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 Makefile

```
dab2 .PHONY : run
427e
207e $(t) : $(t).cpp
2d16   g++ --std=c++14 -Wall -D__LOCAL_DEBUG__ -fsanitize=undefined -fsanitize=
       address -ggdb -pipe -o $@ $<
427e
5f25 run : $(t)
bf3e   ./$(t) < $(t).in
```

1.3 .vimrc

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

1.4 Stack

```
const int STK_SZ = 2000000; bebe
char STK[STK_SZ * sizeof(void*)]; effc
void *STK_BAK; 4e99
 427e
#if defined(__i386__) 7bc9
#define SP "%%esp" 0894
#elif defined(__x86_64__) ac7a
#define SP "%%rsp" a9ea
#endif 1937
 427e
int main() { 3117
  asm volatile("mov␣" SP ",%0;␣mov␣%1," SP: "=g"(STK_BAK):"g"(STK+sizeof(STK)):) 3750
    ;
 427e
  // main program 427e
 427e
  asm volatile("mov␣%0," SP::"g"(STK_BAK)); 6856
  return 0; 7021
} 95cf
```

1.5 Template

```
#include <bits/stdc++.h> 302f
using namespace std; 421c
 427e
#ifdef __LOCAL_DEBUG__ 426f
# define _debug(fmt, ...) fprintf(stderr, "[%s]␣" fmt "\n", \ 3341
    __func__, ##__VA_ARGS__) 611f
#else a8cb
# define _debug(...) ((void) 0) e6b5
#endif 1937
#define rep(i, n) for (int i=0; i<(n); i++) 0d6c
#define Rep(i, n) for (int i=1; i<=(n); i++) cfe3
#define range(x) begin(x), end(x) 3505
typedef long long LL; 5cad
typedef unsigned long long ULL; b773
```

2 Miscellaneous Algorithms

2.1 2-SAT

```
0f42 const int MAXN = 100005;
03a9 struct twoSAT{
5c83     int n;
8f72     vector<int> G[MAXN*2];
d060     bool mark[MAXN*2];
b42d     int S[MAXN*2], c;
427e 
d34f     void init(int n){
b985         this->n = n;
f9ec         for (int i=0; i<n*2; i++) G[i].clear();
0609         memset(mark, 0, sizeof(mark));
95cf     }
427e 
3bd5     bool dfs(int x){
bd70         if (mark[x^1]) return false;
c96a         if (mark[x]) return true;
fd23         mark[x] = true;
4bea         S[c++] = x;
1ce6         for (int i=0; i<G[x].size(); i++)
d942             if (!dfs(G[x][i])) return false;
3361         return true;
95cf     }
427e 
5894     void add_clause(int x, bool xval, int y, bool yval){
6afe         x = x * 2 + xval;
e680         y = y * 2 + yval;
81cc         G[x^1].push_back(y);
6835         G[y^1].push_back(x);
95cf     }
427e 
d0cb     bool solve() {
7c39         for (int i=0; i<n*2; i+=2){
e63f             if (!mark[i] && !mark[i+1]){
88fb                 c = 0;
f4b9                 if (!dfs(i)){
3f03                     while (c > 0) mark[S[--c]] = false;
86c5                     if (!dfs(i+1)) return false;
95cf                 }
95cf             }
            }                                                   95cf
            return true;                                        3361
        }                                                       95cf
                                                                427e
        inline bool value(unsigned i){return mark[2*i+1];}      5f0a
};                                                              329b
```

2.2 Knuth's optimization

```
int n;                                                          5c83
int dp[256][256], dc[256][256];                                 d77c
                                                                427e
template <typename T>                                           b7ec
void compute(T cost) {                                          0bc7
  for (int i = 0; i <= n; i++) {                                0423
    dp[i][i] = 0;                                               8f5e
    dc[i][i] = i;                                               9488
  }                                                             95cf
  rep (i, n) {                                                  be8e
    dp[i][i+1] = 0;                                             95b5
    dc[i][i+1] = i;                                             aa0f
  }                                                             95cf
  for (int len = 2; len <= n; len++) {                          ec08
    for (int i = 0; i + len <= n; i++) {                        88b8
      int j = i + len;                                          d3da
      int lbnd = dc[i][j-1], rbnd = dc[i+1][j];                 9824
      dp[i][j] = INT_MAX / 2;                                   a24a
      int c = cost(i, j);                                       f933
      for (int k = lbnd; k <= rbnd; k++) {                      90d2
        int res = dp[i][k] + dp[k][j] + c;                      9bd0
        if (res < dp[i][j]) {                                   26b5
          dp[i][j] = res;                                       e6af
          dc[i][j] = k;                                         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.

