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.

```
5194 constexpr int BLOCK_SZ = 300;
427e
3ec4 struct query { int l, r, id; };
d26a vector<query> queries;
427e
1e30 void add_query(int id, int l, int r) {
54c9   queries.push_back(query{l, r, id});
95cf }
427e
9f6b int l, r;
427e
427e // ----- functions to implement -----
62b4 inline void init();
50e1 inline void yield(int id);
b20d inline void enter(int o);
13af inline void leave(int o);
427e
37f0 void run() {
ab0b   if (queries.empty()) return;
8508   sort(range(queries), [](query lhs, query rhs) {
c7f8     int lb = lhs.l / BLOCK_SZ, rb = rhs.l / BLOCK_SZ;
03e7     if (lb != rb) return lb < rb;
0780     return lhs.r < rhs.r;
b251   });
6196   l = queries[0].l;
9644   r = queries[0].r;
07e2   init();
5bc9   for (query q : queries) {
7bc7     while (l > q.l) enter(l - 1), l--;
d646     while (r < q.r) enter(r + 1), r++;
13f0     while (l < q.l) leave(l), l++;
e1c6     while (r > q.r) leave(r), r--;
```

```
    yield(q.id);                                        82f5
  }                                                     95cf
}                                                       95cf
```

3 String

3.1 Knuth-Morris-Pratt algorithm

```
const int SIZE = 10005;                                 2836
                                                        427e
struct kmp_matcher {                                    d02b
  char p[SIZE];                                         2d81
  int fail[SIZE];                                       9847
  int len;                                              57b7
                                                        427e
  void construct(const char* needle) {                  60cf
    len = strlen(p);                                    aaa1
    strcpy(p, needle);                                  3a87
    fail[0] = fail[1] = 0;                              3dd4
    for (int i = 1; i < len; i++) {                     d8a8
      int j = fail[i];                                  147f
      while (j && p[i] != p[j]) j = fail[j];            3c79
      fail[i + 1] = p[i] == p[j] ? j + 1 : 0;           4643
    }                                                   95cf
  }                                                     95cf
                                                        427e
  inline void found(int pos) {                          c464
    // ! add codes for having found at pos              427e
  }                                                     95cf
                                                        427e
  void match(const char* haystack) {  // must be called after construct   2daf
    const char* t = haystack;                           700f
    int n = strlen(t);                                  8482
    int j = 0;                                          8fd0
    rep(i, n) {                                         be8e
      while (j && p[j] != t[i]) j = fail[j];            4e19
      if (p[j] == t[i]) j++;                            b5d5
      if (j == len) found(i - len + 1);                 f024
    }                                                   95cf
  }                                                     95cf
};                                                      329b
```

3.2 Manacher algorithm

```
81d4 struct Manacher {
cd09   int Len;
9255   vector<int> lc;
b301   string s;
427e
ec07   void work() {
c033     lc[1] = 1;
6bef     int k = 1;
427e
491f     for (int i = 2; i <= Len; i++) {
7957       int p = k + lc[k] - 1;
5e04       if (i <= p) {
24a1         lc[i] = min(lc[2 * k - i], p - i + 1);
8e2e       } else {
e0e5         lc[i] = 1;
95cf       }
74ff       while (s[i + lc[i]] == s[i - lc[i]]) lc[i]++;
2b9a       if (i + lc[i] > k + lc[k]) k = i;
95cf     }
95cf   }
427e
bfd5   void init(const char *tt) {
aaaf     int len = strlen(tt);
f701     s.resize(len * 2 + 10);
7045     lc.resize(len * 2 + 10);
8e13     s[0] = '*';
ae54     s[1] = '#';
1321     for (int i = 0; i < len; i++) {
e995       s[i * 2 + 2] = tt[i];
69fd       s[i * 2 + 1] = '#';
95cf     }
43fd     s[len * 2 + 1] = '#';
75d1     s[len * 2 + 2] = '\0';
61f7     Len = len * 2 + 2;
3e7a     work();
95cf   }
427e
b194   pair<int, int> maxpal(int l, int r) {
901a     int center = l + r + 1;
ffb2     int rad = lc[center] / 2;
ab54     int rmid = (l + r + 1) / 2;
         int rl = rmid - rad, rr = rmid + rad - 1;     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];
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];
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
```

```
95cf   }
427e
427e   // tag should not be 0
30b0   void add(const char* s, int t) {
d50a     int p = 0, c, len = strlen(s);
9c94     rep(i, len) {
3140       c = id(s[i]);
d6c8       if (!tr[p][c]) {
26dd         memset(tr[n], 0, sizeof(tr[n]));
2e5c         tag[n] = 0;
73bb         tr[p][c] = n++;
95cf       }
f119       p = tr[p][c];
95cf     }
35ef     tag[p] = t;
95cf   }
427e
427e   // returns 0 if not found
427e   // AC automaton does not need this function
216c   int search(const char* s) {
d50a     int p = 0, c, len = strlen(s);
9c94     rep(i, len) {
3140       c = id(s[i]);
f339       if (!tr[p][c]) return 0;
f119       p = tr[p][c];
95cf     }
840e     return tag[p];
95cf   }
329b };
```

