南京大学 ACM-ICPC 集训队代码模版库

Contents

1 General

1.1 Code library checksum

```
ab14 #!/usr/bin/python3
c502 import re, sys, hashlib
427e
f7db for line in sys.stdin.read().strip().split("\n") :
ddf5     print(hashlib.md5(re.sub(r'\s|//.*', '', line).encode('utf8')).hexdigest()
           [-4:], line)
```

1.2 .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]␣" fmt "\n", \
611f     __func__, ##__VA_ARGS__)
a8cb #else
e6b5 # define _debug(...) ((void) 0)
1937 #endif
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;
b773 typedef unsigned long long ULL;
427e
5120 template <unsigned p>
87b8 struct Zp{
7797     unsigned x;
ff67     Zp(unsigned x):x(x){}
22e3     operator unsigned(){return x;}
fecc     Zp operator ^ (ULL e) {
4fce         Zp b=x, r=1;
3e90         while (e) {
5421             if (e&1) r=r*b;
2059             b=b*b;
16fc             e>>=1;
95cf         }
547e         return r;
95cf     }
a2f5     Zp operator + (Zp rhs) {return (x+rhs)%p;}
664b     Zp operator - (Zp rhs) {return (x+p-rhs)%p;}
3ec4     Zp operator * (Zp rhs) {return x*rhs%p;}
7cfd     Zp operator / (Zp rhs) {return Zp(x)*(rhs^(p-2));}
329b };
427e
370f typedef Zp<1000000007> zp;
427e
0795 zp operator"" _ (ULL n){return n;}
```

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;
```

```
    void init(int n){
        this->n = n;
        for (int i=0; i<n*2; i++) G[i].clear();
        memset(mark, 0, sizeof(mark));
    }

    bool dfs(int x){
        if (mark[x^1]) return false;
        if (mark[x]) return true;
        mark[x] = true;
        S[c++] = x;
        for (int i=0; i<G[x].size(); i++)
            if (!dfs(G[x][i])) return false;
        return true;
    }

    void add_clause(int x, bool xval, int y, bool yval){
        x = x * 2 + xval;
        y = y * 2 + yval;
        G[x^1].push_back(y);
        G[y^1].push_back(x);
    }

    bool solve() {
        for (int i=0; i<n*2; i+=2){
            if (!mark[i] && !mark[i+1]){
                c = 0;
                if (!dfs(i)){
                    while (c > 0) mark[S[--c]] = false;
                    if (!dfs(i+1)) return false;
                }
            }
        }
        return true;
    }

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

2.2 Knuth's optimization

```
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;
        }
      }
    }
  }
};
```

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;
  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     }
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;
17e4     int rl = rmid - rad, rr = rmid + rad - 1;
3908     if ((r ^ l) & 1) {
69f3     } else rr++;
69dc     return {max(l, rl), min(r, rr)};
95cf   }
329b };
```

3.3 Aho-corasick automaton

```
a1ad struct AC : Trie {
9143   int fail[MAXN];
daca   int last[MAXN];
427e
8690   void construct() {
93d2     queue<int> q;
a7a6     fail[0] = 0;
ce3c     rep(c, CHARN) {
b1c6       if (int u = tr[0][c]) {
a506         fail[u] = 0;
3e14         q.push(u);
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

```
e6f1const int MAXN = 12000;
dd87const int CHARN = 26;
427e
8ff5inline int id(char c) { return c - 'a'; }
427e
a281struct Trie {
5c83  int n;
f4f5  int tr[MAXN][CHARN];  // Trie tree, 0 denotes fail
35a5  int tag[MAXN];
427e
4feeTrie() {
3cccmemset(tr[0], 0, sizeof(tr[0]));
4d52tag[0] = 0;
46bfn = 1;
95cf}
427e
427e// tag should not be 0
30b0void add(const char* s, int t) {
d50aint p = 0, c, len = strlen(s);
9c94rep(i, len) {
3140c = id(s[i]);
d6c8if (!tr[p][c]) {
26ddmemset(tr[n], 0, sizeof(tr[n]));
2e5ctag[n] = 0;
73bbtr[p][c] = n++;
95cf}
f119p = tr[p][c];
95cf}
35eftag[p] = t;
95cf}
427e
427e// returns 0 if not found
427e// AC automaton does not need this function
216cint search(const char* s) {
d50aint p = 0, c, len = strlen(s);
9c94rep(i, len) {
3140c = id(s[i]);
f339if (!tr[p][c]) return 0;
f119p = tr[p][c];
95cf}
840ereturn tag[p];
95cf}
329b};
```

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

4.2 Linear basis

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

4.3 Gauss elimination over finite field

