

南京大学 ACM-ICPC 集训队代码模版库



Contents

| | | | |
|---|-----------|--|-----------|
| 1 General | 3 | | |
| 1.1 Code library checksum | 3 | | |
| 1.2 .vimrc | 3 | | |
| 1.3 Template | 3 | | |
| 2 Miscellaneous Algorithms | 3 | | |
| 2.1 2-SAT | 3 | | |
| 2.2 Knuth's optimization | 4 | | |
| 3 String | 4 | | |
| 3.1 Knuth-Morris-Pratt algorithm | 4 | | |
| 3.2 Manacher algorithm | 5 | | |
| 3.3 Aho-corasick automaton | 5 | | |
| 3.4 Trie | 6 | | |
| 4 Linear Algebra | 7 | | |
| 4.1 Matrix powermod | 7 | | |
| 4.2 Linear basis | 7 | | |
| 4.3 Gauss elimination over finite field | 7 | | |
| 4.4 Berlekamp-Massey algorithm | 8 | | |
| 4.5 Fast Walsh-Hadamard transform | 9 | | |
| 4.6 Fast fourier transform | 9 | | |
| 4.7 Number theoretic transform | 10 | | |
| 5 Number Theory | 11 | | |
| 5.1 Sieve of Euler | 11 | 5.2 Miller-Rabin primality test | 11 |
| | | 5.3 Pollard's rho algorithm | 11 |
| | | 6 Graph Theory | 12 |
| | | 6.1 Strongly connected component | 12 |
| | | 6.2 Vertex biconnected component | 12 |
| | | 6.3 Minimum spanning arborescence (Chu-Liu) | 13 |
| | | 6.4 Maximum flow (Dinic) | 14 |
| | | 6.5 Maximum cardinality bipartite matching (Hungarian) | 15 |
| | | 6.6 Minimum cost maximum flow | 15 |
| | | 6.7 Global minimum cut (Stoer-Wagner) | 16 |
| | | 6.8 Heavy-light decomposition | 17 |
| | | 7 Data Structures | 17 |
| | | 7.1 Segment tree | 17 |
| | | 7.2 Link/cut tree | 18 |
| | | 7.3 Balanced binary search tree from pb_ds | 20 |
| | | 7.4 Persistent segment tree, range k-th query | 20 |
| | | 7.5 Sparse table, range extremum query | 21 |
| | | 8 Appendices | 21 |
| | | 8.1 Primes | 21 |
| | | 8.1.1 First primes | 21 |
| | | 8.1.2 Arbitrary length primes | 21 |
| | | 8.1.3 $\sim 1 \times 10^9$ | 22 |
| | | 8.1.4 $\sim 1 \times 10^{18}$ | 22 |
| | | 8.2 Pell's equation | 22 |
| | | 8.3 Burnside's lemma and Polya's enumeration theorem | 22 |

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 .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,<,>[,]
```

1.3 Template

```
302f #include <bits/stdc++.h>
421c using namespace std;
427e
426f #ifdef __LOCAL_DEBUG__
3341 # define _debug(fmt, ...) fprintf(stderr, "[%s]\n" fmt "\n", \
611f     __func__, ##__VA_ARGS__)
a8cb #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;}
```

```
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

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

```
0f42
03a9
5c83
8f72
d060
```

```

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             }
95cf         }
3361         return true;
95cf     }
427e
5f0a     inline bool value(unsigned i){return mark[2*i+1];}
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

```

3 String

3.1 Knuth-Morris-Pratt algorithm

```

const int SIZE = 10005;
int fail[SIZE];
int len;

void construct(const char* p) {
    len = strlen(p);
    fail[0] = fail[1] = 0;
}

```

```

2836
9847
57b7
427e
182f
aaa1
3dd4

```

```

d8a8  for (int i = 1; i < len; i++) {
147f    int j = fail[i];
3c79    while (j && p[i] != p[j]) j = fail[j];
4643    fail[i + 1] = p[i] == p[j] ? j + 1 : 0;
95cf  }
95cf  }
427e
c464 inline void found(int pos) {
427e     // ! add codes for having found at pos
95cf }
427e
1932 void match(const char* t, const char* p) { // must be called after construct
8482     int n = strlen(t);
8fd0     int j = 0;
be8e     rep(i, n) {
4e19         while (j && p[j] != t[i]) j = fail[j];
b5d5         if (p[j] == t[i]) j++;
f024         if (j == len) found(i - len + 1);
95cf     }
95cf }

