

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	Matrix powermod	8
4.2	Linear basis	9
4.3	Gauss elimination over finite field	9
4.4	Berlekamp-Massey algorithm	10
4.5	Fast Walsh-Hadamard transform	11
4.6	Fast fourier transform	11
4.7	Number theoretic transform	12
4.8	Sieve of Euler	12
4.9	Sieve of Euler (General)	12
4.10	Miller-Rabin primality test	13
4.11	Pollard's rho algorithm	13
4.12	Qusai-polynomial sum	14

5 Graph Theory

5.1	Strongly connected component	15
5.2	Vertex biconnected component	15
5.3	Minimum spanning arborescence (Chu-Liu)	16
5.4	Maximum flow (Dinic)	17
5.5	Maximum cardinality bipartite matching (Hungarian)	18
5.6	Minimum cost maximum flow	18
5.7	Global minimum cut (Stoer-Wagner)	19
5.8	Heavy-light decomposition	20
5.9	Centroid decomposition	20
5.10	DSU on tree	21

6 Data Structures

6.1	Fenwick tree (point update range query)	21
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	25
6.7	Persistent block list	25
6.8	Sparse table, range extremum query	26

7 Geometrics

7.1	2D geometric template	27
-----	-----------------------	----

8 Appendices

8.1	Primes	28
8.1.1	First primes	28
8.1.2	Arbitrary length primes	28
8.1.3	$\sim 1 \times 10^9$	29
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 r1 = rmid - rad, rr = rmid + rad - 1;
    if ((r ^ 1) & 1) {
    } else rr++;
    return {max(l, r1), 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

Usage:

s[]	the source string
sa[i]	the index of starting position of <i>i</i> -th suffix
rk[i]	the number of suffixes less than the suffix starting from <i>i</i>
h[i]	LCP(S[SA[i]], S[SA[i-1]])
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);
afb6     iota(y, y + n, 0);
7b42     radix_sort(rk, y, sa, n, m);
8c2     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;
    }

    // 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;
95cf }
427e
599a 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 Matrix powermod

```

const int MAXN = 105;
const LL modular = 1000000007;
int n; // order of matrices

```

```

struct matrix{
    LL m[MAXN][MAXN];

    void operator *=(matrix& a){
        static LL t[MAXN][MAXN];
        Rep (i, n){
            Rep (j, n){
                t[i][j] = 0;
                Rep (k, n){
                    t[i][j] += (m[i][k] * a.m[k][j]) % modular;
                    t[i][j] %= modular;
                }
            }
        }
        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 }

```

4.2 Linear basis

```

8b44 const int MAXD = 30;
03a6 struct linearbasis {
3558     ULL b[MAXD] = {};
427e
842f     bool insert(1l v) {
9b2b         for (int j = MAXD - 1; j >= 0; j--) {
de36             if (!(v & (1ll << j))) continue;
ee78             if (b[j]) v ^= b[j]
037f             else {
7836                 for (int k = 0; k < j; k++)
f0b4                     if (v & (1ll << k)) v ^= b[k];
b0aa                 for (int k = j + 1; k < MAXD; k++)
46c9                     if (b[k] & (1ll << j)) b[k] ^= v;
8295                 b[j] = v;
3361                 return true;
95cf             }
95cf         }
438e         return false;
95cf     }
329b };

```

4.3 Gauss elimination over finite field

```

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);
    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];

```

b784
427e
2a2c
95a2
3e90
1783
5549
16fc
95cf
547e
95cf
427e
c130
42ac
427e
2c62
561b
a25e
2976
427e
be8e
d2b5
6b4a
e582
6112
a905
657b
95cf
d480
0305
8dad
aad8
be4d
d080
f156
427e
4ecd
865b
c36a
dd36
1b23
f8f3
e97f

```

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   }

