > i) cnt++;
6808        return cnt;
95cf    }
329b    };

```

## 5.7 Minimum cost maximum flow

```

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

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

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

    bool spfa(){
        queue<int> q;
        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;
                }
            }
        }
    }
};

```

```

bcf8
60e2
d698
32cc
329b
427e
cc3e
2aa8
c6cb
9ceb
9f0c
b891
f74f
8f67
9524
b330
427e
f7f2
24f0
95f0
fe77
dff5
8f2d
95cf
427e
3c52
93d2
8494
fd48
5e7c
2dae
cc78
b0aa
3bba
56d8
3601
55bc
0bea
8249
e5d3
95cf

```

```

95cf      }
95cf      }
6d7c      return d[t] != INF;
95cf      }
427e
71a4      void augment(){
06f1          int u = t;
b19d          while (u != s){
db09              edges[p[u]].flow += a[t];
25a9              edges[p[u]^1].flow -= a[t];
e6c9              u = edges[p[u]].from;
95cf          }
95cf      }
427e
6e20      #ifndef GIVEN_FLOW
5972          bool min_cost(int s, int t, int f, LL& cost) {
590d              this->s = s; this->t = t;
21d4              int flow = 0;
23cb              cost = 0;
22dc              while (spfa()) {
bcd8                  augment();
a671                  if (flow + a[t] >= f){
b14d                      cost += (f - flow) * d[t]; flow = f;
3361                      return true;
8e2e                  } else {
2a83                      flow += a[t]; cost += a[t] * d[t];
95cf                  }
95cf              }
438e              return false;
95cf          }
a8cb      #else
f9a9          int min_cost(int s, int t, LL& cost) {
590d              this->s = s; this->t = t;
21d4              int flow = 0;
23cb              cost = 0;
22dc              while (spfa()) {
bcd8                  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** adjacent matrix w (0-based). Return the capacity of the cut and the indices of one part of the cut.

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

```

typedef vector<LL> VI;
typedef vector<VI> WVI;

```

```

pair<LL, VI> stoer(WVI &w) {
    int n = w.size();
    VI used(n), c, bestc;
    LL bestw = -1;

    for (int ph = n - 1; ph >= 0; ph--) {
        VI wt = w[0], added = used;
        int prev, last = 0;
        rep (i, ph) {
            prev = last;
            last = -1;
            for (int j = 1; j < n; j++)
                if (!added[j] && (last == -1 || wt[j] > wt[last]))
                    last = j;
            if (i == ph - 1) {
                rep (j, n) w[prev][j] += w[last][j];
                rep (j, n) w[j][prev] = w[prev][j];
                used[last] = true;
                c.push_back(last);
                if (bestw == -1 || wt[last] < bestw) {
                    bestc = c;
                    bestw = wt[last];
                }
            } else {
                rep (j, n) wt[j] += w[last][j];
                added[last] = true;
            }
        }
    }
    return {bestw, bestc};
}

```

f9d7  
045e  
427e  
f012  
66f7  
4d98  
329d  
427e  
cd21  
ec6e  
f20e  
4b32  
8bfc  
0706  
4942  
c4b9  
887d  
71bc  
9cfa  
1f25  
5613  
8e11  
bb8e  
bab6  
372e  
95cf  
8e2e  
caeb  
8b92  
95cf  
95cf  
95cf  
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  $v$ .

```
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
3d1b void preprocess(int root) {
3269     nid = 0;
91e1     dfs(root, 0, 1);
5e98     int l = 31 - __builtin_clz(nid);
213b     rep (j, l) rep (i, 1+nid-(1<<j))
1131         st[i][j+1] = min(st[i][j], st[i+(1<<j)][j]);
95cf }
427e
0f0b int lca(int u, int v) {
cfc4     tie(u, v) = minmax(id[u], id[v]);
be9b     int k = 31 - __builtin_clz(v-u+1);
8ebc     return min(st[u][k], st[v-(1<<k)+1][k]).second;
95cf }
```

## 5.10 Heavy-light decomposition

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

```
0f42 const int MAXN = 100005;
0b32 vector<int> adj[MAXN];
42f2 int sz[MAXN], top[MAXN], fa[MAXN], son[MAXN], depth[MAXN], id[MAXN];
427e
```

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

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

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

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

```
be5c
7489
2ee7
adb4
b79d
c861
fe45
fd2f
b790
f0f1
c749
fe19
95cf
95cf
0e08
95cf
427e
ba54
3644
8d96
d314
c4a1
c861
9881
5518
13f9
95cf
95cf
427e
0f04
9fa4
1c88
95cf
427e
2c98
03a1
45ec
427e
005b
95cf
6083
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.