```
b784 const LL p = 1000000007;
427e
2a2c LL powmod(LL b, LL e) {
95a2   LL r = 1;
3e90   while (e) {
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> VVLL;
427e
2c62 LL gauss(VVLL &a, VVLL &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
37e1   for (int j = n - 1; j >= 0; j--) if (irow[j] != icol[j]) {
50dc     for (int k = 0; k < n; k++) swap(a[k][irow[j]], a[k][icol[j]]);
95cf   }
f27f   return det;
95cf }
```

4.4 Berlekamp-Massey algorithm

```
2b86 const LL MOD = 1000000007;
427e
391d LL inverse(LL b) {
32d3   LL e = MOD - 2, r = 1;
3e90   while (e) {
9a62     if (e & 1) r = r * b % MOD;
29ea     b = b * b % MOD;
16fc     e >>= 1;
95cf   }
547e   return r;
95cf }
427e
32a6 struct Poly {
afe0   vector<int> a;
427e
9794   Poly() { a.clear(); }
427e
de81   Poly(vector<int> &a) : a(a) {}
427e
8087   int length() const { return a.size(); }
427e
16de   Poly move(int d) {
b31d     vector<int> na(d, 0);
f915     na.insert(na.end(), a.begin(), a.end());
cecf     return Poly(na);
95cf   }
427e
fa1a   int calc(vector<int> &d, int pos) {
5b57     int ret = 0;
501c     for (int i = 0; i < (int)a.size(); ++i) {
5de5       if ((ret += (long long)d[pos - i] * a[i] % MOD) >= MOD) {
3041         ret -= MOD;
95cf       }
95cf     }
95cf     return ret;
ee0f   }
95cf
427e   Poly operator - (const Poly &b) {
c856     vector<int> na(max(this->length(), b.length()));
bd55     for (int i = 0; i < (int)na.size(); ++i) {
d1a7       int aa = i < this->length() ? this->a[i] : 0,
3507           bb = i < b.length() ? b.a[i] : 0;
2bee       na[i] = (aa + MOD - bb) % MOD;
9526     }
95cf     return Poly(na);
cecf   }
95cf };
329b
427e Poly operator * (const int &c, const Poly &p) {
5473   vector<int> na(p.length());
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){
5595     for (int d = 1; d < n; d <<= 1)
05f2         for (int i = 0; i < n; i += d << 1)
b833             rep (j, d){
7796                 int x = a[i+j], y = a[i+j+d];
427e                 // a[i+j] = (x+y)/2, a[i+j+d] = (x-y)/2;    // xor
427e                 // a[i+j] = x-y;                            // and
427e                 // a[i+j+d] = y-x;                          // or
95cf             }
95cf }
427e
2ab6 void conv(int* a, int* b, int n){
950a     fwt(a, n);
e427     fwt(b, n);
8a42     rep(i, n) a[i] *= b[i];
430f     ifwt(a, n);
95cf }
```

4.6 Fast fourier transform

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