```

4.4 Berlekamp-Massey algorithm

```

2b86   const LL MOD = 1000000007;
427e
391d   LL inverse(LL b) {
32d3       LL e = MOD - 2, r = 1;
3e90       while (e) {
9a62           if (e & 1) r = r * b % MOD;
29ea           b = b * b % MOD;
16fc           e >>= 1;
95cf       }
547e       return r;
95cf   }
427e
32a6   struct Poly {
afe0       vector<int> a;
427e
9794       Poly() { a.clear(); }
427e
de81       Poly(vector<int> &a) : a(a) {}
427e
8087       int length() const { return a.size(); }
427e
16de       Poly move(int d) {
b31d           vector<int> na(d, 0);
f915           na.insert(na.end(), a.begin(), a.end());
cecf           return Poly(na);
95cf       }
427e
fa1a       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);
            }
        }
    }
}

```

5b57
501c
5de5
3041
95cf
95cf
ee0f
95cf
427e
c856
bd55
d1a7
3507
2bee
9526
95cf
cecf
95cf
329b
427e
5473
72de
d1a7
bf0c
95cf
aaab
95cf
427e
afff
9f23
58d0
4e8f
c2aa
4158
d503
c29d
db9d
6bce
1d0e
0889
64f1
8e2e
714e
95cf

```

95cf     }
95cf     }
427e     // Caution: s.a might be shorter than expected
e235     return s.a;
95cf     }

```

4.5 Fast Walsh-Hadamard transform

```

061e void fwt(int* a, int n){
5595     for (int d = 1; d < n; d <= 1)
05f2         for (int i = 0; i < n; i += d << 1)
b833             rep (j, d){
7796                 int x = a[i+j], y = a[i+j+d];
427e                 // a[i+j] = x+y, a[i+j+d] = x-y; // xor
427e                 // a[i+j] = x+y; // and
427e                 // a[i+j+d] = x+y; // or
95cf             }
95cf     }
427e
4db1 void ifwt(int* a, int n){
5595     for (int d = 1; d < n; d <= 1)
05f2         for (int i = 0; i < n; i += d << 1)
b833             rep (j, d){
7796                 int x = a[i+j], y = a[i+j+d];
427e                 // a[i+j] = (x+y)/2, a[i+j+d] = (x-y)/2; // xor
427e                 // a[i+j] = x-y; // and
427e                 // a[i+j+d] = y-x; // or
95cf             }
95cf     }
427e
2ab6 void conv(int* a, int* b, int n){
950a     fwt(a, n);
e427     fwt(b, n);
8a42     rep(i, n) a[i] *= b[i];
430f     ifwt(a, n);
95cf }

```

4.6 Fast fourier transform

```

4e09 const int NMAX = 1<<20;
427e

```

```

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){
        rep (i, N) if (i < rev[i]) swap(a[i], a[rev[i]]);
        for (int l = 2; l <= N; l *= 2){
            int m = l/2;
            for (cplx* p = a; p != a + N; p += l)
                rep (k, m){
                    cplx t = w[N/l*k] * p[k+m];
                    p[k+m] = p[k] - t; p[k] += t;
                }
        }
    }

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

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

```

```

3fbf
427e
abd1
12af
c47c
27d7
9827
427e
1442
e209
b393
7ba3
1908
a166
95cf
95cf
427e
b941
a215
ac6e
2969
b3cf
c24f
fe06
ecbf
95cf
95cf
95cf
427e
617b
a123
3b2f
57fc
95cf
427e
bdc0
6497
12a5
f84e
95cf
329b

```

4.7 Number theoretic transform

```

4ab9 const int NMAX = 1<<21;
427e
427e // 998244353 = 7*17*2^23+1, G = 3
fb9a const int P = 1004535809, G = 3; // = 479*2^21+1
427e
87ab struct NTT{
c47c     int rev[NMAX];
0eda     LL omega[NMAX], oinv[NMAX];
81af     int g, g_inv; // g:  $g_n = G^{(P-1)/n}$ 
9827     int K, N;
427e
2a2c     LL powmod(LL b, LL e){
95a2         LL r = 1;
3e90         while (e){
6624             if (e&1) r = r * b % P;
489e             b = b * b % P;
16fc             e >>= 1;
95cf         }
547e         return r;
95cf     }
427e
f420     NTT(int k){
e209         K = k; N = 1 << k;
7652         g = powmod(G, (P-1)/N);
4b3a         g_inv = powmod(g, N-1);
e04f         omega[0] = oinv[0] = 1;
b393         rep (i, N){
7ba3             rev[i] = (rev[i>>1]>>1) | ((i&1)<<(K-1));
ad4f             if (i){
8d8b                 omega[i] = omega[i-1] * g % P;
9e14                 oinv[i] = oinv[i-1] * g_inv % P;
95cf             }
95cf         }
95cf     }
427e
9668     void _ntt(LL* a, LL* w){
a215         rep (i, N) if (i < rev[i]) swap(a[i], a[rev[i]]);
ac6e         for (int l = 2; l <= N; l *= 2){
2969             int m = l/2;
7a1d             for (LL* p = a; p != a + N; p += l)
c24f                 rep (k, m){

