

南京大学 ACM-ICPC 集训队代码模版库



Contents

1 General	3	5 Graph Theory	14
1.1 Code library checksum	3	5.1 Strongly connected component	14
1.2 Makefile	3	5.2 Vertex biconnected component	15
1.3 .vimrc	3	5.3 Minimum spanning arborescence (Chu-Liu)	15
1.4 Template	3	5.4 Maximum flow (Dinic)	16
2 Miscellaneous Algorithms	4	5.5 Maximum cardinality bipartite matching (Hungarian)	17
2.1 2-SAT	4	5.6 Minimum cost maximum flow	18
2.2 Knuth's optimization	4	5.7 Global minimum cut (Stoer-Wagner)	19
2.3 Mo's algorithm	5	5.8 Heavy-light decomposition	19
3 String	5	5.9 Centroid decomposition	20
3.1 Knuth-Morris-Pratt algorithm	5	5.10 DSU on tree	20
3.2 Manacher algorithm	6	6 Data Structures	21
3.3 Aho-corasick automaton	6	6.1 Fenwick tree (point update range query)	21
3.4 Trie	7	6.2 Fenwick tree (range update point query)	21
3.5 Rolling hash	7	6.3 Segment tree	22
4 Math	8	6.4 Link/cut tree	23
4.1 Matrix powermod	8	6.5 Balanced binary search tree from pb_ds	24
4.2 Linear basis	8	6.6 Persistent segment tree, range k-th query	24
4.3 Gauss elimination over finite field	8	6.7 Sparse table, range extremum query	25
4.4 Berlekamp-Massey algorithm	9	7 Geometrics	25
4.5 Fast Walsh-Hadamard transform	10	7.1 2D geometric template	25
4.6 Fast fourier transform	10	8 Appendices	27
4.7 Number theoretic transform	11	8.1 Primes	27
4.8 Sieve of Euler	12	8.1.1 First primes	27
4.9 Sieve of Euler (General)	12	8.1.2 Arbitrary length primes	27
4.10 Miller-Rabin primality test	12	8.1.3 $\sim 1 \times 10^9$	27
4.11 Pollard's rho algorithm	13	8.1.4 $\sim 1 \times 10^{18}$	27
4.12 Qusai-polynomial sum	13	8.2 Pell's equation	27
		8.3 Burnside's lemma and Polya's enumeration theorem	28
		8.4 Lagrange's interpolation	28

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
5f25 run : $(t)
427e
207e $(t) : $(t).cpp
155d     g++ --std=c++14 -O2 -D__LOCAL_DEBUG__ -fsanitize=undefined -fsanitize=address
        -ggdb -pipe -o $@ $<
```

1.3 .vimrc

```
914c set nocompatible
733d syntax on
6bbc colorscheme slate
7db5 set number
b0e3 set cursorline
061b set shiftwidth=2
8011 set softtabstop=2
a66d set tabstop=2
d23a set expandtab
5245 set magic
740c set smartindent
bee8 set backspace=indent,eol,start
815d set cmdheight=1
0a40 set laststatus=2
e458 set statusline=\ %<%F[%1*M%*%n%R%H]%=\ %y\ %0(%{&fileformat}\ %{&encoding}\ %c
        :%l/%L%\
1c67 set whichwrap=b,s,<,>[,]
```

1.4 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) (x).begin(), (x).end()
typedef long long LL;
typedef unsigned long long ULL;

template <unsigned p>
struct Zp{
    unsigned x;
    Zp(unsigned x):x(x){}
    operator unsigned(){return x;}
    Zp operator ^ (ULL e) {
        Zp b=x, r=1;
        while (e) {
            if (e&1) r=r*b;
            b=b*b;
            e>>=1;
        }
        return r;
    }
    Zp operator + (Zp rhs) {return (x+rhs)%p;}
    Zp operator - (Zp rhs) {return (x+p-rhs)%p;}
    Zp operator * (Zp rhs) {return x*rhs%p;}
    Zp operator / (Zp rhs) {return Zp(x)*(rhs^(p-2));}
};

typedef Zp<1000000007> zp;

zp operator"" _ (ULL n){return n;}
```

302f
421c
427e
426f
3341
611f
a8cb
e6b5
1937
0d6c
cfe3
8843
5cad
b773
427e
5120
87b8
7797
ff67
22e3
fecc
4fce
3e90
5421
2059
16fc
95cf
547e
95cf
a2f5
664b
3ec4
7cfd
329b
427e
370f
427e
0795

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(l, 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 Trie

```

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

```

```

if (!tr[p][c]) {
    memset(tr[n], 0, sizeof(tr[n]));
    tag[n] = 0;
    tr[p][c] = n++;
}
p = tr[p][c];
}
tag[p] = t;
}

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