```

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

```

}

95cf

## 5.12 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 */ };

```

```

1fb6
901d
427e
5559
50c0
18f6
bd87
a851
8449
d28c
95cf
95cf
427e
b7ec
62f5
4412
30b3
95cf
427e
7467
33ff
72a2
427e
427e
427e
9866
427e
18a9

```

```

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

## 6 Data Structures

### 6.1 Fenwick tree (point update range query)

```

9976 struct bit_purq { // point update, range query
d7af     int N;
99ff     vector<LL> tr;
427e
d34f     void init(int n) { // fill the array with 0
1010         tr.resize(N = n + 5);
95cf     }
427e
63d0     LL sum(int n) {
f7ff         LL ans = 0;
e290         while (n) {
0715             ans += tr[n];
c0d4             n &= n - 1;
95cf         }
4206         return ans;
95cf     }
427e
f4bd     void add(int n, LL x){
ad20         while (n < N) {
6c81             tr[n] += x;
```

```

        n += n & -n;
    }
};
```

```

0af5
95cf
95cf
329b
```

### 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;
        }
    }
};
```

```

3d03
d7af
99ff
427e
d34f
1010
95cf
427e
38d4
f7ff
ad20
0715
0af5
95cf
4206
95cf
427e
f4bd
e290
6c81
c0d4
95cf
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)
```

```

3942
1ebb
451a
27be
4510
427e
ac35
```

```

1294 #define rson (o<<1|1)
427e
1344 void pull(int o) {
bbe9     val[o] = (val[lson] + val[rson]) % p;
95cf }
427e
e4bc void push_add(int o, LL x) {
5dd6     val[o] = (val[o] + x * (r[o] - l[o])) % p;
6eff     tadd[o] = (tadd[o] + x) % p;
95cf }
427e
d658 void push_mul(int o, LL x) {
b82c     val[o] = val[o] * x % p;
aa86     tadd[o] = tadd[o] * x % p;
649f     tmul[o] = tmul[o] * x % p;
95cf }
427e
b149 void push(int o) {
3159     if (l[o] == m[o]) return;
0a90     if (tmul[o] != 1) {
0f4a         push_mul(lson, tmul[o]);
045e         push_mul(rson, tmul[o]);
ac0a         tmul[o] = 1;
95cf     }
1b82     if (tadd[o]) {
9547         push_add(lson, tadd[o]);
0e73         push_add(rson, tadd[o]);
6234         tadd[o] = 0;
95cf     }
95cf }
427e
471c void build(int o, int ll, int rr) {
0e87     int mm = (ll + rr) / 2;
9d27     l[o] = ll; r[o] = rr; m[o] = mm;
ac0a     tmul[o] = 1;
5c92     if (ll == rr) {
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 }

```

```

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;

```

```

427e
4406
3c16
db32
8e2e
c4b0
4305
d5a6
ba26
95cf
95cf
427e
48cd
3c16
e7d0
8e2e
c4b0
d1ba
67f3
ba26
95cf
95cf
427e
0f62
3c16
6dfe
8e2e
c4b0
462a
5cca
bbf9
95cf
95cf
4d99

```

## 6.4 Link/cut tree

Dynamic tree supports single-vertex update, path aggregation and relative LCA query.

**Usage:**

<code>pull(x)</code>	Collect information of subtrees.
<code>Root(u)</code>	Get the root of tree where vertex $u$ is in.
<code>Link(u, v)</code>	Link two unconnected trees.
<code>Cut(u, v)</code>	Cut an existent edge.
<code>Query(u, v)</code>	Path aggregation.
<code>Update(u, x)</code>	Single point modification.
<code>LCA(u, v, root)</code>	Get the lowest common ancestor of $u$ and $v$ in tree rooted at root.