```
b3cf            for (cplx* p = a; p != a + N; p += l)
c24f                rep (k, m){
fe06                    cplx t = w[N/l*k] * p[k+m];
ecbf                    p[k+m] = p[k] - t; p[k] += t;
95cf                }
95cf        }
95cf    }
427e
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;
95cf     }
427e
f420     NTT(int k){
e209         K = k; N = 1 << k;
7652         g = powmod(G, (P-1)/N);
4b3a         g_inv = powmod(g, N-1);
e04f         omega[0] = oinv[0] = 1;
b393         rep (i, N){
7ba3             rev[i] = (rev[i>>1]>>1) | ((i&1)<<(K-1));
ad4f             if (i){
8d8b                 omega[i] = omega[i-1] * g % P;
9e14                 oinv[i] = oinv[i-1] * g_inv % P;
95cf             }
95cf         }
95cf     }
427e
9668     void _ntt(LL* a, LL* w){
a215         rep (i, N) if (i < rev[i]) swap(a[i], a[rev[i]]);
ac6e         for (int l = 2; l <= N; l *= 2){
2969             int m = l/2;
7a1d             for (LL* p = a; p != a + N; p += l)
c24f                 rep (k, m){
0ad3                     LL t = w[N/l*k] * p[k+m] % P;
6209                     p[k+m] = (p[k] - t + P) % P;
fa1b                     p[k] = (p[k] + t) % P;
95cf                 }
95cf         }
95cf     }
427e
92ea     void ntt(LL* a){_ntt(a, omega);}
5daf     void intt(LL* a){
1f2a         LL inv = powmod(N, P-2);
9910         _ntt(a, oinv);
a873         rep (i, N) a[i] = a[i] * inv % P;
95cf     }
427e
3a5b     void conv(LL* a, LL* b){
ad16         ntt(a); ntt(b);
e49e         rep (i, N) a[i] = a[i] * b[i] % P;
5748         intt(a);
95cf     }
329b};
```

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     }
   }
}
```

5.2 Miller-Rabin primality test

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

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

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

5.3 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 }
```

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

6.2 Vertex biconnected component

```
0f42 const int MAXN = 100005;
2ea0 struct graph {
33ae     int pre[MAXN], iscut[MAXN], bccno[MAXN], dfs_clock, bcc_cnt;
848f     vector<int> adj[MAXN], bcc[MAXN];
6b06     set<pair<int, int>> bcce[MAXN];
427e
76f7     stack<pair<int, int>> s;
427e
bfab     void add_edge(int u, int v) {
c71a         adj[u].push_back(v);
a717         adj[v].push_back(u);
95cf     }
427e
```

```
7d3c    int dfs(int u, int fa) {
9fe6        int lowu = pre[u] = ++dfs_clock;
ec14        int child = 0;
18f6        for (int v : adj[u]) {
173e            if (!pre[v]) {
e7f8                s.push({u, v});
fdcf                child++;
f851                int lowv = dfs(v, u);
189c                lowu = min(lowu, lowv);
b687                if (lowv >= pre[u]) {
6323                    iscut[u] = 1;
57eb                    bcc[bcc_cnt].clear();
90b8                    bcce[bcc_cnt].clear();
a147                    while (1) {
a6a3                        int xu, xv;
a0c3                        tie(xu, xv) = s.top(); s.pop();
0ef5                        bcce[bcc_cnt].insert({min(xu, xv), max(xu, xv)});
3db2                        if (bccno[xu] != bcc_cnt) {
e0db                            bcc[bcc_cnt].push_back(xu);
d27f                            bccno[xu] = bcc_cnt;
95cf                        }
f357                        if (bccno[xv] != bcc_cnt) {
752b                            bcc[bcc_cnt].push_back(xv);
57c9                            bccno[xv] = bcc_cnt;
95cf                        }
7096                        if (xu == u && xv == v) break;
95cf                    }
03f5                    bcc_cnt++;
95cf                }
7470            } else if (pre[v] < pre[u] && v != fa) {
e7f8                s.push({u, v});
f115                lowu = min(lowu, pre[v]);
95cf            }
95cf        }
e104        if (fa < 0 && child == 1) iscut[u] = 0;
1160        return lowu;
95cf    }
427e
17be    void find_bcc(int n) {
8c2f        memset(pre, 0, sizeof pre);
e2d2        memset(iscut, 0, sizeof iscut);
40d3        memset(bccno, -1, sizeof bccno);
fae2        dfs_clock = bcc_cnt = 0;
5c63        rep (i, n) if (!pre[i]) dfs(i, -1);
    }
};
```

6.3 Minimum spanning arborescence (Chu-Liu)

Usage: `getans(n, root, edges)`
All vertices are 1-based.

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

6.4 Maximum flow (Dinic)