```

```

d6c8
26dd
2e5c
73bb
95cf
f119
95cf
35ef
95cf
427e
427e
427e
216c
d50a
9c94
3140
f339
f119
95cf
840e
95cf
329b

```

3.5 Rolling hash

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

Usage:

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

```

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

```

```

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

```

```

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

```

```

1e42
9f60
0291
427e
6832
c919
95cf
427e
599a
286f
d00f
4aa9
95cf

```

```

427e struct hasher {
7e62     LL val[MAXN];
534a
427e     void build(const char *str) { // assume Lower-case Letter only
4554         for (int i = 0; str[i]; i++)
f937             val[i+1] = (mul(val[i], g) + str[i]) % mod;
9645     }
95cf
427e     LL operator() (int l, int r) { // [l, r)
19f8         return (val[r] - mul(val[l], pg[r - l]) + mod) % mod;
9986     }
95cf } ha;
b179

```

4 Math

4.1 Matrix powermod

```

44b4 const int MAXN = 105;
92df const LL modular = 1000000007;
5c83 int n; // order of matrices
427e
8864 struct matrix{
3180     LL m[MAXN][MAXN];
427e
43c5     void operator *=(matrix& a){
e735         static LL t[MAXN][MAXN];
34d7         Rep (i, n){
4c11             Rep (j, n){
ee1e                 t[i][j] = 0;
c4a7                 Rep (k, n){
fc4f                     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;

```

```

void m_powmod(matrix& b, LL e){
    memset(r.m, 0, sizeof(r.m));
    Rep(i, n)
        r.m[i][i] = 1;
    while (e){
        if (e & 1) r *= b;
        b *= b;
        e >>= 1;
    }
}

```

```

3ec2
83f0
a7c3
de64
3e90
5a0e
35c5
16fc
95cf
95cf

```

4.2 Linear basis

```

const int MAXD = 30;
struct linearbasis {
    ULL b[MAXD] = {};

    bool insert(ull 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;
    }
};

```

```

8b44
03a6
3558
427e
842f
9b2b
de36
ee78
037f
7836
f0b4
b0aa
46c9
8295
3361
95cf
95cf
438e
95cf
329b

```

4.3 Gauss elimination over finite field

```

const LL p = 1000000007;

LL powmod(LL b, LL e) {
    LL r = 1;
    while (e) {

```

```

b784
427e
2a2c
95a2
3e90

```



```

1783     if (e & 1) r = r * b % p;
5549     b = b * b % p;
16fc     e >>= 1;
95cf }
547e return r;
95cf }
427e
c130 typedef vector<LL> VLL;
42ac typedef vector<VLL> VWLL;
427e
2c62 LL gauss(VWLL &a, VWLL &b) {
561b     const int n = a.size(), m = b[0].size();
a25e     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

```

```

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

```

```

37e1
50dc
95cf
f27f
95cf

```

4.4 Berlekamp-Massey algorithm

```

const LL MOD = 1000000007;

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

struct Poly {
    vector<int> a;

    Poly() { a.clear(); }

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

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

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

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

```

```

2b86
427e
391d
32d3
3e90
9a62
29ea
16fc
95cf
547e
95cf
427e
32a6
afe0
427e
9794
427e
de81
427e
8087
427e
16de
b31d
f915
cecf
95cf
427e
fa1a
5b57
501c
5de5
3041
95cf
95cf

```

```

ee0f     return ret;
95cf     }
427e
c856     Poly operator - (const Poly &b) {
bd55         vector<int> na(max(this->length(), b.length()));
d1a7         for (int i = 0; i < (int)na.size(); ++i) {
3507             int aa = i < this->length() ? this->a[i] : 0,
2bee             bb = i < b.length() ? b.a[i] : 0;
9526             na[i] = (aa + MOD - bb) % MOD;
95cf         }
cecf         return Poly(na);
95cf     }
329b };
427e
5473     Poly operator * (const int &c, const Poly &p) {
72de         vector<int> na(p.length());
d1a7         for (int i = 0; i < (int)na.size(); ++i) {
bf0c             na[i] = (long long)c * p.a[i] % MOD;
95cf         }
aaab         return na;
95cf     }
427e
afff     vector<int> solve(vector<int> a) {
9f23         int n = a.size();
58d0         Poly s, b;
4e8f         s.a.push_back(1), b.a.push_back(1);
c2aa         for (int i = 1, j = 0, ld = a[0]; i < n; ++i) {
4158             int d = s.calc(a, i);
d503             if (d) {
c29d                 if ((s.length() - 1) * 2 <= i) {
db9d                     Poly ob = b;
6bce                     b = s;
1d0e                     s = s - (long long)d * inverse(ld) % MOD * ob.move(i - j);
0889                     j = i;
64f1                     ld = d;
8e2e                 } else {
714e                     s = s - (long long)d * inverse(ld) % MOD * b.move(i - j);
95cf                 }
95cf             }
95cf         }
427e         // Caution: s.a might be shorter than expected
e235         return s.a;
95cf     }

