Pollard

Team Reference

```cpp
const int max_step = 4e5;

unsigned long long gcd(unsigned long
    long a, unsigned long long b){
    if (!a) return 1;
    while (a) swap(a, b%=a);
    return b;
}

unsigned long long get(unsigned long
    long a, unsigned long long b){
    if (a > b)
        return a-b;
    else
        return b-a;
}

unsigned long long pollard(unsigned long
    long n){
    unsigned long long x = (rand() + 1)
        % n, y = 1, g;
    int stage = 2, i = 0;
    g = gcd(get(x, y), n);
    while (g == 1){
        if (i == max_step)
            break;
        if (i == stage){
            y = x;
            stage <<= 1;
        }
        x = (x * (__int128)x + 1) % n;
        i++;
        g = gcd(get(x, y), n);
    }
    return g;
}
```

pragma

```
#pragma GCC optimize(''O3,no-stack-protector'')
#pragma GCC target(''sse,sse2,sse4,ssse3,popcnt,abm,mmx,avx,tune=native'')
```

Алгебра Pick

В + Г / 2 − 1 = AREA,

где В — количество целочисленных точек внутри многоугольника, а Г — количество целочисленных точек на границе многоугольника.

Newton

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

Catalan

$$C_n = \sum_{k=0}^{n-1} C_k C_{n-1-k}$$

$$C_i = \frac{1}{n+1}\binom{2n}{n}$$

Кол-во графов

$$G_N := 2^{n(n-1)/2}$$

Количество связных помеченных графов

$$Conn_N = G_N - \frac{1}{N}\sum_{K=1}^{N-1} K\binom{N}{K}Conn_K G_{N-K}$$

Количество помеченных графов с K компонентами связности

$$D[N][K] = \sum_{S=1}^{N}\binom{N-1}{S-1}Conn_S D[N-S][K-1]$$

Miller-Rabbin

```
a=a^t
FOR i = 1...s
    if a^2=1 && |a|!=1
        NOT PRIME
    a=a^2
return a==1 ? PRIME : NOT PRIME
```

Интегрирование по формуле Симпсона

$\int_a^b f(x)dx$?

$x_i := a + ih, i = 0\ldots 2n$

$h = \frac{b-a}{2n}$

$\int = (f(x_0)+4f(x_1)+2f(x_2)+4f(x_3)+2f(x_4)+\ldots+4f(x_{2n-1})+f(x_{2n}))\frac{h}{3}$

$O(n^4)$.

Простые числа

1009,1013;10007,10009;100003,100019
1000003,1000033;10000019,10000079
100000007,100000037
10000000019,10000000033
1000000000039,1000000000061
100000000000031,100000000000067
10000000000000061,10000000000000069
1000000000000000003,1000000000000000009

Числа для Фурье

- prime: $7340033 = 7\cdot2^{20} + 1; w : 5(w^{2^{20}} = 1)$

- prime: $13631489 = 13\cdot2^{20} + 1; w : 3(w^{2^{20}} = 1)$

- prime: $23068673 = 11\cdot2^{21}+1; w : 38(w^{2^{21}} = 1)$

- prime: $69206017 = 33\cdot2^{21}+1; w : 45(w^{2^{21}} = 1)$

- prime: $81788929 = 39\cdot2^{21}+1; w : 94(w^{2^{21}} = 1)$

- prime: $104857601 = 25\cdot2^{22} + 1; w : 21(w^{2^{22}} = 1)$

- prime: $113246209 = 27\cdot2^{22} + 1; w : 66(w^{2^{22}} = 1)$

- prime: $138412033 = 33\cdot2^{22} + 1; w : 30(w^{2^{22}} = 1)$

- prime: $167772161 = 5\cdot2^{25}+1; w : 17(w^{2^{25}} = 1)$

- prime: $469762049 = 7\cdot2^{26}+1; w : 30(w^{2^{26}} = 1)$

- prime: $998244353 = 7\cdot17\cdot2^{23} + 1; w : 3^{7*17}$.