3.6 Rolling hash

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

Usage:

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

```
1e42 const LL mod = 1006658951440146419, g = 967;
9f60 const int MAXN = 200005;
0291 LL pg[MAXN];
427e
6832 inline LL mul(LL x, LL y) {
c919     return __int128_t(x) * y % mod;
```

```
}                                                                  95cf
                                                                   427e
void init_hash() {   // must be called in `int main()`             599a
    pg[0] = 1;                                                     286f
    for (int i = 1; i < MAXN; i++)                                 d00f
        pg[i] = pg[i - 1] * g % mod;                               4aa9
}                                                                  95cf
                                                                   427e
struct hasher {                                                    7e62
    LL val[MAXN];                                                  534a
                                                                   427e
    void build(const char *str) {   // assume lower-case letter only   4554
        for (int i = 0; str[i]; i++)                               f937
            val[i+1] = (mul(val[i], g) + str[i]) % mod;            9645
    }                                                              95cf
                                                                   427e
    LL operator() (int l, int r) { // [l, r)                       19f8
        return (val[r] - mul(val[l], pg[r - l]) + mod) % mod;      9986
    }                                                              95cf
} ha;                                                              b179
```

4 Math

4.1 Matrix powermod

```
const int MAXN = 105;                                              44b4
const LL modular = 1000000007;                                     92df
int n; // order of matrices                                        5c83
                                                                   427e
struct matrix{                                                     8864
    LL m[MAXN][MAXN];                                              3180
                                                                   427e
    void operator *=(matrix& a){                                   43c5
        static LL t[MAXN][MAXN];                                   e735
        Rep (i, n){                                                34d7
            Rep (j, n){                                            4c11
                t[i][j] = 0;                                       ee1e
                Rep (k, n){                                        c4a7
                    t[i][j] += (m[i][k] * a.m[k][j]) % modular;    fcaf
                    t[i][j] %= modular;                            199e
                }                                                  95cf
```

```
            }
        }
        memcpy(m, t, sizeof(t));
    }
};

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

4.2 Linear basis

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

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

4.3 Gauss elimination over finite field

```
const LL p = 1000000007;

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

typedef vector<LL> VLL;
typedef vector<VLL> VVLL;

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

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

    LL c = powmod(a[pk][pk], p - 2);
    det = det * a[pk][pk] % p;
    a[pk][pk] = 1;
    rep (j, n) a[pk][j] = a[pk][j] * c % p;
    rep (j, m) b[pk][j] = b[pk][j] * c % p;
```

```
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
  int calc(vector<int> &d, int pos) {                                    fa1a
    int ret = 0;                                                         5b57
    for (int i = 0; i < (int)a.size(); ++i) {                            501c
      if ((ret += (long long)d[pos - i] * a[i] % MOD) >= MOD) {          5de5
        ret -= MOD;                                                      3041
      }                                                                  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
  for (int i = 0; i < (int)na.size(); ++i) {                             d1a7
    na[i] = (long long)c * p.a[i] % MOD;                                 bf0c
  }                                                                      95cf
  return na;                                                             aaab
}                                                                        95cf
                                                                         427e
vector<int> solve(vector<int> a) {                                       afff
  int n = a.size();                                                      9f23
  Poly s, b;                                                             58d0
  s.a.push_back(1), b.a.push_back(1);                                    4e8f
  for (int i = 1, j = 0, ld = a[0]; i < n; ++i) {                        c2aa
    int d = s.calc(a, i);                                                4158
    if (d) {                                                             d503
      if ((s.length() - 1) * 2 <= i) {                                   c29d
        Poly ob = b;                                                     db9d
        b = s;                                                           6bce
        s = s - (long long)d * inverse(ld) % MOD * ob.move(i - j);       1d0e
        j = i;                                                           0889
        ld = d;                                                          64f1
      } else {                                                           8e2e
```