```

```

LL t = w[N/l*k] * p[k+m] % P;
p[k+m] = (p[k] - t + P) % P;
p[k] = (p[k] + t) % P;
        }
    }
}

void ntt(LL* a){_ntt(a, omega);}
void intt(LL* a){
    LL inv = powmod(N, P-2);
    _ntt(a, oinv);
    rep (i, N) a[i] = a[i] * inv % P;
}

void conv(LL* a, LL* b){
    ntt(a); ntt(b);
    rep (i, N) a[i] = a[i] * b[i] % P;
    intt(a);
}
};

```

0ad3
6209
fa1b
95cf
95cf
95cf
427e
92ea
5daf
1f2a
9910
a873
95cf
427e
3a5b
ad16
e49e
5748
95cf
329b

4.8 Sieve of Euler

```

const int MAXX = 1e7+5;
bool p[MAXX];
int prime[MAXX], sz;

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

```

cfc3
5861
73ae
427e
9bc6
9628
1ec8
bf28
e82c
b6a9
5f51
95cf
95cf
95cf

4.9 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];
e0f3                 pcnt[j] = pcnt[b] + 1;
a96c                 f[j] = _____; // f[j] = f(i^pcnt[j])
95cf             }
95cf         }
34c0     for (int j = 0; i * prime[j] < N; j++) {
f87a         int x = i * prime[j]; p[x] = 1;
20cc         if (i % prime[j] == 0) {
9985             pval[x] = pval[i] * prime[j];
3f93             pcnt[x] = pcnt[i] + 1;
8e2e         } else {
cc91             pval[x] = prime[j];
6322             pcnt[x] = 1;
95cf         }
6191         if (x != pval[x]) {
d614             f[x] = f[x / pval[x]] * f[pval[x]]
95cf         }
5f51         if (i % prime[j] == 0) break;
95cf     }
95cf }
95cf }
95cf }

```

4.10 Miller-Rabin primality test

The array `a[]` (excluding 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 $n < 2^{64}$.

```

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

```

f16f
59f2
427e
3f11
c320
f410
2975
ece1
7f99
e257
d7ff
8d2e
95cf
438e
d490
95cf
3361
95cf

4.11 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;

```

2e6b
427e
54a5
45eb
d3e5
3c69
0964
4753
5952
9e5b
33d5
e1bf
e1bf

```

a313         d = gcd(x>y ? x-y : y-x, n);
95cf     }
95cf     }
5d89     return d;
95cf }

```

4.12 Qusai-polynomial sum

Must call `init()` before use!

```

b24e namespace polysum {
1dc8 #define rep(i, a, n) for (int i = a; i < n; i++)
1481 #define per(i, a, n) for (int i = n - 1; i >= a; i--)
3946 const int D = 2010;
c076 ll a[D], f[D], g[D], p[D], p1[D], p2[D], b[D], h[D][2], C[D];
c4cb ll powmod(ll a, ll b) {
e4b7     ll res = 1;
af5c     a %= mod;
6e39     assert(b >= 0);
b1fa     for (; b; b >>= 1) {
0684         if (b & 1) res = res * a % mod;
05a8         a = a * a % mod;
95cf     }
244d     return res;
95cf }
e88b ll calcn(int d, ll *a, ll n) { // a[0].. a[d] a[n]
b4aa     if (n <= d) return a[n];
d6be     p1[0] = p2[0] = 1;
3245     rep(i, 0, d + 1) {
ffec         ll t = (n - i + mod) % mod;
532d         p1[i + 1] = p1[i] * t % mod;
95cf     }
3245     rep(i, 0, d + 1) {
9800         ll t = (n - d + i + mod) % mod;
9f60         p2[i + 1] = p2[i] * t % mod;
95cf     }
19f3     ll ans = 0;
3245     rep(i, 0, d + 1) {
860e         ll t = g[i] * g[d - i] % mod * p1[i] % mod * p2[d - i] % mod * a[i] % mod;
752a         if ((d - i) & 1)
a69f             ans = (ans - t + mod) % mod;
649a         else
29fe             ans = (ans + t) % mod;