```

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     }

```

```

}

void init(const char *tt) {
    int len = strlen(tt);
    s.resize(len * 2 + 10);
    lc.resize(len * 2 + 10);
    s[0] = '*';
    s[1] = '#';
    for (int i = 0; i < len; i++) {
        s[i * 2 + 2] = tt[i];
        s[i * 2 + 1] = '#';
    }
    s[len * 2 + 1] = '#';
    s[len * 2 + 2] = '\0';
    Len = len * 2 + 2;
    work();
}

pair<int, int> maxpal(int l, int r) {
    int center = l + r + 1;
    int rad = lc[center] / 2;
    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)};
}
};

```

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);
            }
        }
    }

```

```

95cf
427e
bfd5
aaaf
f701
7045
8e13
ae54
1321
e995
69fd
95cf
43fd
75d1
61f7
3e7a
95cf
427e
b194
901a
ffb2
ab54
17e4
3908
69f3
69dc
95cf
329b

```

```

a1ad
9143
daca
427e
8690
93d2
a7a6
ce3c
b1c6
a506
3e14

```

```

f689     last[u] = 0;
95cf     }
95cf     }
cc78     while (!q.empty()) {
31f0         int r = q.front();
15dd         q.pop();
ce3c         rep(c, CHARN) {
ab59             int u = tr[r][c];
0ef5             if (!u) {
9d58                 tr[r][c] = tr[fail[r]][c];
b333                 continue;
95cf             }
3e14             q.push(u);
b3ff             int v = fail[r];
d2ea             while (v && !tr[v][c]) v = fail[v];
c275             fail[u] = tr[v][c];
654c             last[u] = tag[fail[u]] ? fail[u] : last[fail[u]];
95cf         }
95cf     }
95cf }
427e
7752 void found(int pos, int j) {
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

```

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
    void add(const char* s, int t) {
        int p = 0, c, len = strlen(s);
        rep(i, len) {
            c = id(s[i]);
            if (!tr[p][c]) {
                memset(tr[n], 0, sizeof(tr[n]));
                tag[n] = 0;
                tr[p][c] = n++;
            }
            p = tr[p][c];
        }
        tag[p] = t;
    }

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

```

4 Linear Algebra

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){
fc9f                     t[i][j] += (m[i][k] * a.m[k][j]) % modular;
199e                     t[i][j] %= modular;
95cf                 }
95cf             }
95cf         }
dad4         memcpy(m, t, sizeof(t));
95cf     }
329b };
427e
63d8 matrix r;
3ec2 void m_powmod(matrix& b, LL e){
83f0     memset(r.m, 0, sizeof(r.m));
a7c3     Rep(i, n)
de64         r.m[i][i] = 1;
3e90     while (e){
5a0e         if (e & 1) r *= b;
35c5         b *= b;
16fc         e >>= 1;
95cf     }
95cf }

```

4.2 Linear basis

```

8b44 const int MAXD = 30;
03a6 struct linearbasis {

```

```

ULL b[MAXD] = {};

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

```

4.3 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> WLL;

```

```

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

    rep (i, n) {

```

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

b784
427e
2a2c
95a2
3e90
1783
5549
16fc
95cf
547e
95cf
427e
c130
42ac
427e
2c62
561b
a25e
2976
427e
be8e

```

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

```

4.4 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;

```

```

        b = b * b % MOD;
        e >>= 1;
    }
    return r;
}

struct Poly {
    vector<int> a;

    Poly() { a.clear(); }

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

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

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

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

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

Poly operator * (const int &c, const Poly &p) {
    vector<int> na(p.length());

```

```

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
95cf
427e
c856
bd55
d1a7
3507
2bee
9526
95cf
cecf
95cf
329b
427e
5473
72de

```



```

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

```

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){

```

```

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;

    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;

```

5595
05f2
b833
7796
427e
427e
427e
95cf
95cf
427e
2ab6
950a
e427
8a42
430f
95cf

4e09
427e
3fbf
427e
abd1
12af
c47c
27d7
9827
427e
1442
e209
b393
7ba3
1908
a166
95cf
95cf
427e
b941
a215
ac6e
2969