```

4.5 Fast Walsh-Hadamard transform

```

void fwt(int* a, int n){
    for (int d = 1; d < n; d <= 1)
        for (int i = 0; i < n; i += d < 1)
            rep (j, d){
                int x = a[i+j], y = a[i+j+d];
                // a[i+j] = x+y, a[i+j+d] = x-y;    // xor
                // a[i+j] = x+y;                    // and
                // a[i+j+d] = x+y;                    // or
            }
    }

void ifwt(int* a, int n){
    for (int d = 1; d < n; d <= 1)
        for (int i = 0; i < n; i += d < 1)
            rep (j, d){
                int x = a[i+j], y = a[i+j+d];
                // a[i+j] = (x+y)/2, a[i+j+d] = (x-y)/2;    // xor
                // a[i+j] = x-y;                            // and
                // a[i+j+d] = y-x;                            // or
            }
    }

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

```

4.6 Fast fourier transform

```

const int NMAX = 1<<20;

typedef complex<double> cplx;

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

```

```

427e FFT(int k){
1442     K = k; N = 1 << k;
e209     rep (i, N){
b393         rev[i] = (rev[i>>1]>>1) | ((i&1)<<(K-1));
7ba3         omega[i] = polar(1.0, 2.0 * PI / N * i);
1908         oinv[i] = conj(omega[i]);
a166     }
95cf }
95cf
427e void dft(cplx* a, cplx* w){
b941     rep (i, N) if (i < rev[i]) swap(a[i], a[rev[i]]);
a215     for (int l = 2; l <= N; l *= 2){
ac6e         int m = l/2;
2969         for (cplx* p = a; p != a + N; p += l)
b3cf             rep (k, m){
c24f                 cplx t = w[N/l*k] * p[k+m];
fe06                 p[k+m] = p[k] - t; p[k] += t;
ecbf             }
95cf     }
95cf }
427e void fft(cplx* a){dft(a, omega);}
a123 void ifft(cplx* a){
3b2f     dft(a, oinv);
57fc     rep (i, N) a[i] /= N;
95cf }
427e void conv(cplx* a, cplx* b){
bdc0     fft(a); fft(b);
6497     rep (i, N) a[i] *= b[i];
12a5     ifft(a);
f84e }
95cf }
329b };

```

4.7 Number theoretic transform

```

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

```

```

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

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

    NTT(int k){
        K = k; N = 1 << k;
        g = powmod(G, (P-1)/N);
        g_inv = powmod(g, N-1);
        omega[0] = oinv[0] = 1;
        rep (i, N){
            rev[i] = (rev[i>>1]>>1) | ((i&1)<<(K-1));
            if (i){
                omega[i] = omega[i-1] * g % P;
                oinv[i] = oinv[i-1] * g_inv % P;
            }
        }
    }

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

    void ntt(LL* a){_ntt(a, omega);}

```

```

87ab
c47c
0eda
81af
9827
427e
2a2c
95a2
3e90
6624
489e
16fc
95cf
547e
95cf
427e
f420
e209
7652
4b3a
e04f
b393
7ba3
ad4f
8d8b
9e14
95cf
95cf
427e
9668
a215
ac6e
2969
7a1d
c24f
0ad3
6209
fa1b
95cf
95cf
427e
92ea

```

```

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 };

```

4.8 Sieve of Euler

```

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

```

4.9 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;

```

```

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

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

```

4.10 Miller-Rabin primality test

The array a[] (excluding 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){
```

427e
8a8a
bdda
c6b9
427e
a643
01d6
b2b2
37d9
758c
81fd
e0f3
a96c
95cf
95cf
34c0
f87a
20cc
9985
3f93
8e2e
cc91
6322
95cf
6191
d614
95cf
5f51
95cf
95cf
95cf

f16f

```

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 }

```

4.11 Pollard's rho algorithm

```

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

```

namespace polysum {
#define rep(i, a, n) for (int i = a; i < n; i++)
#define per(i, a, n) for (int i = n - 1; i >= a; i--)
const int D = 2010;
ll a[D], f[D], g[D], p[D], p1[D], p2[D], b[D], h[D][2], C[D];
ll powmod(ll a, ll b) {
    ll res = 1;
    a %= mod;
    assert(b >= 0);
    for (; b; b >>= 1) {
        if (b & 1) res = res * a % mod;
        a = a * a % mod;
    }
    return res;
}
ll calcn(int d, ll *a, ll n) { // a[0].. a[d] a[n]
    if (n <= d) return a[n];
    p1[0] = p2[0] = 1;
    rep(i, 0, d + 1) {
        ll t = (n - i + mod) % mod;
        p1[i + 1] = p1[i] * t % mod;
    }
    rep(i, 0, d + 1) {
        ll t = (n - d + i + mod) % mod;
        p2[i + 1] = p2[i] * t % mod;
    }
    ll ans = 0;
    rep(i, 0, d + 1) {
        ll t = g[i] * g[d - i] % mod * p1[i] % mod * p2[d - i] % mod * a[i] % mod;
        if ((d - i) & 1)
            ans = (ans - t + mod) % mod;
        else
            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);
}

```

b24e
1dc8
1481
3946
c076
c4cb
e4b7
af5c
6e39
b1fa
0684
05a8
95cf
244d
95cf
e88b
b4aa
d6be
3245
ffec
532d
95cf
3245
9800
9f60
95cf
19f3
3245
860e
752a
a69f
649a
29fe
95cf
4206
95cf
1901
6323
fe69
b375