```

```

    }
    return ans;
}
void init(int M) {
    f[0] = f[1] = g[0] = g[1] = 1;
    rep(i, 2, M + 5) f[i] = f[i - 1] * i % mod;
    g[M + 4] = powmod(f[M + 4], mod - 2);
    per(i, 1, M + 4) g[i] = g[i + 1] * (i + 1) % mod;
}
ll polysum(ll m, ll *a, ll n) { // a[0].. a[m] \sum_{i=0}^{n-1} a[i]
    ll b[D];
    for (int i = 0; i <= m; i++) b[i] = a[i];
    b[m + 1] = calcn(m, b, m + 1);
    rep(i, 1, m + 2) b[i] = (b[i - 1] + b[i]) % mod;
    return calcn(m + 1, b, n - 1);
}
ll qpolysum(ll R, ll n, ll *a, ll m) { // a[0].. a[m] \sum_{i=0}^{n-1} a[i]*R^i
    if (R == 1) return polysum(n, a, m);
    a[m + 1] = calcn(m, a, m + 1);
    ll r = powmod(R, mod - 2), p3 = 0, p4 = 0, c, ans;
    h[0][0] = 0;
    h[0][1] = 1;
    rep(i, 1, m + 2) {
        h[i][0] = (h[i - 1][0] + a[i - 1]) * r % mod;
        h[i][1] = h[i - 1][1] * r % mod;
    }
    rep(i, 0, m + 2) {
        ll t = g[i] * g[m + 1 - i] % mod;
        if (i & 1)
            p3 = ((p3 - h[i][0] * t) % mod + mod) % mod,
            p4 = ((p4 - h[i][1] * t) % mod + mod) % mod;
        else
            p3 = (p3 + h[i][0] * t) % mod, p4 = (p4 + h[i][1] * t) % mod;
    }
    c = powmod(p4, mod - 2) * (mod - p3) % mod;
    rep(i, 0, m + 2) h[i][0] = (h[i][0] + h[i][1] * c) % mod;
    rep(i, 0, m + 2) C[i] = h[i][0];
    ans = (calcn(m, C, n) * powmod(R, n) - c) % mod;
    if (ans < 0) ans += mod;
    return ans;
}
} // namespace polysum

```

95cf
4206
95cf
1901
6323
fe69
b375
7e87
95cf
5f6d
2f0c
a950
96b8
7785
cc07
95cf
c704
356d
ee67
2f7b
c222
c576
4d99
dcbd
3f1a
95cf
dc94
2d72
59aa
60b1
19f7
649a
b9ee
95cf
6eed
a893
9267
8a10
2dc8
4206
95cf
95cf

5 Graph Theory

5.1 Strongly connected component

```

837c const int MAXV = 100005;
427e
2ea0 struct graph{
88e3     vector<int> adj[MAXV];
9cad     stack<int> s;
3d02     int V; // number of vertices
8b6c     int pre[MAXV], lnk[MAXV], scc[MAXV];
27ee     int time, sccn;
427e
bfab     void add_edge(int u, int v){
c71a         adj[u].push_back(v);
95cf     }
427e
d714     void dfs(int u){
7e41         pre[u] = lnk[u] = ++time;
80f6         s.push(u);
18f6         for (int v : adj[u]){
173e             if (!pre[v]){
5f3c                 dfs(v);
002c                 lnk[u] = min(lnk[u], lnk[v]);
6068             } else if (!scc[v]){
d5df                 lnk[u] = min(lnk[u], pre[v]);
95cf             }
95cf         }
8de2         if (lnk[u] == pre[u]){
660f             sccn++;
3c9e             int x;
a69f             do {
3834                 x = s.top(); s.pop();
b0e9                 scc[x] = sccn;
6757             } while (x != u);
95cf         }
95cf     }
427e
4c88     void find_scc(){
f4a2         time = sccn = 0;
8de7         memset(scc, 0, sizeof scc);
8c2f         memset(pre, 0, sizeof pre);
6901         Rep (i, V){