```
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

```
const int NMAX = 1<<20;                                              4e09
                                                                     427e
typedef complex<double> cplx;                                        3fbf
                                                                     427e
const double PI = 2*acos(0.0);                                       abd1
struct FFT{                                                          12af
    int rev[NMAX];                                                   c47c
    cplx omega[NMAX], oinv[NMAX];                                    27d7
    int K, N;                                                        9827
                                                                     427e
    FFT(int k){                                                      1442
        K = k; N = 1 << k;                                           e209
        rep (i, N){                                                  b393
            rev[i] = (rev[i>>1]>>1) | ((i&1)<<(K-1));                7ba3
            omega[i] = polar(1.0, 2.0 * PI / N * i);                 1908
            oinv[i] = conj(omega[i]);                                a166
        }                                                            95cf
    }                                                                95cf
                                                                     427e
    void dft(cplx* a, cplx* w){                                      b941
        rep (i, N) if (i < rev[i]) swap(a[i], a[rev[i]]);            a215
        for (int l = 2; l <= N; l *= 2){                             ac6e
            int m = l/2;                                             2969
            for (cplx* p = a; p != a + N; p += l)                    b3cf
                rep (k, m){                                          c24f
                    cplx t = w[N/l*k] * p[k+m];                      fe06
                    p[k+m] = p[k] - t; p[k] += t;                    ecbf
                }                                                    95cf
        }                                                            95cf
    }                                                                95cf
                                                                     427e
    void fft(cplx* a){dft(a, omega);}                                617b
    void ifft(cplx* a){                                              a123
        dft(a, oinv);                                                3b2f
        rep (i, N) a[i] /= N;                                        57fc
    }                                                                95cf
                                                                     427e
    void conv(cplx* a, cplx* b){                                     bdc0
        fft(a); fft(b);                                              6497
        rep (i, N) a[i] *= b[i];                                     12a5
        ifft(a);                                                     f84e
    }                                                                95cf
};                                                                   329b
```

4.7 Number theoretic transform

```
4ab9 const int NMAX = 1<<21;
427e
427e // 998244353 = 7*17*2^23+1, G = 3
fb9a const int P = 1004535809, G = 3; // = 479*2^21+1
427e
87ab struct NTT{
c47c     int rev[NMAX];
0eda     LL omega[NMAX], oinv[NMAX];
81af     int g, g_inv; // g: g_n = G^((P-1)/n)
9827     int K, N;
427e
2a2c     LL powmod(LL b, LL e){
95a2         LL r = 1;
3e90         while (e){
6624             if (e&1) r = r * b % P;
489e             b = b * b % P;
16fc             e >>= 1;
95cf         }
547e         return r;
95cf     }
427e
f420     NTT(int k){
e209         K = k; N = 1 << k;
7652         g = powmod(G, (P-1)/N);
4b3a         g_inv = powmod(g, N-1);
e04f         omega[0] = oinv[0] = 1;
b393         rep (i, N){
7ba3             rev[i] = (rev[i>>1]>>1) | ((i&1)<<(K-1));
ad4f             if (i){
8d8b                 omega[i] = omega[i-1] * g % P;
9e14                 oinv[i] = oinv[i-1] * g_inv % P;
95cf             }
95cf         }
95cf     }
427e
9668     void _ntt(LL* a, LL* w){
a215         rep (i, N) if (i < rev[i]) swap(a[i], a[rev[i]]);
ac6e         for (int l = 2; l <= N; l *= 2){
2969             int m = l/2;
7a1d             for (LL* p = a; p != a + N; p += l)
c24f                 rep (k, m){
                    LL t = w[N/l*k] * p[k+m] % P; 0ad3
                    p[k+m] = (p[k] - t + P) % P; 6209
                    p[k] = (p[k] + t) % P; fa1b
                } 95cf
        } 95cf
    } 95cf
 427e
    void ntt(LL* a){_ntt(a, omega);} 92ea
    void intt(LL* a){ 5daf
        LL inv = powmod(N, P-2); 1f2a
        _ntt(a, oinv); 9910
        rep (i, N) a[i] = a[i] * inv % P; a873
    } 95cf
 427e
    void conv(LL* a, LL* b){ 3a5b
        ntt(a); ntt(b); ad16
        rep (i, N) a[i] = a[i] * b[i] % P; e49e
        intt(a); 5748
    } 95cf
}; 329b
```