```

7e87     per(i, 1, M + 4) g[i] = g[i + 1] * (i + 1) % mod;
95cf }
5f6d ll polysum(ll m, ll *a, ll n) { // a[0].. a[m] \sum_{i=0}^{n-1} a[i]
2f0c     ll b[D];
a950     for (int i = 0; i <= m; i++) b[i] = a[i];
96b8     b[m + 1] = calcn(m, b, m + 1);
7785     rep(i, 1, m + 2) b[i] = (b[i - 1] + b[i]) % mod;
cc07     return calcn(m + 1, b, n - 1);
95cf }
c704 ll qpolysum(ll R, ll n, ll *a, ll m) { // a[0].. a[m] \sum_{i=0}^{n-1} a[i]*R^i
356d     if (R == 1) return polysum(n, a, m);
ee67     a[m + 1] = calcn(m, a, m + 1);
2f7b     ll r = powmod(R, mod - 2), p3 = 0, p4 = 0, c, ans;
c222     h[0][0] = 0;
c576     h[0][1] = 1;
4d99     rep(i, 1, m + 2) {
dcbb         h[i][0] = (h[i - 1][0] + a[i - 1]) * r % mod;
3f1a         h[i][1] = h[i - 1][1] * r % mod;
95cf     }
dc94     rep(i, 0, m + 2) {
2d72         ll t = g[i] * g[m + 1 - i] % mod;
59aa         if (i & 1)
60b1             p3 = ((p3 - h[i][0] * t) % mod + mod) % mod,
19f7             p4 = ((p4 - h[i][1] * t) % mod + mod) % mod;
649a         else
b9ee             p3 = (p3 + h[i][0] * t) % mod, p4 = (p4 + h[i][1] * t) % mod;
95cf     }
6eed     c = powmod(p4, mod - 2) * (mod - p3) % mod;
a893     rep(i, 0, m + 2) h[i][0] = (h[i][0] + h[i][1] * c) % mod;
9267     rep(i, 0, m + 2) C[i] = h[i][0];
8a10     ans = (calcn(m, C, n) * powmod(R, n) - c) % mod;
2dc8     if (ans < 0) ans += mod;
4206     return ans;
95cf }
95cf } // namespace polysum

```

5 Graph Theory

5.1 Strongly connected component

```

837c const int MAXV = 100005;

```

```

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

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

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

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

    vector<int> adjc[MAXV];
    void contract(){

```

427e
2ea0
88e3
9cad
3d02
8b6c
27ee
427e
bfab
c71a
95cf
427e
d714
7e41
80f6
18f6
173e
5f3c
002c
6068
d5df
95cf
95cf
8de2
660f
3c9e
a69f
3834
b0e9
6757
95cf
95cf
427e
4c88
f4a2
8de7
8c2f
6901
56d1
95cf
95cf
427e
27ce
364d

```

1a1e     Rep (i, V)
21a2         rep (j, adj[i].size()){
b730             if (scc[i] != scc[adj[i][j]])
b46e                 adjc[scc[i]].push_back(scc[adj[i][j]]);
95cf         }
95cf     }
329b };

```

5.2 Vertex biconnected component

```

0f42 const int MAXN = 100005;
2ea0 struct graph {
33ae     int pre[MAXN], iscut[MAXN], bccno[MAXN], dfs_clock, bcc_cnt;
848f     vector<int> adj[MAXN], bcc[MAXN];
6b06     set<pair<int, int>> bcce[MAXN];
427e
76f7     stack<pair<int, int>> s;
427e
bfab     void add_edge(int u, int v) {
c71a         adj[u].push_back(v);
a717         adj[v].push_back(u);
95cf     }
427e
7d3c     int dfs(int u, int fa) {
9fe6         int lowu = pre[u] = ++dfs_clock;
ec14         int child = 0;
18f6         for (int v : adj[u]) {
173e             if (!pre[v]) {
e7f8                 s.push({u, v});
fdcf                 child++;
f851                 int lowv = dfs(v, u);
189c                 lowu = min(lowu, lowv);
b687                 if (lowv >= pre[u]) {
6323                     iscut[u] = 1;
57eb                     bcc[bcc_cnt].clear();
90b8                     bcce[bcc_cnt].clear();
a147                     while (1) {
a6a3                         int xu, xv;
a0c3                         tie(xu, xv) = s.top(); s.pop();
0ef5                         bcce[bcc_cnt].insert({min(xu, xv), max(xu, xv)});
3db2                         if (bccno[xu] != bcc_cnt) {
e0db                             bcc[bcc_cnt].push_back(xu);

```