Erdős–Gallai theorem

A sequence of non-negative integers $d_1 \geq \cdots \geq d_n$ can be represented as the degree sequence of a finite simple graph on n vertices if and only if $d_1 + \cdots + d_n$ $d_1 + \cdots + d_n$ is even and

$\sum_{i=1}^k d_i \leq k(k-1) + \sum_{i=k+1}^n \min(d_i, k)$ holds for every k in $1 \leq k \leq n$.

```cpp
/** Begin fast allocation */
const int MAX_MEM = 5e8;
int mpos = 0;
char mem[MAX_MEM];
inline void * operator new ( size_t n )
    {
    assert((mpos += n) <= MAX_MEM);
    return (void *)(mem + mpos - n);
}
inline void operator delete ( void * )
    noexcept { } // must have!

/** End fast allocation */


#define pb push_back
#define mp make_pair
#define fst first
#define snd second
#define ll long long
#define forn(i, n) for (int i = 0; i <
    (int) (n); i++)
#define forlr(i, l, r) for (int i =
    (int) l; i <= (int) (r); i++)
#define forrl(i, r, l) for (int i =
    (int) r; i >= (int) (l); i--)

/** Interface */

inline int readChar();
template <class T = int> inline T
    readInt();
template <class T> inline void writeInt(
    T x, char end = 0 );
inline void writeChar( int x );
inline void writeWord( const char *s );

/** Read */

static const int buf_size = 2048;

inline int getChar() {
    static char buf[buf_size];
    static int len = 0, pos = 0;
    if (pos == len)
        pos = 0, len = fread(buf, 1,
            buf_size, stdin);
    if (pos == len)
        return -1;
    return buf[pos++];
}

inline int readWord(char * buffer) {
    int c = getChar();
    while (c <= 32) {
        c = getChar();
    }

    int len = 0;
    while (c > 32) {
        *buffer = (char) c;
        c = getChar();
        buffer++;
        len++;
    }
    return len;
}

inline int readChar() {
    int c = getChar();
    while (c <= 32)
        c = getChar();
    return c;
}

template <class T>
inline T readInt() {
    int s = 1, c = readChar();
    T x = 0;
    if (c == '-')
        s = -1, c = getChar();
    while ('0' <= c && c <= '9')
        x = x * 10 + c - '0', c =
            getChar();
    return s == 1 ? x : -x;
}

/** Write */

static int write_pos = 0;
static char write_buf[buf_size];

inline void writeChar( int x ) {
    if (write_pos == buf_size)
        fwrite(write_buf, 1, buf_size,
            stdout), write_pos = 0;
    write_buf[write_pos++] = x;
}

template <class T>
inline void writeInt( T x, char end ) {
    if (x < 0)
```

```
93          writeChar('-'), x = -x;
94
95      char s[24];
96      int n = 0;
97      while (x || !n)
98          s[n++] = '0' + x % 10, x /= 10;
99      while (n--)
100         writeChar(s[n]);
101     if (end)
102         writeChar(end);
103 }
104
105 inline void writeWord( const char *s ) {
106     while (*s)
107         writeChar(*s++);
108 }
109
110 struct Flusher {
111     ~Flusher() {
112         if (write_pos)
113             fwrite(write_buf, 1,
               ↪  write_pos, stdout),
               ↪  write_pos = 0;
114     }
115 } flusher;
```

```
1  struct line {
2      ll k, b;
3      ll at(ll x) const {
4          return k * x + b;
5      }
6  };
7  double intersec(line a, line b) {
8      return 1.0 * (b.b - a.b) / (a.k -
          ↪  b.k);
9  }
10 struct convex_hull_trick {
11     vector<double> x = {-1e18};
12     vector<line> lines;
13     void add(line l) {
14         // l.k increasing
15         if (lines.empty()) {
16             lines.pb(l);
17         } else {
18             while (lines.size() > 1 &&
               ↪  intersec(l,
               ↪  lines[lines.size() - 2])
               ↪  < x.back()) {
19                 lines.pop_back();
20                 x.pop_back();
21             }
22             x.push_back(intersec(l,
               ↪  lines.back()));
23             lines.push_back(l);
24         }
25     }
26 };
27 struct cht_forward_iterator {