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

```

2e73 const int MAXN = 1000005;
ca06 struct LCT {
6a6d     int fa[MAXN], ch[MAXN][2], val[MAXN], sum[MAXN];
c6e1     bool rev[MAXN];

427e
eba3     bool isroot(int x) { return ch[fa[x]][0] == x || ch[fa[x]][1] == x; }
f19f     void pull(int x) { sum[x] = val[x] ^ sum[ch[x][0]] ^ sum[ch[x][1]]; }
1c4d     void reverse(int x) { swap(ch[x][0], ch[x][1]); rev[x] ^= 1; }
1a53     void push(int x) {
89a0         if (rev[x]) rep (i, 2) if (ch[x][i]) reverse(ch[x][i]); rev[x] = 0;
95cf     }

425f     void rotate(int x) {
51af         int y = fa[x], z = fa[y], k = ch[y][1] == x, w = ch[x][!k];
e1fe         if (isroot(y)) ch[z][ch[z][1] == y] = x;
1e6f         ch[x][!k] = y; ch[y][k] = w; if (w) fa[w] = y;
6d09         fa[y] = x; fa[x] = z; pull(y);
95cf     }

52c6     void pushall(int x) { if (isroot(x)) pushall(fa[x]); push(x); }
f69c     void splay(int x) {
d095         int y = x, z = 0;
c494         for (pushall(y); isroot(x); rotate(x)) {
ceef             y = fa[x]; z = fa[y];
4449             if (isroot(y)) rotate((ch[y][0] == x) ^ (ch[z][0] == y) ? x : y);
95cf         }
78a0         pull(x);
95cf     }

6229     void access(int x) {
1548         int z = x;
8854         for (int y = 0; x; x = fa[y = x]) { splay(x); ch[x][1] = y; pull(x); }
7afd         splay(z);
95cf     }

a067     void chroot(int x) { access(x); reverse(x); }
126d     void split(int x, int y) { chroot(x); access(y); }
427e

```

```

int Root(int x) {
    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);
    for (int rt = fa[y]; rt; rt = fa[y]) splay(y = rt);
    return y;
}
};

```

## 6.5 Balanced binary search tree from pb\_ds

```

#include <ext/pb_ds/assoc_container.hpp>
using namespace __gnu_pbds;

tree<int, null_type, less<int>, rb_tree_tag, tree_order_statistics_node_update>
rkt;
// null_tree_node_update

// SAMPLE USAGE
rkt.insert(x);           // insert element
rkt.erase(x);           // erase element
rkt.order_of_key(x);     // obtain the number of elements less than x
rkt.find_by_order(i);    // iterator to i-th (numbered from 0) smallest element
rkt.lower_bound(x);
rkt.upper_bound(x);
rkt.join(rkt2);          // merge tree (only if their ranges do not intersect)
rkt.split(x, rkt2);      // split all elements greater than x to rkt2

```

## 6.6 Persistent segment tree, range k-th query

```

struct node {
    static int n, pos;

    int value;
    node *left, *right;
}

```



```

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

```

```

    }
    a->value++;
    return a;
}

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

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

```

```

95cf
2b3e
5ffd
95cf
427e
e80f
c246
95cf
865a
427e
99ce
1987
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:

<code>maintain()</code>	Maintain the block list property.
<code>split(pos)</code>	Split the block list at position <code>pos</code> . Returns an iterator to a block starting at <code>pos</code> .
<code>sum(l, r)</code>	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) :

```

```

a19e
76b3
0563
013b
427e
a771
2989
8fd0
427e
427e
a613

```

```

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() {
3131             lit it = blk.begin();
5e44             while (it != blk.end() and next(it) != blk.end()) {
852d                 lit it2 = it;
0b03                 while (next(it2) != blk.end() and
029f                     it2->data->size() + next(it2)->data->size() <= BLOCK) {
93e1                     it2->merge(*next(it2));
e1fa                     blk.erase(next(it2));
95cf                 }
5771                 ++it;
95cf             }
427e
b7b3     lit split(int pos) {
2273         for (lit it = blk.begin(); ; it++) {
5502             if (pos == 0) return it;
d480             while (it->data->size() > pos) {
2099                 blk.insert(next(it), it->split(pos));
95cf             }
a1c8             pos -= it->data->size();
95cf         }
95cf     }

```

```

LL sum(int l, int r) { // traverse
    lit it1 = split(l), it2 = split(r);
    LL res = 0;
    while (it1 != it2) {
        res += it1->sum;
        it1++;
    }
    maintain();
    return res;
}
};