4.8 Sieve of Euler

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

4.9 Sieve of Euler (General)

```
b62e namespace sieve {
6589   constexpr int MAXN = 10000007;
e982   bool p[MAXN]; // true if not prime
6ae8   int prime[MAXN], sz;
cbf7   int pval[MAXN], pcnt[MAXN];
6030   int f[MAXN];
427e
76f6   void exec(int N = MAXN) {
9628     p[0] = p[1] = 1;
427e
8a8a     pval[1] = 1;
bdda     pcnt[1] = 0;
c6b9     f[1] = 1;
427e
a643     for (int i = 2; i < N; i++) {
01d6       if (!p[i]) {
b2b2         prime[sz++] = i;
37d9         for (LL j = i; j < N; j *= i) {
758c           int b = j / i;
81fd           pval[j] = i * pval[b];
e0f3           pcnt[j] = pcnt[b] + 1;
a96c           f[j] = ______; // f[j] = f(i^pcnt[j])
95cf         }
95cf       }
34c0       for (int j = 0; i * prime[j] < N; j++) {
f87a         int x = i * prime[j]; p[x] = 1;
20cc         if (i % prime[j] == 0) {
9985           pval[x] = pval[i] * prime[j];
3f93           pcnt[x] = pcnt[i] + 1;
8e2e         } else {
cc91           pval[x] = prime[j];
6322           pcnt[x] = 1;
95cf         }
6191         if (x != pval[x]) {
d614           f[x] = f[x / pval[x]] * f[pval[x]]
95cf         }
5f51         if (i % prime[j] == 0) break;
95cf       }
95cf     }
95cf   }
95cf }
```

4.10 Miller-Rabin primality test

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

4.11 Pollard's rho algorithm

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

```
      d = gcd(x>y ? x-y : y-x, n);
    }
  }
  return d;
}
```

4.12 Qusai-polynomial sum

Must call `init()` before use!

```
namespace polysum {
#define rep(i, a, n) for (int i = a; i < n; i++)
#define per(i, a, n) for (int i = n - 1; i >= a; i--)
const int D = 2010;
ll a[D], f[D], g[D], p[D], p1[D], p2[D], b[D], h[D][2], C[D];
ll powmod(ll a, ll b) {
  ll res = 1;
  a %= mod;
  assert(b >= 0);
  for (; b; b >>= 1) {
    if (b & 1) res = res * a % mod;
    a = a * a % mod;
  }
  return res;
}
ll calcn(int d, ll *a, ll n) {  // a[0].. a[d]  a[n]
  if (n <= d) return a[n];
  p1[0] = p2[0] = 1;
  rep(i, 0, d + 1) {
    ll t = (n - i + mod) % mod;
    p1[i + 1] = p1[i] * t % mod;
  }
  rep(i, 0, d + 1) {
    ll t = (n - d + i + mod) % mod;
    p2[i + 1] = p2[i] * t % mod;
  }
  ll ans = 0;
  rep(i, 0, d + 1) {
    ll t = g[i] * g[d - i] % mod * p1[i] % mod * p2[d - i] % mod * a[i] % mod;
    if ((d - i) & 1)
      ans = (ans - t + mod) % mod;
    else
      ans = (ans + t) % mod;
  }
  return ans;
}
void init(int M) {
  f[0] = f[1] = g[0] = g[1] = 1;
  rep(i, 2, M + 5) f[i] = f[i - 1] * i % mod;
  g[M + 4] = powmod(f[M + 4], mod - 2);
  per(i, 1, M + 4) g[i] = g[i + 1] * (i + 1) % mod;
}
ll polysum(ll m, ll *a, ll n) {  // a[0].. a[m] \sum_{i=0}^{n-1} a[i]
  ll b[D];
  for (int i = 0; i <= m; i++) b[i] = a[i];
  b[m + 1] = calcn(m, b, m + 1);
  rep(i, 1, m + 2) b[i] = (b[i - 1] + b[i]) % mod;
  return calcn(m + 1, b, n - 1);
}
ll qpolysum(ll R, ll n, ll *a, ll m) {  // a[0].. a[m] \sum_{i=0}^{n-1} a[i]*R^i
  if (R == 1) return polysum(n, a, m);
  a[m + 1] = calcn(m, a, m + 1);
  ll r = powmod(R, mod - 2), p3 = 0, p4 = 0, c, ans;
  h[0][0] = 0;
  h[0][1] = 1;
  rep(i, 1, m + 2) {
    h[i][0] = (h[i - 1][0] + a[i - 1]) * r % mod;
    h[i][1] = h[i - 1][1] * r % mod;
  }
  rep(i, 0, m + 2) {
    ll t = g[i] * g[m + 1 - i] % mod;
    if (i & 1)
      p3 = ((p3 - h[i][0] * t) % mod + mod) % mod,
      p4 = ((p4 - h[i][1] * t) % mod + mod) % mod;
    else
      p3 = (p3 + h[i][0] * t) % mod, p4 = (p4 + h[i][1] * t) % mod;
  }
  c = powmod(p4, mod - 2) * (mod - p3) % mod;
  rep(i, 0, m + 2) h[i][0] = (h[i][0] + h[i][1] * c) % mod;
  rep(i, 0, m + 2) C[i] = h[i][0];
  ans = (calcn(m, C, n) * powmod(R, n) - c) % mod;
  if (ans < 0) ans += mod;
  return ans;
}
}  // namespace polysum
```