```

```

            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]]);
        }
}
};

```

56d1
95cf
95cf
427e
27ce
364d
1a1e
21a2
b730
b46e
95cf
95cf
329b

5.2 Vertex biconnected component

```

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

    stack<pair<int, int>> s;

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

    int dfs(int u, int fa) {
        int lowu = pre[u] = ++dfs_clock;
        int child = 0;
        for (int v : adj[u]) {
            if (!pre[v]) {
                s.push({u, v});
                child++;
                int lowv = dfs(v, u);
                lowu = min(lowu, lowv);
                if (lowv >= pre[u]) {
                    iscut[u] = 1;
                    bcc[bcc_cnt].clear();
                    bcce[bcc_cnt].clear();

```

0f42
2ea0
33ae
848f
6b06
427e
76f7
427e
bfab
c71a
a717
95cf
427e
7d3c
9fe6
ec14
18f6
173e
e7f8
fdcf
f851
189c
b687
6323
57eb
90b8

```

a147         while (1) {
a6a3             int xu, xv;
a0c3             tie(xu, xv) = s.top(); s.pop();
0ef5             bccno[bcc_cnt].insert({min(xu, xv), max(xu, xv)});
3db2             if (bccno[xu] != bcc_cnt) {
e0db                 bcc[bcc_cnt].push_back(xu);
d27f                 bccno[xu] = bcc_cnt;
95cf             }
f357             if (bccno[xv] != bcc_cnt) {
752b                 bcc[bcc_cnt].push_back(xv);
57c9                 bccno[xv] = bcc_cnt;
95cf             }
7096             if (xu == u && xv == v) break;
95cf         }
03f5         bcc_cnt++;
95cf     }
7470 } else if (pre[v] < pre[u] && v != fa) {
e7f8     s.push({u, v});
f115     lowu = min(lowu, pre[v]);
95cf }
95cf }
e104 if (fa < 0 && child == 1) iscut[u] = 0;
1160 return lowu;
95cf }
427e
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 {

```

```

int u, v;
LL w;
};

const int MAXN = 10005;
LL in[MAXN];
int pre[MAXN], vis[MAXN], id[MAXN];

LL getans(int n, int rt, vector<edge>& edges) {
    LL ans = 0;
    int cnt = 0;
    while (1) {
        Rep (i, n) in[i] = LLONG_MAX, id[i] = vis[i] = 0;
        for (auto e : edges) {
            if (e.u != e.v and e.w < in[e.v]) {
                pre[e.v] = e.u;
                in[e.v] = e.w;
            }
        }
        in[rt] = 0;
        Rep (i, n) {
            if (in[i] == LLONG_MAX) return -1;
            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;
    }
}

```

```

54f1
309c
329b
427e
f5a4
7124
1c1d
427e
5a43
f7ff
8abb
a147
641a
0705
073a
c1df
5fbc
95cf
95cf
3fdb
34d7
3c97
cf57
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, $O(\sqrt{VE})$.

```
bcf8 struct edge{
60e2     int from, to;
5e6d     LL cap, flow;
329b };
427e
e2cd const int MAXN = 1005;
9062 struct Dinic {
4dbf     int n, m, s, t;
9f0c     vector<edge> edges;
b891     vector<int> G[MAXN];
bbb6     bool vis[MAXN];
b40a     int d[MAXN];
ddec     int cur[MAXN];
427e
5973     void add_edge(int from, int to, LL cap) {
7b55         edges.push_back(edge{from, to, cap, 0});
1db7         edges.push_back(edge{to, from, 0, 0});
fe77         m = edges.size();
dff5         G[from].push_back(m-2);
8f2d         G[to].push_back(m-1);
95cf     }
427e
1836     bool bfs() {
3b73         memset(vis, 0, sizeof(vis));
93d2         queue<int> q;
5d13         q.push(s);
2cd2         vis[s] = 1;
721d         d[s] = 0;
cc78         while (!q.empty()) {
66ba             int x = q.front(); q.pop();
3b61             for (int i = 0; i < G[x].size(); i++) {
b510                 edge& e = edges[G[x][i]];
bba9                 if (!vis[e.to] && e.cap > e.flow) {
cd72                     vis[e.to] = 1;
cf26                     d[e.to] = d[x] + 1;
ca93                     q.push(e.to);
```