```

```

427e
fd38
48b4
ac09
9f1d
8284
61fd
95cf
b204
244d
95cf
329b

```

## 6.8 Sparse table, range extremum query

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

```

const int MAXN = 100007;
int a[MAXN];
int st[MAXN][32 - __builtin_clz(MAXN)];

inline int ext(int x, int y){return x>y?x:y;} // ! max

void init(int n){
    int l = 31 - __builtin_clz(n);
    rep (i, n) st[i][0] = a[i];
    rep (j, l)
        rep (i, 1+n-(1<<j))
            st[i][j+1] = ext(st[i][j], st[i+(1<<j)][j]);
}

int rmq(int l, int r){
    int k = 31 - __builtin_clz(r-l+1);
    return ext(st[l][k], st[r-(1<<k)+1][k]);
}

```

```

db63
b330
69ae
427e
8041
427e
d34f
ce01
cf75
b811
6937
082a
95cf
427e
c863
92f5
baa2
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
427e // 1 on segment except vertices
427e // 2 on vertices
8421 int ptOnSeg2(pt& p, seg& s){
ce77     vec v1 = s.first - p, v2 = s.second - p;
70ca     T ip = (v1, v2);
8b14     if (v1 * v2 != 0 || ip > 0) return 0;
0847     return (v1, v2) ? 1 : 2;
95cf }
427e
427e // if two orthogonal rectangles do not touch, return true
72bb inline bool nIntRectRect(seg a, seg b){
f9ac     return min(a.first.x, a.second.x) > max(b.first.x, b.second.x) ||
f486         min(a.first.y, a.second.y) > max(b.first.y, b.second.y) ||
39ce         min(b.first.x, b.second.x) > max(a.first.x, a.second.x) ||
80c7         min(b.first.y, b.second.y) > max(a.first.y, a.second.y);
95cf }
427e
427e // >0 in order
427e // <0 out of order
427e // =0 not standard

```

```

inline double rotOrder(vec a, vec b, vec c){return double(a*b)*(b*c);}
427e
inline bool intersect(seg a, seg b){
427e     // ! if (nIntRectRect(a, b)) return false; // if commented, assume that a
    and b are non-collinear
    return rotOrder(b.first-a.first, a.second-a.first, b.second-a.first) >= 0 &&
    rotOrder(a.first-b.first, b.second-b.first, a.second-b.first) >= 0;
    }
427e
    // 0 not intersect
    // 1 standard intersection
    // 2 vertex-Line intersection
    // 3 vertex-vertex intersection
    // 4 collinear and have common point(s)
    int intersect2(seg& a, seg& b){
    if (nIntRectRect(a, b)) return 0;
    vec va = a.second - a.first, vb = b.second - b.first;
    double j1 = rotOrder(b.first-a.first, va, b.second-a.first),
    j2 = rotOrder(a.first-b.first, vb, a.second-b.first);
    if (j1 < 0 || j2 < 0) return 0;
    if (j1 != 0 && j2 != 0) return 1;
    if (j1 == 0 && j2 == 0){
    if (va * vb == 0) return 4; else return 3;
    } else return 2;
    }
427e
    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));
    }
427e
    // -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;
    }
    }

```

7538  
427e  
31ed  
427e  
cb52  
059e  
95cf  
427e  
427e  
427e  
427e  
427e  
4d19  
5dc4  
42c0  
2096  
72fe  
5ac6  
9400  
83db  
6b0c  
fb17  
95cf  
427e  
2c68  
5894  
6850  
7c9a  
95cf  
427e  
427e  
427e  
427e  
cbdd  
5fb4  
1294  
427e  
3cae  
b957  
8c40  
3c4d  
aad3

```

95cf     }
0a5f     return wn ? 1 : -1;
95cf }
427e
d4a3 istream& operator >> (istream& lhs, pt& rhs){
fa86     lhs >> rhs.x >> rhs.y;
331a     return lhs;
95cf }
427e
07ae istream& operator >> (istream& lhs, seg& rhs){
5cab     lhs >> rhs.first >> rhs.second;
331a     return lhs;
95cf }

```

### 8.1.2 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 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

#### 8.1.3 $\sim 1 \times 10^9$

$p$	$g(p)$	$p$	$g(p)$	$p$	$g(p)$
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  $(x_0, y_0), \dots, (x_k, y_k)$ , define

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

then the Lagrange polynomial is

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