5 Graph Theory

5.1 Strongly connected component

```
837c const int MAXV = 100005;
427e 
2ea0 struct graph{
88e3     vector<int> adj[MAXV];
9cad     stack<int> s;
3d02     int V; // number of vertices
8b6c     int pre[MAXV], lnk[MAXV], scc[MAXV];
27ee     int time, sccn;
427e 
bfab     void add_edge(int u, int v){
c71a         adj[u].push_back(v);
95cf     }
427e 
d714     void dfs(int u){
7e41         pre[u] = lnk[u] = ++time;
80f6         s.push(u);
18f6         for (int v : adj[u]){
173e             if (!pre[v]){
5f3c                 dfs(v);
002c                 lnk[u] = min(lnk[u], lnk[v]);
6068             } else if (!scc[v]){
d5df                 lnk[u] = min(lnk[u], pre[v]);
95cf             }
95cf         }
8de2         if (lnk[u] == pre[u]){
660f             sccn++;
3c9e             int x;
a69f             do {
3834                 x = s.top(); s.pop();
b0e9                 scc[x] = sccn;
6757             } while (x != u);
95cf         }
95cf     }
427e 
4c88     void find_scc(){
f4a2         time = sccn = 0;
8de7         memset(scc, 0, sizeof scc);
8c2f         memset(pre, 0, sizeof pre);
6901         Rep (i, V){
                 if (!pre[i]) dfs(i);                       56d1
             }                                              95cf
         }                                                  95cf
                                                            427e
         vector<int> adjc[MAXV];                            27ce
         void contract(){                                   364d
             Rep (i, V)                                     1a1e
                 rep (j, adj[i].size()){                    21a2
                     if (scc[i] != scc[adj[i][j]])          b730
                         adjc[scc[i]].push_back(scc[adj[i][j]]);  b46e
                 }                                          95cf
         }                                                  95cf
     };                                                     329b
```

5.2 Vertex biconnected component

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

```
                while (1) {
                    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, $O(\sqrt{V}E)$.

```
bcf8 struct edge{
60e2     int from, to;
5e6d     LL cap, flow;
329b };
427e
e2cd const int MAXN = 1005;
9062 struct Dinic {
4dbf     int n, m, s, t;
9f0c     vector<edge> edges;
b891     vector<int> G[MAXN];
bbb6     bool vis[MAXN];
b40a     int d[MAXN];
ddec     int cur[MAXN];
427e
5973     void add_edge(int from, int to, LL cap) {
7b55         edges.push_back(edge{from, to, cap, 0});
1db7         edges.push_back(edge{to, from, 0, 0});
fe77         m = edges.size();
dff5         G[from].push_back(m-2);
8f2d         G[to].push_back(m-1);
95cf     }
427e
1836     bool bfs() {
3b73         memset(vis, 0, sizeof(vis));
93d2         queue<int> q;
5d13         q.push(s);
2cd2         vis[s] = 1;
721d         d[s] = 0;
cc78         while (!q.empty()) {
66ba             int x = q.front(); q.pop();
3b61             for (int i = 0; i < G[x].size(); i++) {
b510                 edge& e = edges[G[x][i]];
bba9                 if (!vis[e.to] && e.cap > e.flow) {
cd72                     vis[e.to] = 1;
cf26                     d[e.to] = d[x] + 1;
ca93                     q.push(e.to);
                    }                                                                    95cf
                }                                                                        95cf
            }                                                                            95cf
            return vis[t];                                                               b23b
        }                                                                                95cf
                                                                                         427e
        LL dfs(int x, LL a) {                                                            9252
            if (x == t || a == 0) return a;                                              6904
            LL flow = 0, f;                                                              8bf9
            for (int& i = cur[x]; i < G[x].size(); i++) {                                f515
                edge& e = edges[G[x][i]];                                                b510
                if(d[x] + 1 == d[e.to] && (f = dfs(e.to, min(a, e.cap-e.flow))) > 0)     2374
                   {
                    e.flow += f;                                                         1cce
                    edges[G[x][i]^1].flow -= f;                                          e16d
                    flow += f;                                                           a74d
                    a -= f;                                                              23e5
                    if(a == 0) break;                                                    97ed
                }                                                                        95cf
            }                                                                            95cf
            return flow;                                                                 84fb
        }                                                                                95cf
                                                                                         427e
        LL max_flow(int s, int t) {                                                      5bf2
            this->s = s; this->t = t;                                                    590d
            LL flow = 0;                                                                 62e2
            while (bfs()) {                                                              ed58
                memset(cur, 0, sizeof(cur));                                             f326
                flow += dfs(s, LLONG_MAX);                                               fb3a
            }                                                                            95cf
            return flow;                                                                 84fb
        }                                                                                95cf
                                                                                         427e
        vector<int> min_cut() { // call this after maxflow                               c72e
            vector<int> ans;                                                             1df9
            for (int i = 0; i < edges.size(); i++) {                                     df9a
                edge& e = edges[i];                                                      56d8
                if(vis[e.from] && !vis[e.to] && e.cap > 0) ans.push_back(i);             46a2
            }                                                                            95cf
            return ans;                                                                  4206
        }                                                                                95cf
    };                                                                                   329b
```

5.5 Maximum cardinality bipartite matching (Hungarian)