```
constexpr int BLOCK_SZ = 300;

struct query { int l, r, id; };
vector<query> queries;

void add_query(int id, int l, int r) {
  queries.push_back(query{l, r, id});
}

int l, r;

// ----- functions to implement -----
inline void init();
inline void yield(int id);
inline void enter(int o);
inline void leave(int o);

void run() {
  if (queries.empty()) return;
  sort(range(queries), [](query lhs, query rhs) {
    int lb = lhs.l / BLOCK_SZ, rb = rhs.l / BLOCK_SZ;
    if (lb != rb) return lb < rb;
    return lhs.r < rhs.r;
  });
  l = queries[0].l;
  r = queries[0].r;
  init();
  for (query q : queries) {
    while (l > q.l) enter(l - 1), l--;
    while (r < q.r) enter(r + 1), r++;
    while (l < q.l) leave(l), l++;
    while (r > q.r) leave(r), r--;
    yield(q.id);
  }
}
```

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);
    }
  }
};
```

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

3.3 Aho-corasick automaton

```
struct AC : Trie {                                          a1ad
  int fail[MAXN];                                           9143
  int last[MAXN];                                           daca
                                                            427e
  void construct() {                                        8690
    queue<int> q;                                           93d2
    fail[0] = 0;                                            a7a6
    rep(c, CHARN) {                                         ce3c
      if (int u = tr[0][c]) {                               b1c6
        fail[u] = 0;                                        a506
        q.push(u);                                          3e14
        last[u] = 0;                                        f689
      }                                                     95cf
    }                                                       95cf
    while (!q.empty()) {                                    cc78
      int r = q.front();                                    31f0
      q.pop();                                              15dd
      rep(c, CHARN) {                                       ce3c
        int u = tr[r][c];                                   ab59
        if (!u) {                                           0ef5
          tr[r][c] = tr[fail[r]][c];                        9d58
          continue;                                         b333
        }                                                   95cf
        q.push(u);                                          3e14
        int v = fail[r];                                    b3ff
        while (v && !tr[v][c]) v = fail[v];                 d2ea
        fail[u] = tr[v][c];                                 c275
        last[u] = tag[fail[u]] ? fail[u] : last[fail[u]];   654c
      }                                                     95cf
    }                                                       95cf
  }                                                         95cf
                                                            427e
  void found(int pos, int j) {                              7752
```

```
043e      if (j) {
427e        // ! add codes for having found word with tag[j]
4a96        found(pos, last[j]);
95cf      }
95cf    }
427e
9785    void find(const char* text) {  // must be called after construct()
80a4      int p = 0, c, len = strlen(text);
9c94      rep(i, len) {
b3db        c = id(text[i]);
f119        p = tr[p][c];
f08e        if (tag[p])
389b          found(i, p);
1e67        else if (last[p])
299e          found(i, last[p]);
95cf      }
95cf    }
329b  };
```

3.4 Suffix array

The character immediately after the end of the string **MUST** be set to the **UNIQUE SMALLEST** element.

Usage:

`s[]`	the source string
`sa[i]`	the index of starting position of i-th suffix
`rk[i]`	the number of suffixes less than the suffix starting from i
`h[i]`	the longest common prefix between the i-th and $(i-1)$-th lexicographically smallest suffixes
`n`	size of source string
`m`	size of character set

```
de09  void radix_sort(int x[], int y[], int sa[], int n, int m) {
ec00      static int cnt[1000005];    // size > max(n, m)
6066      fill(cnt, cnt + m, 0);
93b7      rep (i, n) cnt[x[y[i]]]++;
9154      partial_sum(cnt, cnt + m, cnt);
acac      for (int i = n - 1; i >= 0; i--) sa[--cnt[x[y[i]]]] = y[i];
95cf  }
427e
c939  void suffix_array(int s[], int sa[], int rk[], int n, int m) {
a69a      static int y[1000005];  // size > n
7306      copy(s, s + n, rk);
    iota(y, y + n, 0);                                                      afbb
    radix_sort(rk, y, sa, n, m);                                            7b42
    for (int j = 1, p = 0; j <= n; j <<= 1, m = p, p = 0) {                 c8c2
        for (int i = n - j; i < n; i++) y[p++] = i;                         8c3a
        rep (i, n) if (sa[i] >= j) y[p++] = sa[i] - j;                      9323
        radix_sort(rk, y, sa, n, m + 1);                                    9e9d
        swap_ranges(rk, rk + n, y);                                         ae41
        rk[sa[0]] = p = 1;                                                  ffd2
        for (int i = 1; i < n; i++)                                         445e
            rk[sa[i]] = ((y[sa[i]] == y[sa[i-1]] and y[sa[i]+j] == y[sa[i-1]+j])  f8dc
              ? p : ++p);
        if (p == n) break;                                                  02f0
    }                                                                       95cf
    rep (i, n) rk[sa[i]] = i;                                               97d9
}                                                                           95cf
                                                                            427e
void calc_height(int s[], int sa[], int rk[], int h[], int n) {             1715
    int k = 0;                                                              c41f
    h[0] = 0;                                                               f313
    rep (i, n) {                                                            be8e
        k = max(k - 1, 0);                                                  0883
        if (rk[i]) while (s[i+k] == s[sa[rk[i]-1]+k]) ++k;                  527d
        h[rk[i]] = k;                                                       56b7
    }                                                                       95cf
}                                                                           95cf
```

3.5 Trie

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

```
  }

  // 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];
  }
};
```

3.6 Rolling hash

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

Usage:

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

```
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;
}

struct hasher {
    LL val[MAXN];

    void build(const char *str) {   // assume lower-case letter only
        for (int i = 0; str[i]; i++)
            val[i+1] = (mul(val[i], g) + str[i]) % mod;
    }

    LL operator() (int l, int r) { // [l, r)
        return (val[r] - mul(val[l], pg[r - l]) + mod) % mod;
    }
} ha;
```