```

b3cf      for (cplx* p = a; p != a + N; p += 1)
c24f          rep (k, m){
fe06              cplx t = w[N/l*k] * p[k+m];
ecbf              p[k+m] = p[k] - t; p[k] += t;
95cf          }
95cf      }
95cf  }
427e
617b  void fft(cplx* a){dft(a, omega);}
a123  void ifft(cplx* a){
3b2f      dft(a, oinv);
57fc      rep (i, N) a[i] /= N;
95cf  }
427e
bdc0  void conv(cplx* a, cplx* b){
6497      fft(a); fft(b);
12a5      rep (i, N) a[i] *= b[i];
f84e      ifft(a);
95cf  }
329b };

```

4.7 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;

```

```

}
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);}
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);
}
};

```

```

95cf
427e
f420
e209
7652
4b3a
e04f
b393
7ba3
ad4f
8d8b
9e14
95cf
95cf
95cf
427e
9668
a215
ac6e
2969
7a1d
c24f
0ad3
6209
fa1b
95cf
95cf
95cf
427e
92ea
5daf
1f2a
9910
a873
95cf
427e
3a5b
ad16
e49e
5748
95cf
329b

```

5 Number Theory

5.1 Sieve of Euler

```

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

5.2 Miller-Rabin primality test

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

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

5.3 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;

```

2e6b
427e
54a5
45eb
d3e5
3c69
0964

```

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 }

```

6 Graph Theory

6.1 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]){

```

```

sccn++;
int x;
do {
    x = s.top(); s.pop();
    scc[x] = sccn;
} while (x != u);
}
}

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

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

};

```

6.2 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);
    }
}

```

```

427e int dfs(int u, int fa) {
7d3c     int lowu = pre[u] = ++dfs_clock;
9fe6     int child = 0;
ec14     for (int v : adj[u]) {
18f6         if (!pre[v]) {
173e             s.push({u, v});
e7f8             child++;
fdcf             int lowv = dfs(v, u);
f851             lowu = min(lowu, lowv);
189c             if (lowv >= pre[u]) {
b687                 iscut[u] = 1;
6323                 bcc[bcc_cnt].clear();
57eb                 bcce[bcc_cnt].clear();
90b8                 while (1) {
a147                     int xu, xv;
a6a3                     tie(xu, xv) = s.top(); s.pop();
a0c3                     bcce[bcc_cnt].insert({min(xu, xv), max(xu, xv)});
0ef5                     if (bccno[xu] != bcc_cnt) {
3db2                         bcc[bcc_cnt].push_back(xu);
e0db                         bccno[xu] = bcc_cnt;
d27f                     }
95cf                     if (bccno[xv] != bcc_cnt) {
f357                         bcc[bcc_cnt].push_back(xv);
752b                         bccno[xv] = bcc_cnt;
57c9                     }
95cf                     if (xu == u && xv == v) break;
7096                 }
95cf                 bcc_cnt++;
03f5             }
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 void find_bcc(int n) {
17be     memset(pre, 0, sizeof pre);
8c2f     memset(iscut, 0, sizeof iscut);
e2d2     memset(bccno, -1, sizeof bccno);
40d3     dfs_clock = bcc_cnt = 0;
fae2

```

```

        rep (i, n) if (!pre[i]) dfs(i, -1);
    }
};

```

5c63
95cf
329b

6.3 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];
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;
            }
        }
    }
    return ans;
}

```

bcf8
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      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  }

```

6.4 Maximum flow (Dinic)

Usage:

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

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];
dded     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);

```

```

}
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);
            }
        }
    }
    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));

```

```
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 };
```

6.5 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     }
```

```
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){
        fill(range(mark), false);
        if (augment(i)) ret++;
    }
    return ret;
}

};
```

6.6 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
```

```

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;
ddff          rep (i, G[u].size()){
c234              edge& e = edges[G[u][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  #ifndef GIVEN_FLOW
5972      bool min_cost(int s, int t, int f, LL& cost) {
590d          this->s = s; this->t = t;

```

```

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

```

21d4
23cb
22dc
bcd b
a671
9c87
3361
8e2e
2a83
95cf
95cf
438e
95cf
a8cb
f9a9
590d
21d4
23cb
22dc
bcd b
2a83
95cf
84fb
95cf
1937
329b

6.7 Global minimum cut (Stoer-Wagner)

```

typedef vector<LL> VI;
typedef vector<VI> VWI;