```

        }
    }
    return vis[t];
}

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

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

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

95cf
95cf
95cf
b23b
95cf
427e
9252
6904
8bf9
f515
b510
2374
1cce
e16d
a74d
23e5
97ed
95cf
95cf
84fb
95cf
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)

```

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
84ee struct Hungarian{
fbf6     int nx, ny;
9ec6     vector<int> mx, my;
9d4c     vector<vector<int> > e;
edec     vector<bool> mark;
427e
8324     void init(int nx, int ny){
c1d1         this->nx = nx;
f9c1         this->ny = ny;
ac92         mx.resize(nx); my.resize(ny);
3f11         e.clear(); e.resize(nx);
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;

```

```

        fill(range(mx), -1);
        fill(range(my), -1);
        rep (i, nx){
            fill(range(mark), false);
            if (augment(i)) ret++;
        }
        return ret;
    }
};

```

b0f1
b957
4ed1
13a5
cc89
95cf
ee0f
95cf
329b

5.6 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;

```

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      p[s] = 0; a[s] = INT_MAX;
cc78      while (!q.empty()){
b0aa          int u = q.front(); q.pop(); inq[u] = false;
3bba          for (int i : G[u]) {
56d8              edge& e = edges[i];
3601              if (e.cap > e.flow && d[e.to] > d[u] + e.cost){
55bc                  d[e.to] = d[u] + e.cost;
0bea                  p[e.to] = G[u][i];
8249                  a[e.to] = min(a[u], e.cap - e.flow);
e5d3                  if (!inq[e.to]) q.push(e.to), inq[e.to] = true;
95cf              }
95cf          }
95cf      }
6d7c      return d[t] != INF;
95cf  }

427e
71a4  void augment(){
06f1      int u = t;
b19d      while (u != s){
db09          edges[p[u]].flow += a[t];
25a9          edges[p[u]^1].flow -= a[t];
e6c9          u = edges[p[u]].from;
95cf      }
95cf  }
427e

6e20  #ifdef GIVEN_FLOW
5972      bool min_cost(int s, int t, int f, LL& cost) {
590d          this->s = s; this->t = t;
21d4          int flow = 0;
23cb          cost = 0;
22dc          while (spfa()) {
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;

```

```

int flow = 0;
cost = 0;
while (spfa()) {
    augment();
    flow += a[t]; cost += a[t] * d[t];
}
return flow;
}
#endif
};

```

5.7 Global minimum cut (Stoer-Wagner)

```

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;

```

21d4
23cb
22dc
bcd8
2a83
95cf
84fb
95cf
1937
329b

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    return {bestw, bestc};
95cf  }

```

5.8 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  }

```

```

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

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

```

5.9 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];
int sz[100005], sum;

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

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

```

```

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 }

```

5.10 DSU on tree

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

Usage:

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

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

```

1fb6 vector<int> adj[100005];
901d int sz[100005], son[100005];
427e
5559 void decomp(int u, int p) {
50c0     sz[u] = 1;
18f6     for (int v : adj[u]) {
bd87         if (v == p) continue;
a851         decomp(v, u);
8449         sz[u] += sz[v];
d28c         if (sz[v] > sz[son[u]]) son[u] = v;
95cf     }

```

```

}

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);
}

```

```

95cf
427e
b7ec
62f5
4412
30b3
95cf
427e
7467
33ff
72a2
427e
427e
427e
9866
427e
18a9
1ab0
f241
427e
3d3b
74c6
c13d
95cf
427e
427e
427e
c54f
9dec
427e
427e
4e3e
95cf

```

6 Data Structures

6.1 Fenwick tree (point update range query)