```

                                bccno[xu] = bcc_cnt;
                                }
                                if (bccno[xv] != bcc_cnt) {
                                    bcc[bcc_cnt].push_back(xv);
                                    bccno[xv] = bcc_cnt;
                                }
                                if (xu == u && xv == v) break;
                                }
                                bcc_cnt++;
                                }
                                } else if (pre[v] < pre[u] && v != fa) {
                                    s.push({u, v});
                                    lowu = min(lowu, pre[v]);
                                }
                                }
                                if (fa < 0 && child == 1) iscut[u] = 0;
                                return lowu;
                                }
                                }

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

};

```

5.3 Minimum spanning arborescence (Chu-Liu)

All vertices are 1-based.

Usage:

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

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

```

struct edge {
    int u, v;
    LL w;
};

const int MAXN = 10005;
LL in[MAXN];

```

```

d27f
95cf
f357
752b
57c9
95cf
7096
95cf
03f5
95cf
7470
e7f8
f115
95cf
95cf
e104
1160
95cf
427e
17be
8c2f
e2d2
40d3
fae2
5c63
95cf
329b

```

```

bcf8
54f1
309c
329b
427e
f5a4
7124

```

```

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];
a763             int u;
4b0e             for (u = i; u != rt && vis[u] != i && !id[u]; u = pre[u])
88a2                 vis[u] = i;
4b22             if (u != rt && !id[u]) {
b66e                 id[u] = ++cnt;
0443                 for (int v = pre[u]; v != u; v = pre[v])
5c22                     id[v] = cnt;
95cf             }
95cf         }
91e9         if (!cnt) return ans;
5e22         Rep (i, n) if (!id[i]) id[i] = ++cnt;
7400         for (auto& e : edges) {
7750             LL laz = in[e.v];
97ae             e.u = id[e.u];
fae6             e.v = id[e.v];
bdd2             if (e.u != e.v) e.w -= laz;
95cf         }
6cc4         n = cnt; rt = id[rt]; cnt = 0;
95cf     }
95cf }

```

5.4 Maximum flow (Dinic)

Usage:

add_edge(u, v, c) Add an edge from u to v with capacity c .
max_flow(s, t) Compute maximum flow from s to t .
Time Complexity: For general graph, $O(V^2E)$; for network with unit capacity, $O(\min\{V^{2/3}, \sqrt{E}\}E)$; for bipartite network, $O(\sqrt{VE})$.

```

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

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

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

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

```

bcf8
60e2
5e6d
329b
427e
e2cd
9062
4dbf
9f0c
b891
bbb6
b40a
ddec
427e
5973
7b55
1db7
fe77
dff5
8f2d
95cf
427e
1836
3b73
93d2
5d13
2cd2
721d
cc78
66ba
3b61
b510
bba9
cd72
cf26
ca93
95cf
95cf
95cf


```

b23b     return vis[t];
95cf     }
427e
9252     LL dfs(int x, LL a) {
6904         if (x == t || a == 0) return a;
8bf9         LL flow = 0, f;
f515         for (int& i = cur[x]; i < G[x].size(); i++) {
b510             edge& e = edges[G[x][i]];
2374             if(d[x] + 1 == d[e.to] && (f = dfs(e.to, min(a, e.cap-e.flow))) > 0)
                {
1cce                 e.flow += f;
e16d                 edges[G[x][i]^1].flow -= f;
a74d                 flow += f;
23e5                 a -= f;
97ed                 if(a == 0) break;
95cf             }
95cf         }
84fb         return flow;
95cf     }
427e
5bf2     LL max_flow(int s, int t) {
590d         this->s = s; this->t = t;
62e2         LL flow = 0;
ed58         while (bfs()) {
f326             memset(cur, 0, sizeof(cur));
fb3a             flow += dfs(s, LLONG_MAX);
95cf         }
84fb         return flow;
95cf     }
427e
c72e     vector<int> min_cut() { // call this after maxflow
1df9         vector<int> ans;
df9a         for (int i = 0; i < edges.size(); i++) {
56d8             edge& e = edges[i];
46a2             if(vis[e.from] && !vis[e.to] && e.cap > 0) ans.push_back(i);
95cf         }
4206         return ans;
95cf     }
329b };

```

5.5 Maximum cardinality bipartite matching (Hungarian)