```
bcf8struct edge{
60e2    int from, to;
5e6d    LL cap, flow;
329b};
427e
e2cdconst int MAXN = 1005;
9062struct Dinic {
4dbf    int n, m, s, t;
9f0c    vector<edge> edges;
b891    vector<int> G[MAXN];
bbb6    bool vis[MAXN];
b40a    int d[MAXN];
ddec    int cur[MAXN];
427e
5973    void add_edge(int from, int to, LL cap) {
7b55        edges.push_back(edge{from, to, cap, 0});
1db7        edges.push_back(edge{to, from, 0, 0});
fe77        m = edges.size();
dff5        G[from].push_back(m-2);
8f2d        G[to].push_back(m-1);
95cf    }
427e
1836    bool bfs() {
3b73        memset(vis, 0, sizeof(vis));
93d2        queue<int> q;
5d13        q.push(s);
2cd2        vis[s] = 1;
721d        d[s] = 0;
cc78        while (!q.empty()) {
66ba            int x = q.front(); q.pop();
3b61            for (int i = 0; i < G[x].size(); i++) {
b510                edge& e = edges[G[x][i]];
bba9                if (!vis[e.to] && e.cap > e.flow) {
cd72                    vis[e.to] = 1;
cf26                    d[e.to] = d[x] + 1;
ca93                    q.push(e.to);
95cf                }
95cf            }
95cf        }
b23b        return vis[t];
95cf}
427e
9252LL 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
5bf2LL 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}
427ec72evector<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>
421cusing 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()
5cadtypedef long long LL;
427e
84eestruct Hungarian{
fbf6    int nx, ny;
9ec6    vector<int> mx, my;
9d4c    vector<vector<int> > e;
edec    vector<bool> mark;
427e
8324    void init(int nx, int ny){
c1d1        this->nx = nx;
f9c1        this->ny = ny;
ac92        mx.resize(nx); my.resize(ny);
3f11        e.clear(); e.resize(nx);
1023        mark.resize(nx);
95cf    }
427e
4589    inline void add(int a, int b){
486c        e[a].push_back(b);
95cf    }
427e
0c2b    bool augment(int i){
207c        if (!mark[i]) {
dae4            mark[i] = true;
6a1e            for (int j : e[i]){
0892                if (my[j] == -1 || augment(my[j])){
9ca3                    mx[i] = j; my[j] = i;
3361                    return true;
95cf                }
95cf            }
95cf        }
95cf        return false;
438e    }
95cf
427e    int match(){
3fac        int ret = 0;
5b57        fill(range(mx), -1);
b0f1        fill(range(my), -1);
b957        rep (i, nx){
4ed1            fill(range(mark), false);
13a5            if (augment(i)) ret++;
cc89        }
95cf        return ret;
ee0f    }
95cf};
329b
```

6.6 Minimum cost maximum flow

```
bcf8struct edge{
60e2    int from, to;
d698    int cap, flow;
32cc    LL cost;
329b};
427e
cc3econst LL INF = LLONG_MAX / 2;
2aa8const int MAXN = 5005;
c6cbstruct 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;
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#ifdef GIVEN_FLOW
5972    bool min_cost(int s, int t, int f, LL& cost) {
590d        this->s = s; this->t = t;
21d4        int flow = 0;
23cb        cost = 0;
22dc        while (spfa()) {
bcdb            augment();
a671            if (flow + a[t] >= f){
9c87                cost += (f - flow) * a[t]; flow = f;
3361                return true;
8e2e            } else {
2a83                flow += a[t]; cost += a[t] * d[t];
95cf            }
95cf        }
95cf        return false;
438e    }
95cf#else
a8cb    int min_cost(int s, int t, LL& cost) {
f9a9        this->s = s; this->t = t;
590d        int flow = 0;
21d4        cost = 0;
23cb        while (spfa()) {
22dc            augment();
bcdb            flow += a[t]; cost += a[t] * d[t];
2a83        }
95cf        return flow;
84fb    }
95cf#endif
1937};
329b
```

6.7 Global minimum cut (Stoer-Wagner)