4 Math

4.1 Extended Euclidean algorithm and Chinese remainder theorem

```
void exgcd(LL a, LL b, LL &g, LL &x, LL &y) {
    if (!b) g = a, x = 1, y = 0;
    else {
        exgcd(b, a % b, g, y, x);
        y -= x * (a / b);
    }
}

LL crt(LL r[], LL p[], int n) {
  LL q = 1, ret = 0;
  rep (i, n) q *= p[i];
  rep (i, n) {
        LL m = q / p[i];
        LL d, x, y;
    exgcd(p[i], m, d, x, y);
    ret = (ret + y * m * r[i]) % q;
```

```
95cf   }
2e47   return (q + ret) % q;
95cf }
```

4.2 Matrix powermod

```
44b4 const int MAXN = 105;
92df const LL modular = 1000000007;
5c83 int n; // order of matrices
427e
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 }
```

4.3 Linear basis

```
const int MAXD = 30;                                          8b44
struct linearbasis {                                          03a6
    ULL b[MAXD] = {};                                         3558
                                                              427e
    bool insert(LL v) {                                       1566
        for (int j = MAXD - 1; j >= 0; j--) {                 9b2b
            if (!(v & (1ll << j))) continue;                  de36
            if (b[j]) v ^= b[j]                               ee78
            else {                                            037f
                for (int k = 0; k < j; k++)                   7836
                    if (v & (1ll << k)) v ^= b[k];            f0b4
                for (int k = j + 1; k < MAXD; k++)            b0aa
                    if (b[k] & (1ll << j)) b[k] ^= v;         46c9
                b[j] = v;                                     8295
                return true;                                  3361
            }                                                 95cf
        }                                                     95cf
        return false;                                         438e
    }                                                         95cf
};                                                            329b
```

4.4 Gauss elimination over finite field

```
const LL p = 1000000007;                                      b784
                                                              427e
LL powmod(LL b, LL e) {                                       2a2c
  LL r = 1;                                                   95a2
  while (e) {                                                 3e90
    if (e & 1) r = r * b % p;                                 1783
    b = b * b % p;                                            5549
    e >>= 1;                                                  16fc
  }                                                           95cf
  return r;                                                   547e
}                                                             95cf
                                                              427e
typedef vector<LL> VLL;                                       c130
typedef vector<VLL> VVLL;                                     42ac
                                                              427e
LL gauss(VVLL &a, VVLL &b) {                                  2c62
  const int n = a.size(), m = b[0].size();                    561b
  vector<int> irow(n), icol(n), ipiv(n);                      a25e
```

```
  LL det = 1;

  rep (i, n) {
    int pj = -1, pk = -1;
    rep (j, n) if (!ipiv[j])
      rep (k, n) if (!ipiv[k])
        if (pj == -1 || a[j][k] > a[pj][pk]) {
          pj = j;
          pk = k;
        }
    if (a[pj][pk] == 0) return 0;
    ipiv[pk]++;
    swap(a[pj], a[pk]);
    swap(b[pj], b[pk]);
    if (pj != pk) det = (p - det) % p;
    irow[i] = pj;
    icol[i] = pk;

    LL c = powmod(a[pk][pk], p - 2);
    det = det * a[pk][pk] % p;
    a[pk][pk] = 1;
    rep (j, n) a[pk][j] = a[pk][j] * c % p;
    rep (j, m) b[pk][j] = b[pk][j] * c % p;
    rep (j, n) if (j != pk) {
      c = a[j][pk];
      a[j][pk] = 0;
      rep (k, n) a[j][k] = (a[j][k] + p - a[pk][k] * c % p) % p;
      rep (k, m) b[j][k] = (b[j][k] + p - b[pk][k] * c % p) % p;
    }
  }

  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;
}
```

4.5 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;
      }
    }
    return ret;
  }

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

```

Poly operator * (const int &c, const Poly &p) {
  vector<int> na(p.length());
  for (int i = 0; i < (int)na.size(); ++i) {
    na[i] = (long long)c * p.a[i] % MOD;
  }
  return na;
}

vector<int> solve(vector<int> a) {
  int n = a.size();
  Poly s, b;
  s.a.push_back(1), b.a.push_back(1);
  for (int i = 1, j = 0, ld = a[0]; i < n; ++i) {
    int d = s.calc(a, i);
    if (d) {
      if ((s.length() - 1) * 2 <= i) {
        Poly ob = b;
        b = s;
        s = s - (long long)d * inverse(ld) % MOD * ob.move(i - j);
        j = i;
        ld = d;
      } else {
        s = s - (long long)d * inverse(ld) % MOD * b.move(i - j);
      }
    }
  }
  // Caution: s.a might be shorter than expected
  return s.a;
}
```

4.6 Fast Walsh-Hadamard transform

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

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);
}
```

4.7 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;
            for (cplx* p = a; p != a + N; p += l)
                rep (k, m){
                    cplx t = w[N/l*k] * p[k+m];
                    p[k+m] = p[k] - t; p[k] += t;
                }
        }
    }

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

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

4.8 Number theoretic transform

```
const int NMAX = 1<<21;

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

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

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

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

```
};
```

4.9 Sieve of Euler

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

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

4.10 Sieve of Euler (General)

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

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

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

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

4.11 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 $n < 2^{64}$.

```
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;
}
```