```

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

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

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

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

    inline void add(int a, int b){
        e[a].push_back(b);
    }

    bool augment(int i){
        if (!mark[i]) {
            mark[i] = true;
            for (int j : e[i]){
                if (my[j] == -1 || augment(my[j])){
                    mx[i] = j; my[j] = i;
                    return true;
                }
            }
        }
        return false;
    }

    int match(){
        int ret = 0;
        fill(range(mx), -1);
        fill(range(my), -1);
        rep(i, nx){

```

```

302f
421c
427e
0d6c
cfe3
8843
5cad
427e
84ee
fbf6
9ec6
9d4c
edec
427e
8324
c1d1
f9c1
ac92
3f11
1023
95cf
427e
4589
486c
95cf
427e
0c2b
207c
dae4
6a1e
0892
9ca3
3361
95cf
95cf
95cf
438e
95cf
427e
3fac
5b57
b0f1
b957
4ed1

```

```

13a5         fill(range(mark), false);
cc89         if (augment(i)) ret++;
95cf     }
ee0f     return ret;
95cf }
329b };

```

5.6 Minimum cost maximum flow

```

bcf8 struct edge{
60e2     int from, to;
d698     int cap, flow;
32cc     LL cost;
329b };
427e
cc3e const LL INF = LLONG_MAX / 2;
2aa8 const int MAXN = 5005;
c6cb struct MCMF {
9ceb     int s, t, n, m;
9f0c     vector<edge> edges;
b891     vector<int> G[MAXN];
f74f     bool inq[MAXN]; // queue
8f67     LL d[MAXN];    // distance
9524     int p[MAXN];    // previous
b330     int a[MAXN];   // improvement
427e
f7f2 void add_edge(int from, int to, int cap, LL cost) {
24f0     edges.push_back(edge{from, to, cap, 0, cost});
95f0     edges.push_back(edge{to, from, 0, 0, -cost});
fe77     m = edges.size();
dff5     G[from].push_back(m-2);
8f2d     G[to].push_back(m-1);
95cf }
427e
3c52 bool spfa(){
93d2     queue<int> q;
8494     fill(d, d + MAXN, INF); d[s] = 0;
fd48     memset(inq, 0, sizeof(inq));
5e7c     q.push(s); inq[s] = true;
2dae     p[s] = 0; a[s] = INT_MAX;
cc78     while (!q.empty()){
b0aa         int u = q.front(); q.pop(); inq[u] = false;

```

```

        for (int i : G[u]) {
            edge& e = edges[i];
            if (e.cap > e.flow && d[e.to] > d[u] + e.cost){
                d[e.to] = d[u] + e.cost;
                p[e.to] = G[u][i];
                a[e.to] = min(a[u], e.cap - e.flow);
                if (!inq[e.to]) q.push(e.to), inq[e.to] = true;
            }
        }
    }
    return d[t] != INF;
}

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

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

```

```

3bba
56d8
3601
55bc
0bea
8249
e5d3
95cf
95cf
95cf
6d7c
95cf
427e
71a4
06f1
b19d
db09
25a9
e6c9
95cf
95cf
427e
6e20
5972
590d
21d4
23cb
22dc
bcdb
a671
b14d
3361
8e2e
2a83
95cf
95cf
438e
95cf
a8cb
f9a9
590d
21d4
23cb
22dc

```

```

bcd b      augment();
2a83      flow += a[t]; cost += a[t] * d[t];
95cf    }
84fb      return flow;
95cf    }
1937 #endif
329b };

```

5.7 Global minimum cut (Stoer-Wagner)

```

f9d7 typedef vector<LL> VI;
045e typedef vector<VI> WI;
427e
f012 pair<LL, VI> stoer(WI &w) {
66f7     int n = w.size();
4d98     VI used(n), c, bestc;
329d     LL bestw = -1;
427e
cd21     for (int ph = n - 1; ph >= 0; ph--) {
ec6e         VI wt = w[0], added = used;
f20e         int prev, last = 0;
4b32         rep (i, ph) {
8bfc             prev = last;
0706             last = -1;
4942             for (int j = 1; j < n; j++)
c4b9                 if (!added[j] && (last == -1 || wt[j] > wt[last]))
887d                     last = j;
71bc             if (i == ph - 1) {
9cfa                 rep (j, n) w[prev][j] += w[last][j];
1f25                 rep (j, n) w[j][prev] = w[prev][j];
5613                 used[last] = true;
8e11                 c.push_back(last);
bb8e                 if (bestw == -1 || wt[last] < bestw) {
bab6                     bestc = c;
372e                     bestw = wt[last];
95cf                 }
8e2e             } else {
caeb                 rep (j, n) wt[j] += w[last][j];
8b92                 added[last] = true;
95cf             }
95cf         }
95cf     }

```

```

    return {bestw, bestc};
}

```

```

038c
95cf

```

5.8 Heavy-light decomposition

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