```
f9d7typedef vector<LL> VI;
045etypedef vector<VI> VVI;
427e
f012pair<LL, VI> stoer(VVI &w) {
66f7    int n = w.size();
4d98    VI used(n), c, bestc;
329d    LL bestw = -1;
427e
cd21    for (int ph = n - 1; ph >= 0; ph--) {
ec6e        VI wt = w[0], added = used;
f20e        int prev, last = 0;
4b32        rep (i, ph) {
8bfc            prev = last;
0706            last = -1;
4942            for (int j = 1; j < n; j++)
c4b9                if (!added[j] && (last == -1 || wt[j] > wt[last]))
887d                    last = j;
71bc            if (i == ph - 1) {
9cfa                rep (j, n) w[prev][j] += w[last][j];
1f25                rep (j, n) w[j][prev] = w[prev][j];
5613                used[last] = true;
8e11                c.push_back(last);
bb8e                if (bestw == -1 || wt[last] < bestw) {
```

```
                    bestc = c;
                    bestw = wt[last];
                }
            } else {
                rep (j, n) wt[j] += w[last][j];
                added[last] = true;
            }
        }
    }
    return {bestw, bestc};
}
```

6.8 Heavy-light decomposition

```
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);
    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)
#define rson (o<<1|1)

  void pull(int o) {
    val[o] = (val[lson] + val[rson]) % p;
  }

  void push_add(int o, LL x) {
    val[o] = (val[o] + x * (r[o] - l[o])) % p;
    tadd[o] = (tadd[o] + x) % p;
  }
```

```
  void push_mul(int o, LL x) {
    val[o] = val[o] * x % p;
    tadd[o] = tadd[o] * x % p;
    tmul[o] = tmul[o] * x % p;
  }

  void push(int o) {
    if (l[o] == m[o]) return;
    if (tmul[o] != 1) {
      push_mul(lson, tmul[o]);
      push_mul(rson, tmul[o]);
      tmul[o] = 1;
    }
    if (tadd[o]) {
      push_add(lson, tadd[o]);
      push_add(rson, tadd[o]);
      tadd[o] = 0;
    }
  }

  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 {
      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;
```

7.2 Link/cut tree

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

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

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

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

```
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);
cf90      rotate(x);
95cf    }
78a0    pull(x);
95cf  }
427e
6229  void access(int x) {
1548    int z = x;
ba78    for (int y = 0; x; x = fa[y = x]) {
8fec      splay(x);
b05d      ch[x][1] = y;
78a0      pull(x);
95cf    }
7afd    splay(z);
95cf  }
427e
502e  void chroot(int x) {
766a    access(x);
cb0d    reverse(x);
95cf  }
427e
471a  void split(int x, int y) {
3015    chroot(x);
29b5    access(y);
95cf  }
427e
d87a  int Root(int x) {
766a    access(x);
874d    while (ch[x][0]) {
a97b      push(x);
b83a      x = ch[x][0];
95cf    }
8fec    splay(x);
d074    return x;
95cf  }
427e
70d3  void Link(int u, int v) {  // assume unconnected before
b8a5    chroot(u);
2448    fa[u] = v;
95cf  }
427e
c2f4  void Cut(int u, int v) {  // assume connected before
e8ce    split(u, v);
fd95    fa[u] = ch[v][0] = 0;
743b    pull(v);
95cf  }
427e
6ca2  int Query(int u, int v) {
e8ce    split(u, v);
a5ba    return sum[v];
95cf  }
427e
eaba  void Update(int u, int x) {
46ce    splay(u);
1d62    val[u] = x;
95cf  }
```

```
329b };
```

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

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 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

8.1.4 $\sim 1 \times 10^9$

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

8.1.5 $\sim 1 \times 10^{18}$

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.