4.12 Pollard's rho algorithm

```
ULL gcd(ULL a, ULL b) {return b ? gcd(b, a % b) : a;}

ULL PollardRho(ULL n){
    ULL c, x, y, d = n;
    if (~n&1) return 2;
    while (d == n){
        x = y = 2;
        d = 1;
        c = rand() % (n - 1) + 1;
        while (d == 1){
            x = (mulmod(x, x, n) + c) % n;
            y = (mulmod(y, y, n) + c) % n;
            y = (mulmod(y, y, n) + c) % n;
            d = gcd(x>y ? x-y : y-x, n);
        }
    }
    return d;
}
```

5 Graph Theory

5.1 Strongly connected component

```
const int MAXV = 100005;

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

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

    void dfs(int u){
        pre[u] = lnk[u] = ++time;
        s.push(u);
        for (int v : adj[u]){
            if (!pre[v]){
                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(){
        Rep (i, V)
            rep (j, adj[i].size()){
                if (scc[i] != scc[adj[i][j]])
                    adjc[scc[i]].push_back(scc[adj[i][j]]);
            }
    }
};
```

5.2 Vertex biconnected component

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

    stack<pair<int, int>> s;

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

    int dfs(int u, int fa) {
        int lowu = pre[u] = ++dfs_clock;
        int child = 0;
        for (int v : adj[u]) {
            if (!pre[v]) {
                s.push({u, v});
                child++;
                int lowv = dfs(v, u);
                lowu = min(lowu, lowv);
                if (lowv >= pre[u]) {
                    iscut[u] = 1;
                    bcc[bcc_cnt].clear();
                    bcce[bcc_cnt].clear();
                    while (1) {
                        int xu, xv;
                        tie(xu, xv) = s.top(); s.pop();
                        bcce[bcc_cnt].insert({min(xu, xv), max(xu, xv)});
                        if (bccno[xu] != bcc_cnt) {
                            bcc[bcc_cnt].push_back(xu);
                            bccno[xu] = bcc_cnt;
                        }
                        if (bccno[xv] != bcc_cnt) {
                            bcc[bcc_cnt].push_back(xv);
                            bccno[xv] = bcc_cnt;
                        }
                        if (xu == u && xv == v) break;
                    }
                    bcc_cnt++;
                }
            } else if (pre[v] < pre[u] && v != fa) {
                s.push({u, v});
                lowu = min(lowu, pre[v]);
            }
        }
        if (fa < 0 && child == 1) iscut[u] = 0;
        return lowu;
    }

    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];
int pre[MAXN], vis[MAXN], id[MAXN];

LL getans(int n, int rt, vector<edge>& edges) {
    LL ans = 0;
    int cnt = 0;
    while (1) {
```

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

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, $O(\sqrt{V}E)$.

```
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);
                }
            }
        }
        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]];
```

```
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)

```
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 
```

```
struct Hungarian{                                84ee
    int nx, ny;                                  fbf6
    vector<int> mx, my;                          9ec6
    vector<vector<int> > e;                      9d4c
    vector<bool> mark;                           edec
                                                 427e
    void init(int nx, int ny){                   8324
        this->nx = nx;                           c1d1
        this->ny = ny;                           f9c1
        mx.resize(nx); my.resize(ny);            ac92
        e.clear(); e.resize(nx);                 3f11
        mark.resize(nx);                         1023
    }                                            95cf
                                                 427e
    inline void add(int a, int b){               4589
        e[a].push_back(b);                       486c
    }                                            95cf
                                                 427e
    bool augment(int i){                         0c2b
        if (!mark[i]) {                          207c
            mark[i] = true;                      dae4
            for (int j : e[i]){                  6a1e
                if (my[j] == -1 || augment(my[j])){   0892
                    mx[i] = j; my[j] = i;        9ca3
                    return true;                 3361
                }                                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
```

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

Usage:

`init(n)`	Initialize the template with n vertices, numbered from 1.
`add_edge(u, v)`	Add an undirected edge uv.
`solve()`	Find the maximum matching. Return the number of matched edges.
`mate[]`	The mate of a matched vertex. If it is not matched, then the value is 0.