```

const int MAXN = 100005;
vector<int> adj[MAXN];
int sz[MAXN], top[MAXN], fa[MAXN], son[MAXN], depth[MAXN], id[MAXN];

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);
}

```

```

0f42
0b32
42f2
427e
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     dfs2(root, root);
95cf }
427e
2c98 void query(int u, int v){
03a1     while (top[u] != top[v]){
45ec         if (depth[top[u]] < depth[top[v]]) swap(u, v);
427e         // id[top[u]] to id[u]
005b         u = fa[top[u]];
95cf     }
6083     if (depth[u] > depth[v]) swap(u, v);
427e     // id[u] to id[v]
95cf }

```

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.

```

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) {

```

```

sum = 0; getsz(u, 0);
u = getcent(u, 0); // update u to the centroid

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

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

```

```

618e
303c
427e
18f6
427e
95cf
427e
427e
18f6
c375
fa6b
a717
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`.

```

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>

```

```

1fb6
901d
427e
5559
50c0
18f6
bd87
a851
8449
d28c
95cf
95cf
427e
b7ec

```

```

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

```

6 Data Structures

6.1 Fenwick tree (point update range query)

```

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);

```

```

    }
    LL sum(int n) {
        LL ans = 0;
        while (n) {
            ans += tr[n];
            n &= n - 1;
        }
        return ans;
    }
    void add(int n, LL x){
        while (n < N) {
            tr[n] += x;
            n += n & -n;
        }
    }
};

```

6.2 Fenwick tree (range update point query)

```

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

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

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

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

```

```

95cf     }
95cf     }
329b };

```

6.3 Segment tree

```

3942 LL p;
1ebb const int MAXN = 4 * 100006;
451a struct segtree {
27be     int l[MAXN], m[MAXN], r[MAXN];
4510     LL val[MAXN], tadd[MAXN], tmul[MAXN];
427e
ac35 #define lson (o<<1)
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         }

```

```

}

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

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

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

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

```

```

95cf
427e
471c
0e87
9d27
ac0a
5c92
001f
e5b6
8e2e
7293
5e67
ba26
95cf
95cf
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    if (ll >= m[o]) return query(rson, ll, rr);
bbf9    return query(lson, ll, rr) + query(rson, ll, rr);
95cf    }
95cf    }
4d99    } seg;

```

6.4 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.

```

427e // about 0.13s per 100k ops @Luogu.org

```

```

427e namespace LCT {
ed4d     const int MAXN = 300005;
5ece     int fa[MAXN], ch[MAXN][2], val[MAXN], sum[MAXN];
6a6d     bool rev[MAXN];
427e
7839     bool isroot(int x) {
45a9         return ch[fa[x]][0] == x || ch[fa[x]][1] == x;
95cf     }
427e
3bf9     void pull(int x) {
6664         sum[x] = val[x] ^ sum[ch[x][0]] ^ sum[ch[x][1]];
95cf     }
427e
3698     void reverse(int x) {
7850         swap(ch[x][0], ch[x][1]);
52c6         rev[x] ^= 1;
95cf     }
427e
1a53     void push(int x) {
8f1f         if (rev[x]) {
ebf3             if (ch[x][0]) reverse(ch[x][0]);
6eb0             if (ch[x][1]) reverse(ch[x][1]);
8fc1             rev[x] = 0;
95cf         }
95cf     }
427e

```

```

void rotate(int x) {
    int y = fa[x], z = fa[y], k = ch[y][1] == x, w = ch[x][!k];
    if (isroot(y)) ch[z][ch[z][1] == y] = x;
    ch[x][!k] = y; ch[y][k] = w;
    if (w) fa[w] = y;
    fa[y] = x; fa[x] = z;
    pull(y);
}

void pushall(int x) {
    if (isroot(x)) pushall(fa[x]);
    push(x);
}

void splay(int x) {
    int y = x, z = 0;
    pushall(y);
    while (isroot(x)) {
        y = fa[x]; z = fa[y];
        if (isroot(y)) rotate((ch[y][0] == x) ^ (ch[z][0] == y) ? x : y);
        rotate(x);
    }
    pull(x);
}

void access(int x) {
    int z = x;
    for (int y = 0; x; x = fa[y = x]) {
        splay(x);
        ch[x][1] = y;
        pull(x);
    }
    splay(z);
}

void chroot(int x) {
    access(x);
    reverse(x);
}

void split(int x, int y) {
    chroot(x);
    access(y);
}

```

```

425f
51af
e1fe
af46
fa6f
3540
72ef
95cf
427e
bc1b
a316
a97b
95cf
427e
f69c
d095
8ab3
f244
ceef
4449
cf90
95cf
78a0
95cf
427e
6229
1548
ba78
8fec
b05d
78a0
95cf
7afd
95cf
427e
502e
766a
cb0d
95cf
427e
471a
3015
29b5
95cf