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;

```

f9d7
045e
427e
f012
66f7
4d98
329d
427e
cd21
ec6e
f20e
4b32
8bfc


```

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 }
038c return {bestw, bestc};
95cf }

```

6.8 Heavy-light decomposition

```

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     }

```

```

son[x] = s;
}

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

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

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

```

7 Data Structures

7.1 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)

```

```

1294 #define rson (o<<1|1)
427e
1344 void pull(int o) {
bbe9     val[o] = (val[lson] + val[rson]) % p;
95cf }
427e
e4bc void push_add(int o, LL x) {
5dd6     val[o] = (val[o] + x * (r[o] - l[o])) % p;
6eff     tadd[o] = (tadd[o] + x) % p;
95cf }
427e
d658 void push_mul(int o, LL x) {
b82c     val[o] = val[o] * x % p;
aa86     tadd[o] = tadd[o] * x % p;
649f     tmul[o] = tmul[o] * x % p;
95cf }
427e
b149 void push(int o) {
3159     if (l[o] == m[o]) return;
0a90     if (tmul[o] != 1) {
0f4a         push_mul(lson, tmul[o]);
045e         push_mul(rson, tmul[o]);
ac0a         tmul[o] = 1;
95cf     }
1b82     if (tadd[o]) {
9547         push_add(lson, tadd[o]);
0e73         push_add(rson, tadd[o]);
6234         tadd[o] = 0;
95cf     }
95cf }
427e
471c void build(int o, int ll, int rr) {
0e87     int mm = (ll + rr) / 2;
9d27     l[o] = ll; r[o] = rr; m[o] = mm;
ac0a     tmul[o] = 1;
5c92     if (ll == rr) {
001f         scanf("%lld", val + o);
e5b6         val[o] %= p;
8e2e     } else {
7293         build(lson, ll, mm);
5e67         build(rson, mm, rr);
ba26         pull(o);
95cf     }
95cf }

```

```

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

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

LL query(int o, int ll, int rr) {
    if (ll <= l[o] && r[o] <= rr) {
        return val[o];
    } else {
        LL ans = 0;
        push(o);
        if (m[o] > ll) ans += query(lson, ll, rr);
        if (m[o] < rr) ans += query(rson, ll, rr);
        return ans % p;
    }
}
} seg;

```

427e
4406
3c16
db32
8e2e
c4b0
4305
d5a6
ba26
95cf
95cf
427e
48cd
3c16
e7d0
8e2e
c4b0
d1ba
67f3
ba26
95cf
95cf
427e
0f62
3c16
6dfe
8e2e
f7ff
c4b0
c5f8
ef81
a420
95cf
95cf
4d99

7.2 Link/cut tree

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

```

namespace LCT {
    const int MAXN = 300005;

```

427e
427e
ed4d
5ece

```

6a6d  int fa[MAXN], ch[MAXN][2], val[MAXN], sum[MAXN];
c6e1  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
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);

```

```

        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);
}

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);
}

```

```

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
d87a
766a
874d
a97b
b83a
95cf
8fec
d074
95cf
427e
70d3
b8a5
2448
95cf
427e
c2f4
e8ce
fd95
743b

```

```

95cf    }
427e
6ca2    int Query(int u, int v) {
e8ce        split(u, v);
a5ba        return sum[v];
95cf    }
427e
eaba    void Update(int u, int x) {
46ce        splay(u);
1d62        val[u] = x;
95cf    }
329b    };

```

7.3 Balanced binary search tree from pb_ds

```

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

```

7.4 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);

```

```

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) {
        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++;
}

```

427e
3dc0
b6c5
ce96
181e
3ba2
8aaf
8e2e
bfc4
95cf
5ffd
95cf
427e
5a45
2c46
7ee3
be52
95cf
427e
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     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     }

```

7.5 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
427e
d34f     void init(int n){
ce01         int l = 31 - __builtin_clz(n);
cf75         rep (i, n) st[i][0] = a[i];
b811         rep (j, l)
6937             rep (i, 1+n-(1<<j))
082a                 st[i][j+1] = ext(st[i][j], st[i+(1<<j)][j]);
95cf     }
427e
c863     int rmq(int l, int r){
92f5         int k = 31 - __builtin_clz(r-l+1);
baa2         return ext(st[l][k], st[r-(1<<k)+1][k]);
95cf     }

```

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 .