Time Complexity: $O(|V|^3)$, but extremely fast in practice.

```
c041 const int MAXN = 1024;
6ab1 struct Blossom {
0b32     vector<int> adj[MAXN];
93d2     queue<int> q;
5c83     int n;  // set n to number of vertices before use
0de2     int label[MAXN], mate[MAXN], save[MAXN], used[MAXN];
427e
427e
2186     void init(int nv) {
6646         n = nv;
e962         Rep (i, n) adj[i].clear();
f7e2         memset(label, 0, sizeof label);
5f6a         memset(mate, 0, sizeof mate);
c4b9         memset(save, 0, sizeof save);
ee13         memset(used, 0, sizeof used);
95cf     }
427e
bfab     void add_edge(int u, int v) {
c71a         adj[u].push_back(v);
a717         adj[v].push_back(u);
95cf     }
427e
2a48     void rematch(int x, int y){
8af8         int m = mate[x]; mate[x] = y;
1aa4         if (mate[m] == x) {
f4ba             if (label[x] <= n) {
e7ce                 mate[m] = label[x];
bec9                 rematch(label[x], m);
8e2e             } else {
3341                 int a = 1 + (label[x] - n - 1) / n;
2885                 int b = 1 + (label[x] - n - 1) % n;
ef33                 rematch(a, b); rematch(b, a);
95cf             }
```

```
        }                                                    95cf
    }                                                        95cf
                                                             427e
    void traverse(int x) {                                   8a50
        Rep (i, n) save[i] = mate[i];                        43c0
        rematch(x, x);                                       2ef7
        Rep (i, n) {                                         34d7
            if (mate[i] != save[i]) used[i]++;               62c5
            mate[i] = save[i];                               97ef
        }                                                    95cf
    }                                                        95cf
                                                             427e
    void relabel(int x, int y) {                             8bf8
        Rep (i, n) used[i] = 0;                              d101
        traverse(x); traverse(y);                            c4ea
        Rep (i, n) {                                         34d7
            if (used[i] == 1 and label[i] < 0) {             dee9
                label[i] = n + x + (y - 1) * n;              1c22
                q.push(i);                                   eb31
            }                                                95cf
        }                                                    95cf
    }                                                        95cf
                                                             427e
    int solve() {                                            a0ce
        Rep (i, n) {                                         34d7
            if (mate[i]) continue;                           a073
            Rep (j, n) label[j] = -1;                        1fc0
            label[i] = 0; q = queue<int>(); q.push(i);       7676
            while (q.size()) {                               1c7d
                int x = q.front(); q.pop();                  66ba
                for (int y : adj[x]) {                       b98c
                    if (mate[y] == 0 and i != y) {           c07f
                        mate[y] = x;                         0593
                        rematch(x, y);                       2b14
                        q = queue<int>();                    8ea8
                        break;                               6173
                    }                                        95cf
                    if (label[y] >= 0) {                     9079
                        relabel(x, y);                       a72e
                        continue;                            b333
                    }                                        95cf
                    if (label[mate[y]] < 0) {                58ec
                        label[mate[y]] = x;                  9773
                        q.push(mate[y]);                     086d
```

```
95cf                     }
95cf                 }
95cf             }
95cf         }
8abb         int cnt = 0;
c816         Rep (i, n) if (mate[i] > i) cnt++;
6808         return cnt;
95cf     }
329b };
```

5.7 Minimum cost maximum flow

```
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;
```

```
        p[s] = 0; a[s] = INT_MAX; 2dae
        while (!q.empty()){ cc78
            int u = q.front(); q.pop(); inq[u] = false; b0aa
            for (int i : G[u]) { 3bba
                edge& e = edges[i]; 56d8
                if (e.cap > e.flow && d[e.to] > d[u] + e.cost){ 3601
                    d[e.to] = d[u] + e.cost; 55bc
                    p[e.to] = G[u][i]; 0bea
                    a[e.to] = min(a[u], e.cap - e.flow); 8249
                    if (!inq[e.to]) q.push(e.to), inq[e.to] = true; e5d3
                } 95cf
            } 95cf
        } 95cf
        return d[t] != INF; 6d7c
    } 95cf
 427e
    void augment(){ 71a4
        int u = t; 06f1
        while (u != s){ b19d
            edges[p[u]].flow += a[t]; db09
            edges[p[u]^1].flow -= a[t]; 25a9
            u = edges[p[u]].from; e6c9
        } 95cf
    } 95cf
 427e
#ifdef GIVEN_FLOW 6e20
    bool min_cost(int s, int t, int f, LL& cost) { 5972
        this->s = s; this->t = t; 590d
        int flow = 0; 21d4
        cost = 0; 23cb
        while (spfa()) { 22dc
            augment(); bcdb
            if (flow + a[t] >= f){ a671
                cost += (f - flow) * d[t]; flow = f; b14d
                return true; 3361
            } else { 8e2e
                flow += a[t]; cost += a[t] * d[t]; 2a83
            } 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
```

```
21d4         int flow = 0;
23cb         cost = 0;
22dc         while (spfa()) {
bcdb             augment();
2a83             flow += a[t]; cost += a[t] * d[t];
95cf         }
84fb         return flow;
95cf     }
1937 #endif
329b };
```

5.8 Global minimum cut (Stoer-Wagner)

Usage:

`stoer(w)`	Compute the global minimum cut of the graph specified by the **symmetric** adjacenct matrix `w` (0-based). Return the capaity of the cut and the indices of one part of the cut.

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

```
f9d7 typedef vector<LL> VI;
045e typedef vector<VI> VVI;
427e
f012 pair<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;                                  bab6
                    bestw = wt[last];                           372e
                }                                               95cf
            } else {                                            8e2e
                rep (j, n) wt[j] += w[last][j];                 caeb
                added[last] = true;                             8b92
            }                                                   95cf
        }                                                       95cf
    }                                                           95cf
    return {bestw, bestc};                                      038c
}                                                               95cf
```

5.9 Fast LCA

All indices of the tree are 1-based.