```
302f #include <bits/stdc++.h>
421c using namespace std;
427e
0d6c #define rep(i, n) for (int i = 0; i < (n); i++)
cfe3 #define Rep(i, n) for (int i = 1; i <= (n); i++)
8843 #define range(x) (x).begin(), (x).end()
5cad typedef long long LL;
427e
84ee struct Hungarian{
fbf6     int nx, ny;
9ec6     vector<int> mx, my;
9d4c     vector<vector<int> > e;
edec     vector<bool> mark;
427e
8324     void init(int nx, int ny){
c1d1         this->nx = nx;
f9c1         this->ny = ny;
ac92         mx.resize(nx); my.resize(ny);
3f11         e.clear(); e.resize(nx);
1023         mark.resize(nx);
95cf     }
427e
4589     inline void add(int a, int b){
486c         e[a].push_back(b);
95cf     }
427e
0c2b     bool augment(int i){
207c         if (!mark[i]) {
dae4             mark[i] = true;
6a1e             for (int j : e[i]){
0892                 if (my[j] == -1 || augment(my[j])){
9ca3                     mx[i] = j; my[j] = i;
3361                     return true;
95cf                 }
95cf             }
95cf         }
438e         return false;
95cf     }
427e
3fac     int match(){
5b57         int ret = 0;
        fill(range(mx), -1);    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 Minimum cost maximum flow

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

```
2dae         p[s] = 0; a[s] = INT_MAX;
cc78         while (!q.empty()){
b0aa             int u = q.front(); q.pop(); inq[u] = false;
3bba             for (int i : G[u]) {
56d8                 edge& e = edges[i];
3601                 if (e.cap > e.flow && d[e.to] > d[u] + e.cost){
55bc                     d[e.to] = d[u] + e.cost;
0bea                     p[e.to] = G[u][i];
8249                     a[e.to] = min(a[u], e.cap - e.flow);
e5d3                     if (!inq[e.to]) q.push(e.to), inq[e.to] = true;
95cf                 }
95cf             }
95cf         }
6d7c         return d[t] != INF;
95cf     }
427e
71a4     void augment(){
06f1         int u = t;
b19d         while (u != s){
db09             edges[p[u]].flow += a[t];
25a9             edges[p[u]^1].flow -= a[t];
e6c9             u = edges[p[u]].from;
95cf         }
95cf     }
427e
6e20 #ifdef GIVEN_FLOW
5972     bool min_cost(int s, int t, int f, LL& cost) {
590d         this->s = s; this->t = t;
21d4         int flow = 0;
23cb         cost = 0;
22dc         while (spfa()) {
bcdb             augment();
a671             if (flow + a[t] >= f){
b14d                 cost += (f - flow) * d[t]; flow = f;
3361                 return true;
8e2e             } else {
2a83                 flow += a[t]; cost += a[t] * d[t];
95cf             }
95cf         }
438e         return false;
95cf     }
a8cb #else
f9a9     int min_cost(int s, int t, LL& cost) {
590d         this->s = s; this->t = t;
```

```
        int flow = 0;                                      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
```

5.7 Global minimum cut (Stoer-Wagner)

```
typedef vector<LL> VI;                                              f9d7
typedef vector<VI> VVI;                                             045e
                                                                    427e
pair<LL, VI> stoer(VVI &w) {                                        f012
    int n = w.size();                                               66f7
    VI used(n), c, bestc;                                           4d98
    LL bestw = -1;                                                  329d
                                                                    427e
    for (int ph = n - 1; ph >= 0; ph--) {                           cd21
        VI wt = w[0], added = used;                                 ec6e
        int prev, last = 0;                                         f20e
        rep (i, ph) {                                               4b32
            prev = last;                                            8bfc
            last = -1;                                              0706
            for (int j = 1; j < n; j++)                             4942
                if (!added[j] && (last == -1 || wt[j] > wt[last]))  c4b9
                    last = j;                                       887d
            if (i == ph - 1) {                                      71bc
                rep (j, n) w[prev][j] += w[last][j];                9cfa
                rep (j, n) w[j][prev] = w[prev][j];                 1f25
                used[last] = true;                                  5613
                c.push_back(last);                                  8e11
                if (bestw == -1 || wt[last] < bestw) {              bb8e
                    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    }
038c    return {bestw, bestc};
95cf }
```