```

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

```

```

9976
d7af
99ff

```

```

427e void init(int n) { // fill the array with 0
d34f     tr.resize(N = n + 5);
1010 }
95cf
427e LL sum(int n) {
63d0     LL ans = 0;
f7ff     while (n) {
e290         ans += tr[n];
0715         n &= n - 1;
c0d4     }
95cf     return ans;
4206 }
95cf
427e void add(int n, LL x){
f4bd     while (n < N) {
ad20         tr[n] += x;
6c81         n += n & -n;
0af5     }
95cf }
95cf
329b };

```

6.2 Fenwick tree (range update point query)

```

3d03 struct bit_rupq{ // range update, point 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
38d4 LL query(int n) {
f7ff     LL ans = 0;
ad20     while (n < N) {
0715         ans += tr[n];
0af5         n += n & -n;
95cf     }
4206     return ans;
95cf }
427e
f4bd void add(int n, LL x) {

```

```

while (n){
    tr[n] += x;
    n &= n - 1;
}
};

```

```

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)
#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]);

```

```

3942
1ebb
451a
27be
4510
427e
ac35
1294
427e
1344
bbe9
95cf
427e
e4bc
5dd6
6eff
95cf
427e
d658
b82c
aa86
649f
95cf
427e
b149
3159
0a90
0f4a
045e
ac0a
95cf
1b82
9547

```

```

0e73     push_add(rson, tadd[o]);
6234     tadd[o] = 0;
95cf     }
95cf     }
427e
471c void build(int o, int ll, int rr) {
0e87     int mm = (ll + rr) / 2;
9d27     l[o] = ll; r[o] = rr; m[o] = mm;
ac0a     tmul[o] = 1;
5c92     if (ll == mm) {
001f         scanf("%lld", val + o);
e5b6         val[o] %= p;
8e2e     } else {
7293         build(lson, ll, mm);
5e67         build(rson, mm, rr);
ba26         pull(o);
95cf     }
95cf     }
427e
4406 void add(int o, int ll, int rr, LL x) {
3c16     if (ll <= l[o] && r[o] <= rr) {
db32         push_add(o, x);
8e2e     } else {
c4b0         push(o);
4305         if (m[o] > ll) add(lson, ll, rr, x);
d5a6         if (m[o] < rr) add(rson, ll, rr, x);
ba26         pull(o);
95cf     }
95cf     }
427e
48cd void mul(int o, int ll, int rr, LL x) {
3c16     if (ll <= l[o] && r[o] <= rr) {
e7d0         push_mul(o, x);
8e2e     } else {
c4b0         push(o);
d1ba         if (ll < m[o]) mul(lson, ll, rr, x);
67f3         if (m[o] < rr) mul(rson, ll, rr, x);
ba26         pull(o);
95cf     }
95cf     }
427e
0f62 LL query(int o, int ll, int rr) {
3c16     if (ll <= l[o] && r[o] <= rr) {
6dfe         return val[o];

```

```

    } 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;

```

```

8e2e
c4b0
462a
5cca
bbf9
95cf
95cf
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

```

namespace LCT {
    const int MAXN = 300005;
    int fa[MAXN], ch[MAXN][2], val[MAXN], sum[MAXN];
    bool rev[MAXN];

    bool isroot(int x) {
        return ch[fa[x]][0] == x || ch[fa[x]][1] == x;
    }

    void pull(int x) {
        sum[x] = val[x] ^ sum[ch[x][0]] ^ sum[ch[x][1]];
    }

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

    void push(int x) {
        if (rev[x]) {
            if (ch[x][0]) reverse(ch[x][0]);
            if (ch[x][1]) reverse(ch[x][1]);
            rev[x] = 0;

```

```

427e
427e
ed4d
5ece
6a6d
c6e1
427e
7839
45a9
95cf
427e
3bf9
6664
95cf
427e
3698
7850
52c6
95cf
427e
1a53
8f1f
ebf3
6eb0
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) {

```

```

chroot(x);
access(y);
}

int Root(int x) {
    access(x);
    while (ch[x][0]) {
        push(x);
        x = ch[x][0];
    }
    splay(x);
    return x;
}

void Link(int u, int v) { // assume unconnected before
    chroot(u);
    fa[u] = v;
}

void Cut(int u, int v) { // assume connected before
    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;
}
};