28     int ci = 0;
29     ll promote(const convex_hull_trick&
          ↪  cht, ll value) {
30         while (ci < (int)cht.x.size() -
           ↪  1 && cht.x[ci + 1] < value)
31             ci++;
32         return cht.lines[ci].at(value);
33     }
34 };
35 const int g = 275;
36 struct sqrt_dec {
37     struct block {
38         vector<int> bs;
39         convex_hull_trick cht;
40         cht_forward_iterator it;
41         block(vector<int> b) {
42             bs = b;
43             int k = b.size();
44             forn(i, k) {
```

```
45              cht.add({ b[i], 1ll *
                ↪  b[i] * (k - i) });
46          }
47       }
48        ll value(ll x) {
49           if (cht.lines.empty())
50               return 0;
51           return it.promote(cht, x);
52        }
53     };
54      void insert(block& bl, int b) {
55         auto v = std::move(bl.bs);
56         v.insert(lower_bound(all(v), b),
                ↪  b);
57         bl = block(std::move(v));
58      }
59      vector<block> blocks;
60      sqrt_dec(int n) {
61         blocks.resize((n + g - 1) / g,
                ↪  block({}));
62      }
63      void add(int i, int b) {
64         insert(blocks[i / g], b);
65      }
66      ll get_max() {
67         int sm = 0;
68         ll ans = 0;
69         for (int i = blocks.size() - 1;
                ↪  i >= 0; i--) {
70            ans = max(ans,
                ↪  blocks[i].value(sm));
71            sm += blocks[i].bs.size();
72         }
73         return ans;
74      }
75 };
```

```
1  struct treap{
2      map<char, int> go;
3      int len, suff;
4      long long sum_in;
5      treap(){}
6  };
7
8  treap v[max_n * 4];
9  int last = 0;
10
11 int add_treap(int max_len){
12     v[number].sum_in = 0;
13     v[number].len = max_len;
14     v[number].suff = -1;
15     number++;
16     return number - 1;
17 }
18
19 void add_char(char c){
20     int cur = last;
21     int new_treap = add_treap(v[cur].len
           ↪  + 1);
22     last = new_treap;
23     while (cur != -1){
24        if (v[cur].go.count(c) == 0){
25            v[cur].go[c] = new_treap;
26            v[new_treap].sum_in +=
                ↪  v[cur].sum_in;
27            cur = v[cur].suff;
28            if (cur == -1)
29                v[new_treap].suff = 0;
30        }else{
31            int a = v[cur].go[c];
32            if (v[a].len == v[cur].len +
                ↪  1){
33                v[new_treap].suff = a;
34            }else{
35                int b =
                ↪  add_treap(v[cur].len
                ↪  + 1);
36                v[b].go = v[a].go;
37                v[b].suff = v[a].suff;
38                v[new_treap].suff = b;
39                while (cur != -1 &&
                ↪  v[cur].go.count(c)
                ↪  != 0 && v[cur].go[c]
                ↪  == a){
40                    v[cur].go[c] = b;
41                    v[a].sum_in -=
                    ↪  v[cur].sum_in;
```

```
42              v[b].sum_in +=
                 ↪  v[cur].sum_in;
43              cur = v[cur].suff;
44            }
45            v[a].suff = b;
46          }
47        return;
48      }
49    }
50  }
```

```
1  int k = sqrt((double)p) + 2;
2
3  for (int i = k; i >= 1; i--)
4      mp[bin(b, (i * 1ll * k) % (p-1), p)]
        ↪  = i;
5
6  bool answered = false;
7  int ans = INT32_MAX;
8  for (int i = 0; i <= k; i++){
9      int sum = (n * 1ll * bin(b, i, p)) %
        ↪  p;
10      if (mp.count(sum) != 0){
11          int an = mp[sum] * 1ll * k - i;
12          if (an < p)
13              ans = min(an, ans);
14      }
15  }
```

```cpp
int gcd (int a, int b, int & x, int & y) {
    if (a == 0) {
        x = 0; y = 1;
        return b;
    }
    int x1, y1;
    int d = gcd (b%a, a, x1, y1);
    x = y1 - (b / a) * x1;
    y = x1;
    return d;
}
```