5.8 Heavy-light decomposition

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

```
0f42 const int MAXN = 100005;
0b32 vector<int> adj[MAXN];
42f2 int sz[MAXN], top[MAXN], fa[MAXN], son[MAXN], depth[MAXN], id[MAXN];
427e
be5c void dfs1(int x, int dep, int par){
7489     depth[x] = dep;
2ee7     sz[x] = 1;
adb4     fa[x] = par;
b79d     int maxn = 0, s = 0;
c861     for (int c: adj[x]){
fe45         if (c == par) continue;
fd2f         dfs1(c, dep + 1, x);
b790         sz[x] += sz[c];
f0f1         if (sz[c] > maxn){
c749             maxn = sz[c];
fe19             s = c;
95cf         }
95cf     }
0e08     son[x] = s;
95cf }
427e
ba54 int cid = 0;
3644 void dfs2(int x, int t){
8d96     top[x] = t;
d314     id[x] = ++cid;
c4a1     if (son[x]) dfs2(son[x], t);
c861     for (int c: adj[x]){
9881         if (c == fa[x]) continue;
5518         if (c == son[x]) continue;
13f9         else dfs2(c, c);
95cf     }
95cf }
```

```
                                                            427e
void decomp(int root){                                      0f04
    dfs1(root, 1, 0);                                       9fa4
    dfs2(root, root);                                       1c88
}                                                           95cf
                                                            427e
void query(int u, int v){                                   2c98
    while (top[u] != top[v]){                               03a1
        if (depth[top[u]] < depth[top[v]]) swap(u, v);      45ec
        // id[top[u]] to id[u]                              427e
        u = fa[top[u]];                                     005b
    }                                                       95cf
    if (depth[u] > depth[v]) swap(u, v);                    6083
    // id[u] to id[v]                                       427e
}                                                           95cf
```

5.9 Centroid decomposition

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

Usage:

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

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

```
vector<int> adj[100005];                        1fb6
int sz[100005], sum;                            88e0
                                                427e
void getsz(int u, int p) {                      f93d
  sz[u] = 1; sum++;                             5b36
  for (int v : adj[u]) {                        18f6
    if (v == p) continue;                       bd87
    getsz(v, u);                                e3cb
    sz[u] += sz[v];                             8449
  }                                             95cf
}                                               95cf
                                                427e
int getcent(int u, int p) {                     67f9
  for (int v : adj[u])                          d51f
    if (v != p and sz[v] > sum / 2)             76e4
      return getcent(v, u);                     18e3
  return u;                                     81b0
```

```
95cf }
427e
4662 void decompose(int u) {
618e   sum = 0; getsz(u, 0);
303c   u = getcent(u, 0); // update u to the centroid
427e
18f6   for (int v : adj[u]) {
427e     // get answer for subtree v
95cf   }
427e   // get answer for the whole tree
427e   // don't forget to count the centroid itself
427e
18f6   for (int v : adj[u]) { // divide and conquer
c375     adj[v].erase(find(range(adj[v]), u));
fa6b     decompose(v);
a717     adj[v].push_back(u); // restore deleted edge
95cf   }
95cf }
```

5.10 DSU on tree

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

Usage:

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

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

```
1fb6 vector<int> adj[100005];
901d int sz[100005], son[100005];
427e
5559 void decomp(int u, int p) {
50c0     sz[u] = 1;
18f6     for (int v : adj[u]) {
bd87         if (v == p) continue;
a851         decomp(v, u);
8449         sz[u] += sz[v];
d28c         if (sz[v] > sz[son[u]]) son[u] = v;
95cf     }
95cf }
427e
b7ec template <typename T>
62f5 void trav(T fn, int u, int p) {
4412     fn(u);
30b3     for (int v : adj[u]) if (v != p) trav(fn, v, u);
95cf }
427e
7467 #define for_light(v) for (int v : adj[u]) if (v != p and v != son[u])
33ff void work(int u, int p, bool keep) {
72a2     for_light(v) work(v, u, 0); // process light children
427e
427e     // process heavy child
427e     // current data structure contains info of heavy child
9866     if (son[u]) work(son[u], u, 1);
427e
18a9     auto merge = [u] (int c) { /* count contribution of c */ };
1ab0     auto enter = [] (int c) { /* add vertex c */ };
f241     auto leave = [] (int c) { /* remove vertex c*/ };
427e
3d3b     for_light(v) {
74c6         trav(merge, v, u);
c13d         trav(enter, v, u);
95cf     }
427e
427e     // count answer for root and add it
427e     // Warning: special check may apply to root!
c54f     merge(u);
9dec     enter(u);
427e
427e     // leave current tree
4e3e     if (!keep) trav(leave, u, p);
95cf }
```