```

```

3015
29b5
95cf
427e
d87a
766a
874d
a97b
b83a
95cf
8fec
d074
95cf
427e
70d3
b8a5
2448
95cf
427e
c2f4
e8ce
fd95
743b
95cf
427e
6ca2
e8ce
a5ba
95cf
427e
eaba
46ce
1d62
95cf
329b

```

6.5 Balanced binary search tree from pb_ds

```

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

```

```

0475
332d
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

```

```

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

```

```

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

```

```

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

```

```

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

```

```

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

```

6.7 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

93c0
d30c
181e
cb5a
8edb
2412
8e2e
0119
95cf
95cf
427e
c9ad
9e27
95cf
427e
b19c
5794
ce96
181e
203d
f44a
649a
1024
95cf
2b3e
5ffd
95cf
427e
e80f
c246
95cf
865a
427e
99ce
1987
bb3c
95cf

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 $[l, r)$.

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

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

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;
}
};
```

3131
5e44
852d
0b03
029f
93e1
e1fa
95cf
5771
95cf
95cf
427e
b7b3
2273
5502
d480
2099
95cf
a1c8
95cf
95cf
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
```

db63
b330
69ae
427e
8041

```

427e void init(int n){
d34f     int l = 31 - __builtin_clz(n);
ce01     rep (i, n) st[i][0] = a[i];
cf75     rep (j, l)
b811         rep (i, 1+n-(1<<j))
6937             st[i][j+1] = ext(st[i][j], st[i+(1<<j)][j]);
082a }
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;
ffa9     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
// 2 on vertices
int ptOnSeg2(pt& p, seg& s){
    vec v1 = s.first - p, v2 = s.second - p;
    T ip = (v1, v2);
    if (v1 * v2 != 0 || ip > 0) return 0;
    return (v1, v2) ? 1 : 2;
}

// if two orthogonal rectangles do not touch, return true
inline bool nIntRectRect(seg a, seg b){
    return min(a.first.x, a.second.x) > max(b.first.x, b.second.x) ||
           min(a.first.y, a.second.y) > max(b.first.y, b.second.y) ||
           min(b.first.x, b.second.x) > max(a.first.x, a.second.x) ||
           min(b.first.y, b.second.y) > max(a.first.y, a.second.y);
}

// >0 in order
// <0 out of order
// =0 not standard
inline double rotOrder(vec a, vec b, vec c){return double(a*b)*(b*c);}

inline bool intersect(seg a, seg b){
    // ! 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;
}

// 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;
}

```

```

95cf }
427e
2c68 template <typename Tp = T>
5894 inline pt getIntersection(pt P, vec v, pt Q, vec w){
6850     static_assert(is_same<Tp, double>::value, "must_be_double!");
7c9a     return P + v * (w*(P-Q)/(v*w));
95cf }
427e
427e // -1 outside the polygon
427e // 0 on the border of the polygon
427e // 1 inside the polygon
cbdd int ptOnPoly(pt p, pt* poly, int n){
5fb4     int wn = 0;
1294     for (int i = 0; i < n; i++) {
427e         T k, d1 = poly[i].y - p.y, d2 = poly[(i+1)%n].y - p.y;
b957         if (k = (poly[(i+1)%n] - poly[i])*(p - poly[i])){
8c40             if (k > 0 && d1 <= 0 && d2 > 0) wn++;
3c4d             if (k < 0 && d2 <= 0 && d1 > 0) wn--;
aad3         } else return 0;
95cf     }
0a5f     return wn ? 1 : -1;
95cf }
427e
d4a3 istream& operator >> (istream& lhs, pt& rhs){
fa86     lhs >> rhs.x >> rhs.y;
331a     return lhs;
95cf }
427e
07ae istream& operator >> (istream& lhs, seg& rhs){
5cab     lhs >> rhs.first >> rhs.second;
331a     return lhs;
95cf }

```

8 Appendices

8.1 Primes

8.1.1 First primes

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

8.1.2 Arbitrary length primes

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

8.1.3 $\sim 1 \times 10^9$

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

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

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

8.2 Pell's equation

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

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

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

Some smallest solutions to Pell's equation:

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

8.3 Burnside's lemma and Polya's enumeration theorem

The Burnside's lemma says that

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

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

The unweighted version of Pólya enumeration theorem says that

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

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

8.4 Lagrange's interpolation

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

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