Usage:

`preprocess(root)`	Initialize with tree rooted at `root`.
`lca(u, v)`	Query the lowest common ancestor of u and v.

```
const int MAXN = 500005;                                        0e34
vector<int> adj[MAXN];                                          0b32
int id[MAXN], nid;                                              fccb
pair<int, int> st[MAXN << 1][33 - __builtin_clz(MAXN)];         1356
                                                                427e
void dfs(int u, int p, int d) {                                 e16d
    st[id[u] = nid++][0] = {d, u};                              0df2
    for (int v : adj[u]) {                                      18f6
        if (v == p) continue;                                   bd87
        dfs(v, u, d + 1);                                       f58c
        st[nid++][0] = {d, u};                                  08ad
    }                                                           95cf
}                                                               95cf
                                                                427e
void preprocess(int root) {                                     3d1b
    nid = 0;                                                    3269
    dfs(root, 0, 1);                                            91e1
    int l = 31 - __builtin_clz(nid);                            5e98
    rep (j, l) rep (i, 1+nid-(1<<j))                            213b
        st[i][j+1] = min(st[i][j], st[i+(1<<j)][j]);            1131
}                                                               95cf
                                                                427e
int lca(int u, int v) {                                         0f0b
    tie(u, v) = minmax(id[u], id[v]);                           cfc4
```

```
    int k = 31 - __builtin_clz(v-u+1);
    return min(st[u][k], st[v-(1<<k)+1][k]).second;
}
```

5.10 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);
    dfs2(root, root);
}

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

5.11 Centroid decomposition

Note that the centroid here is not the exact centroid of the graph. It only guarantees that the size of each subtree does not exceed half of that of the original tree. This is enough to guarantee the correct time complexity. All vertices are numbered from 1. Call `decomp(root)` to use.

Usage:

`decomp(u, p)`	Decompose the tree rooted at u with parent p.

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

```
vector<int> adj[100005];
int sz[100005], sum;

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

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

```
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
  }
}
```

5.12 DSU on tree

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

Usage:

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

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

```
vector<int> adj[100005];
int sz[100005], son[100005];

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

template <typename T>
void trav(T fn, int u, int p) {
    fn(u);
    for (int v : adj[u]) if (v != p) trav(fn, v, u);
}

#define for_light(v) for (int v : adj[u]) if (v != p and v != son[u])
void work(int u, int p, bool keep) {
    for_light(v) work(v, u, 0); // process light children

    // process heavy child
    // current data structure contains info of heavy child
    if (son[u]) work(son[u], u, 1);

    auto merge = [u] (int c) { /* count contribution of c */ };
    auto enter = [] (int c) { /* add vertex c */ };
    auto leave = [] (int c) { /* remove vertex c*/ };

    for_light(v) {
        trav(merge, v, u);
        trav(enter, v, u);
    }

    // count answer for root and add it
    // Warning: special check may apply to root!
    merge(u);
    enter(u);

    // leave current tree
    if (!keep) trav(leave, u, p);
}
```

6 Data Structures

6.1 Fenwick tree (point update range query)

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

    void init(int n) { // fill the array with 0
```

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

6.2 Fenwick tree (range update point query)

```
3d03 struct bit_rupq{ // range update, point query
d7af     int N;
99ff     vector<LL> tr;
427e
d34f     void init(int n) { // fill the array with 0
1010         tr.resize(N = n + 5);
95cf     }
427e
38d4     LL query(int n) {
f7ff         LL ans = 0;
ad20         while (n < N) {
0715             ans += tr[n];
0af5             n += n & -n;
95cf         }
4206         return ans;
95cf     }
427e
f4bd     void add(int n, LL x) {
e290         while (n){
6c81             tr[n] += x;
```

```
            n &= n - 1; c0d4
        } 95cf
    } 95cf
}; 329b
```

6.3 Segment tree

```
LL p; 3942
const int MAXN = 4 * 100006; 1ebb
struct segtree { 451a
  int l[MAXN], m[MAXN], r[MAXN]; 27be
  LL val[MAXN], tadd[MAXN], tmul[MAXN]; 4510
 427e
#define lson (o<<1) ac35
#define rson (o<<1|1) 1294
 427e
  void pull(int o) { 1344
    val[o] = (val[lson] + val[rson]) % p; bbe9
  } 95cf
 427e
  void push_add(int o, LL x) { e4bc
    val[o] = (val[o] + x * (r[o] - l[o])) % p; 5dd6
    tadd[o] = (tadd[o] + x) % p; 6eff
  } 95cf
 427e
  void push_mul(int o, LL x) { d658
    val[o] = val[o] * x % p; b82c
    tadd[o] = tadd[o] * x % p; aa86
    tmul[o] = tmul[o] * x % p; 649f
  } 95cf
 427e
  void push(int o) { b149
    if (l[o] == m[o]) return; 3159
    if (tmul[o] != 1) { 0a90
      push_mul(lson, tmul[o]); 0f4a
      push_mul(rson, tmul[o]); 045e
      tmul[o] = 1; ac0a
    } 95cf
    if (tadd[o]) { 1b82
      push_add(lson, tadd[o]); 9547
      push_add(rson, tadd[o]); 0e73
      tadd[o] = 0; 6234
```

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

```
      if (rr <= m[o]) return query(lson, ll, rr);          462a
      if (ll >= m[o]) return query(rson, ll, rr);          5cca
      return query(lson, ll, rr) + query(rson, ll, rr);    bbf9
    }                                                      95cf
  }                                                        95cf
} seg;                                                     4d99
```

6.4 Link/cut tree

Usage:

`pull(x)`	Collect information of subtrees.
`Link(u, v)`	Link two unconnected trees.
`Cut(u, v)`	Cut an existent edge.
`Query(u, v)`	Path aggregation.
`Update(u, x)`	Single point modification.

```
// about 0.13s per 100k ops @Luogu.org                      427e
                                                            427e
namespace LCT {                                             ed4d
  const int MAXN = 300005;                                  5ece
  int fa[MAXN], ch[MAXN][2], val[MAXN], sum[MAXN];          6a6d
  bool rev[MAXN];                                           c6e1
                                                            427e
  bool isroot(int x) {                                      7839
    return ch[fa[x]][0] == x || ch[fa[x]][1] == x;          45a9
  }                                                         95cf
                                                            427e
  void pull(int x) {                                        3bf9
    sum[x] = val[x] ^ sum[ch[x][0]] ^ sum[ch[x][1]];        6664
  }                                                         95cf
                                                            427e
  void reverse(int x) {                                     3698
    swap(ch[x][0], ch[x][1]);                               7850
    rev[x] ^= 1;                                            52c6
  }                                                         95cf
                                                            427e
  void push(int x) {                                        1a53
    if (rev[x]) {                                           8f1f
      if (ch[x][0]) reverse(ch[x][0]);                      ebf3
      if (ch[x][1]) reverse(ch[x][1]);                      6eb0
      rev[x] = 0;                                           8fc1
    }                                                       95cf
  }                                                         95cf
```

```
427e
425f   void rotate(int x) {
51af     int y = fa[x], z = fa[y], k = ch[y][1] == x, w = ch[x][!k];
e1fe     if (isroot(y)) ch[z][ch[z][1] == y] = x;
af46     ch[x][!k] = y; ch[y][k] = w;
fa6f     if (w) fa[w] = y;
3540     fa[y] = x; fa[x] = z;
72ef     pull(y);
95cf   }
427e
bc1b   void pushall(int x) {
a316     if (isroot(x)) pushall(fa[x]);
a97b     push(x);
95cf   }
427e
f69c   void splay(int x) {
d095     int y = x, z = 0;
8ab3     pushall(y);
f244     while (isroot(x)) {
ceef       y = fa[x]; z = fa[y];
4449       if (isroot(y)) rotate((ch[y][0] == x) ^ (ch[z][0] == y) ? x : y);
cf90       rotate(x);
95cf     }
78a0     pull(x);
95cf   }
427e
6229   void access(int x) {
1548     int z = x;
ba78     for (int y = 0; x; x = fa[y = x]) {
8fec       splay(x);
b05d       ch[x][1] = y;
78a0       pull(x);
95cf     }
7afd     splay(z);
95cf   }
427e
502e   void chroot(int x) {
766a     access(x);
cb0d     reverse(x);
95cf   }
427e
471a   void split(int x, int y) {
3015     chroot(x);
29b5     access(y);
```

```
  }                                                          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`

```
#include <ext/pb_ds/assoc_container.hpp>                     0475
using namespace __gnu_pbds;                                  332d
                                                             427e
tree<int, null_type, less<int>, rb_tree_tag, tree_order_statistics_node_update>   43a7
  rkt;
// 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) {
      k -= rt->left->value - lt->left->value;
      return Query(lt->right, rt->right, mid, r, k);
    } else {
      return Query(lt->left, rt->left, l, mid, k);
    }
  }

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

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

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

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

6.7 Persistent block list

Block list that supports persistence. All indices are 0-based. `std::shared_ptr` is used to ease memory management. One should modify the constructor of `block` to maintain extra information. Here we use this policy that the size of each block does not exceed `BLOCK`, while the sum of sizes of two adjacent blocks does not less than `BLOCK`.
When some operation that breaks block list property, please call `maintain` in time to restore

the property.

Usage:

`maintain()`	Maintain the block list property.
`split(pos)`	Split the block list at position `pos`. Returns an iterator to a block starting at `pos`.
`sum(l, r)`	An example function of list traversal between $[l, r)$.

Time Complexity: When `BLOCK` is property selected, the time complexity is $O(\sqrt{n})$ per operation.

```
a19e constexpr int BLOCK = 800;
76b3 typedef vector<int> vi;
0563 typedef shared_ptr<vi> pvi;
013b typedef shared_ptr<const vi> pcvi;
427e
a771 struct block {
2989     pcvi data;
8fd0     LL sum;
427e
427e     // add information to maintain
a613     block(pcvi ptr) :
24b5         data(ptr),
0cf0         sum(accumulate(ptr->begin(), ptr->end(), 0ll))
e93b     { }
427e
5c0f     void merge(const block& another) {
0b18         pvi temp = make_shared<vi>(data->begin(), data->end());
ac21         temp->insert(temp->end(), another.data->begin(), another.data->end());
6467         *this = block(temp);
95cf     }
427e
42e8     block split(int pos) {
dac1         block result(make_shared<vi>(data->begin() + pos, data->end()));
01db         *this = block(make_shared<vi>(data->begin(), data->begin() + pos));
56b0         return result;
95cf     }
329b };
427e
2a18 typedef list<block>::iterator lit;
427e
ce14 struct blocklist {
5540     list<block> blk;
427e
7b8e     void maintain() {
        lit it = blk.begin();                                                    3131
        while (it != blk.end() and next(it) != blk.end()) {                      5e44
            lit it2 = it;                                                        852d
            while (next(it2) != blk.end() and                                    0b03
                    it2->data->size() + next(it2)->data->size() <= BLOCK) {      029f
                it2->merge(*next(it2));                                          93e1
                blk.erase(next(it2));                                            e1fa
            }                                                                    95cf
            ++it;                                                                5771
        }                                                                        95cf
    }                                                                            95cf
                                                                                 427e
    lit split(int pos) {                                                         b7b3
        for (lit it = blk.begin(); ; it++) {                                     2273
            if (pos == 0) return it;                                             5502
            while (it->data->size() > pos) {                                     d480
                blk.insert(next(it), it->split(pos));                            2099
            }                                                                    95cf
            pos -= it->data->size();                                             a1c8
        }                                                                        95cf
    }                                                                            95cf
                                                                                 427e
    LL sum(int l, int r) { // traverse                                           fd38
        lit it1 = split(l), it2 = split(r);                                      48b4
        LL res = 0;                                                              ac09
        while (it1 != it2) {                                                     9f1d
            res += it1->sum;                                                     8284
            it1++;                                                               61fd
        }                                                                        95cf
        maintain();                                                              b204
        return res;                                                              244d
    }                                                                            95cf
};                                                                               329b
```