```

```

427e int Root(int x) {
d87a     access(x);
766a     while (ch[x][0]) {
874d         push(x);
a97b         x = ch[x][0];
b83a     }
95cf     splay(x);
8fec     return x;
d074 }
95cf
427e void Link(int u, int v) { // assume unconnected before
70d3     chroot(u);
b8a5     fa[u] = v;
2448 }
95cf
427e void Cut(int u, int v) { // assume connected before
c2f4     split(u, v);
e8ce     fa[u] = ch[v][0] = 0;
fd95     pull(v);
743b }
95cf
427e int Query(int u, int v) {
6ca2     split(u, v);
e8ce     return sum[v];
a5ba }
95cf
427e void Update(int u, int x) {
eaba     splay(u);
46ce     val[u] = x;
1d62 }
95cf }
329b };

```

6.5 Balanced binary search tree from pb_ds

```

0475 #include <ext/pb_ds/assoc_container.hpp>
332d using namespace __gnu_pbds;
427e
43a7 tree<int, null_type, less<int>, rb_tree_tag, tree_order_statistics_node_update>
    rkt;
427e // null_tree_node_update
427e

```

```

// 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;

    void* operator new(size_t size);

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

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

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

```



```

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);
95cf     }
2b3e     a->value++;
5ffd     return a;
95cf   }
427e
e80f   node *inc(int index) {
c246     return Inc(0, n, index);
95cf   }
865a   } nodes[8000000];
427e
99ce   int node::n, node::pos;
1987   inline void* node::operator new(size_t size) {
bb3c     return nodes + (pos++);
95cf   }

```

6.7 Sparse table, range extremum query

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

```

db63   const int MAXN = 100007;
b330   int a[MAXN];
69ae   int st[MAXN][32 - __builtin_clz(MAXN)];
427e
8041   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]);
}

```

7 Geometrics

7.1 2D geometric template

```

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

typedef int T;
typedef struct pt {
    T x, y;
    T operator , (pt a) { return x*a.x + y*a.y; } // inner product
    T operator * (pt a) { return x*a.y - y*a.x; } // outer product
    pt operator + (pt a) { return {x+a.x, y+a.y}; }
    pt operator - (pt a) { return {x-a.x, y-a.y}; }

    pt operator * (T k) { return {x*k, y*k}; }
    pt operator - () { return {-x, -y}; }
} vec;

typedef pair<pt, pt> seg;

bool ptOnSeg(pt& p, seg& s){
    vec v1 = s.first - p, v2 = s.second - p;
    return (v1, v2) <= 0 && v1 * v2 == 0;
}

// 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
7538 inline double rotOrder(vec a, vec b, vec c){return double(a*b)*(b*c);}
427e
31ed inline bool intersect(seg a, seg b){
427e     // ! if (nIntRectRect(a, b)) return false; // if commented, assume that a
        and b are non-collinear
cb52     return rotOrder(b.first-a.first, a.second-a.first, b.second-a.first) >= 0 &&
059e         rotOrder(a.first-b.first, b.second-b.first, a.second-b.first) >= 0;
95cf }
427e
427e // 0 not intersect
427e // 1 standard intersection
427e // 2 vertex-line intersection
427e // 3 vertex-vertex intersection
427e // 4 collinear and have common point(s)
4d19 int intersect2(seg& a, seg& b){
5dc4     if (nIntRectRect(a, b)) return 0;
42c0     vec va = a.second - a.first, vb = b.second - b.first;
2096     double j1 = rotOrder(b.first-a.first, va, b.second-a.first),
72fe         j2 = rotOrder(a.first-b.first, vb, a.second-b.first);
5ac6     if (j1 < 0 || j2 < 0) return 0;
9400     if (j1 != 0 && j2 != 0) return 1;
83db     if (j1 == 0 && j2 == 0){
6b0c         if (va * vb == 0) return 4; else return 3;
fb17     } else return 2;

```

```

}

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

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

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

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

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

```

```

95cf
427e
2c68
5894
6850
7c9a
95cf
427e
427e
427e
427e
cbdd
5fb4
1294
427e
3cae
b957
8c40
3c4d
aad3
95cf
0a5f
95cf
427e
d4a3
fa86
331a
95cf
427e
07ae
5cab
331a
95cf

```

8 Appendices

8.1 Primes

8.1.1 First primes

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

8.1.2 Arbitrary length primes

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

8.1.3 $\sim 1 \times 10^9$

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

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

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

8.2 Pell's equation

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

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

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

Some smallest solutions to Pell's equation:

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

8.3 Burnside's lemma and Polya's enumeration theorem

The Burnside's lemma says that

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

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

The unweighted version of Pólya enumeration theorem says that

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

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

8.4 Lagrange's interpolation

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

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