6 Data Structures

6.1 Fenwick tree (point update range query)

```
9976 struct bit_purq { // point update, range query
d7af     int N;
99ff     vector<LL> tr;
```

```
427e
d34f     void init(int n) { // fill the array with 0
1010         tr.resize(N = n + 5);
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) {
```

```
        while (n){                                  e290
            tr[n] += x;                             6c81
            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]);
      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 {
      push(o);
      if (rr <= m[o]) return query(lson, ll, rr);
      if (ll >= m[o]) return query(rson, ll, rr);
      return query(lson, ll, rr) + query(rson, ll, rr);
    }
  }
} seg;
```

6.4 Link/cut tree

Usage:

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

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

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

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

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

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

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

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

```
      chroot(x);                                           3015
      access(y);                                           29b5
    }                                                      95cf
                                                           427e
    int Root(int x) {                                      d87a
      access(x);                                           766a
      while (ch[x][0]) {                                   874d
        push(x);                                           a97b
        x = ch[x][0];                                      b83a
      }                                                    95cf
      splay(x);                                            8fec
      return x;                                            d074
    }                                                      95cf
                                                           427e
    void Link(int u, int v) {  // assume unconnected before  70d3
      chroot(u);                                           b8a5
      fa[u] = v;                                           2448
    }                                                      95cf
                                                           427e
    void Cut(int u, int v) {  // assume connected before   c2f4
      split(u, v);                                         e8ce
      fa[u] = ch[v][0] = 0;                                fd95
      pull(v);                                             743b
    }                                                      95cf
                                                           427e
    int Query(int u, int v) {                              6ca2
      split(u, v);                                         e8ce
      return sum[v];                                       a5ba
    }                                                      95cf
                                                           427e
    void Update(int u, int x) {                            eaba
      splay(u);                                            46ce
      val[u] = x;                                          1d62
    }                                                      95cf
};                                                         329b
```

6.5 Balanced binary search tree from pb_ds

```
#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>
  rkt;
// null_tree_node_update

// 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 Sparse table, range extremum query

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

```
const int MAXN = 100007;
```

```
int a[MAXN];
int st[MAXN][32 - __builtin_clz(MAXN)];

inline int ext(int x, int y){return x>y?x:y;} // ! max

void init(int n){
    int l = 31 - __builtin_clz(n);
    rep (i, n) st[i][0] = a[i];
    rep (j, l)
        rep (i, 1+n-(1<<j))
            st[i][j+1] = ext(st[i][j], st[i+(1<<j)][j]);
}

int rmq(int l, int r){
    int k = 31 - __builtin_clz(r-l+1);
    return ext(st[l][k], st[r-(1<<k)+1][k]);
}
```

7 Geometrics

7.1 2D geometric template

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

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

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

typedef pair<pt, pt> seg;

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

// 0 not on segment
// 1 on segment except vertices
// 2 on vertices
int ptOnSeg2(pt& p, seg& s){
    vec v1 = s.first - p, v2 = s.second - p;
    T ip = (v1, v2);
    if (v1 * v2 != 0 || ip > 0) return 0;
    return (v1, v2) ? 1 : 2;
}

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

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

inline bool intersect(seg a, seg b){
    // ! if (nIntRectRect(a, b)) return false; // if commented, assume that a
      and b are non-collinear
    return rotOrder(b.first-a.first, a.second-a.first, b.second-a.first) >= 0 &&
           rotOrder(a.first-b.first, b.second-b.first, a.second-b.first) >= 0;
}

// 0 not insersect
// 1 standard intersection
// 2 vertex-line intersection
// 3 vertex-vertex intersection
// 4 collinear and have common point(s)
int intersect2(seg& a, seg& b){
    if (nIntRectRect(a, b)) return 0;
    vec va = a.second - a.first, vb = b.second - b.first;
    double j1 = rotOrder(b.first-a.first, va, b.second-a.first),
           j2 = rotOrder(a.first-b.first, vb, a.second-b.first);
    if (j1 < 0 || j2 < 0) return 0;
```

```
9400	    if (j1 != 0 && j2 != 0) return 1;
83db	    if (j1 == 0 && j2 == 0){
6b0c	        if (va * vb == 0) return 4; else return 3;
fb17	    } else return 2;
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

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