6.8 Sparse table, range extremum query

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

```
const int MAXN = 100007;                                                         db63
int a[MAXN];                                                                     b330
int st[MAXN][32 - __builtin_clz(MAXN)];                                          69ae
                                                                                 427e
inline int ext(int x, int y){return x>y?x:y;} // ! max                           8041
```

```
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 }
```

7 Geometrics

7.1 2D geometric template

```
302f #include <bits/stdc++.h>
421c using namespace std;
427e
4553 typedef int T;
c0ae typedef struct pt {
7a9d     T x, y;
ffaa     T operator , (pt a) { return x*a.x + y*a.y; } // inner product
3ec7     T operator * (pt a) { return x*a.y - y*a.x; } // outer product
221a     pt operator + (pt a) { return {x+a.x, y+a.y}; }
8b34     pt operator - (pt a) { return {x-a.x, y-a.y}; }
427e
368b     pt operator * (T k) { return {x*k, y*k}; }
90f4     pt operator - () { return {-x, -y};}
ba8c } vec;
427e
0ea6 typedef pair<pt, pt> seg;
427e
8d6e bool ptOnSeg(pt& p, seg& s){
ce77     vec v1 = s.first - p, v2 = s.second - p;
de97     return (v1, v2) <= 0 && v1 * v2 == 0;
95cf }
427e
427e // 0 not on segment
```

```
// 1 on segment except vertices 427e
// 2 on vertices 427e
int ptOnSeg2(pt& p, seg& s){ 8421
    vec v1 = s.first - p, v2 = s.second - p; ce77
    T ip = (v1, v2); 70ca
    if (v1 * v2 != 0 || ip > 0) return 0; 8b14
    return (v1, v2) ? 1 : 2; 0847
} 95cf
 427e
// if two orthogonal rectangles do not touch, return true 427e
inline bool nIntRectRect(seg a, seg b){ 72bb
    return min(a.first.x, a.second.x) > max(b.first.x, b.second.x) || f9ac
           min(a.first.y, a.second.y) > max(b.first.y, b.second.y) || f486
           min(b.first.x, b.second.x) > max(a.first.x, a.second.x) || 39ce
           min(b.first.y, b.second.y) > max(a.first.y, a.second.y); 80c7
} 95cf
 427e
// >0 in order 427e
// <0 out of order 427e
// =0 not standard 427e
inline double rotOrder(vec a, vec b, vec c){return double(a*b)*(b*c);} 7538
 427e
inline bool intersect(seg a, seg b){ 31ed
    // ! if (nIntRectRect(a, b)) return false; // if commented, assume that a 427e
      and b are non-collinear
    return rotOrder(b.first-a.first, a.second-a.first, b.second-a.first) >= 0 && cb52
           rotOrder(a.first-b.first, b.second-b.first, a.second-b.first) >= 0; 059e
} 95cf
 427e
// 0 not insersect 427e
// 1 standard intersection 427e
// 2 vertex-line intersection 427e
// 3 vertex-vertex intersection 427e
// 4 collinear and have common point(s) 427e
int intersect2(seg& a, seg& b){ 4d19
    if (nIntRectRect(a, b)) return 0; 5dc4
    vec va = a.second - a.first, vb = b.second - b.first; 42c0
    double j1 = rotOrder(b.first-a.first, va, b.second-a.first), 2096
           j2 = rotOrder(a.first-b.first, vb, a.second-b.first); 72fe
    if (j1 < 0 || j2 < 0) return 0; 5ac6
    if (j1 != 0 && j2 != 0) return 1; 9400
    if (j1 == 0 && j2 == 0){ 83db
        if (va * vb == 0) return 4; else return 3; 6b0c
    } else return 2; fb17
```

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

8 Appendices

8.1 Primes

8.1.1 First primes

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

8.1.2 Arbitrary length primes

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

8.1.3 $\sim 1 \times 10^9$

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

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

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

8.2 Pell's equation

$x^2 - ny^2 = 1$, where n is a positive nonsquare integer.
Let (x_0, y_0) be the smallest positive solution of the equation, then the k-th solution is:

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

Some smallest solutions to Pell's equation:

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

8.3 Burnside's lemma and Polya's enumeration theorem

The Burnside's lemma says that

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

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

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

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

8.4 Lagrange's interpolation

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

$$l_j(x) = \prod_{0 \le m \le k, m \ne j} fracx - x_m x_j - x_m$$

then the Lagrange polynomial is

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