linear sieve

```cpp
const int N = 1000000;

int pr[N + 1], sz = 0;
/* minimal prime, mobius function, euler
    function */
int lp[N + 1], mu[N + 1], phi[N + 1];

lp[1] = mu[1] = phi[1] = 1;
for (int i = 2; i <= N; ++i) {
    if (lp[i] == 0)
        lp[i] = pr[sz++] = i;
    for (int j = 0; j < sz && pr[j]
        <= lp[i] && i * pr[j] <= N;
        ++j)
            lp[i * pr[j]] = pr[j];

    mu[i] = lp[i] == lp[i / lp[i]] ? 0 :
        -1 * mu[i / lp[i]];
    phi[i] = phi[i / lp[i]] * (lp[i] ==
        lp[i / lp[i]] ? lp[i] : lp[i] -
        1);
}
```

kasai

```cpp
// Edmonds O(n^3) begins

vector<int> gr[MAXN];
int match[MAXN], p[MAXN], base[MAXN],
    q[MAXN];
bool used[MAXN], blossom[MAXN];
int mark[MAXN];
int C = 1;

int lca(int a, int b) {
    C++;
    for (;;) {
        a = base[a];
        mark[a] = C;
        if (match[a] == -1) break;
        a = p[match[a]];
    }

    for (;;) {
        b = base[b];
        if (mark[b] == C) return b;
        b = p[match[b]];
    }
}

void mark_path(int v, int b, int
    children) {
    while (base[v] != b) {
        blossom[base[v]] =
            blossom[base[match[v]]] =
            true;
        p[v] = children;
        children = match[v];
        v = p[match[v]];
    }
}

int find_path(int root) {
    memset(used, 0, sizeof(used));
    memset(p, -1, sizeof p);
    for (int i = 0; i < N; i++)
        base[i] = i;

    used[root] = true;
    int qh = 0, qt = 0;
    q[qt++] = root;
    while (qh < qt) {
        int v = q[qh++];
        for (int to : gr[v]) {
            if (base[v] == base[to] ||
                match[v] == to)
                continue;
            if (to == root || match[to]
                != -1 && p[match[to]] !=
                -1) {
                int curbase = lca(v,
                    to);
                memset(blossom, 0,
                    sizeof(blossom));
                mark_path(v, curbase,
                    to);
```

```
51              mark_path(to, curbase,
                 ↪  v);
52              for (int i = 0; i < N;
                 ↪  i++)
53                  if
                     ↪  (blossom[base[i]])
                     ↪  {
54                      base[i] =
                         ↪  curbase;
55                      if (!used[i]) {
56                          used[i] =
                             ↪  true;
57                          q[qt++] = i;
58                      }
59                  }
60          } else if (p[to] == -1) {
61              p[to] = v;
62              if (match[to] == -1)
63                  return to;
64              to = match[to];
65              used[to] = true;
66              q[qt++] = to;
67          }
68      }
69  }

71  return -1;
72  }

74  memset(match, -1, sizeof match);
75      for (int i = 0; i < N; i++) {
76          if (match[i] == -1 &&
             ↪  !gr[i].empty()) {
77              int v = find_path(i);
78              while (v != -1) {
79                  int pv = p[v], ppv =
                     ↪  match[pv];
80                  match[v] = pv; match[pv]
                     ↪  = v;
81                  v = ppv;
82              }
83          }
84      }
85  // Edmonds O(n^3) ends

1   fill(par, par + 301, -1);
2   fill(par2, par2 + 301, -1);

4   int ans = 0;
5   for (int v = 0; v < n; v++){
6       memset(useda, false, sizeof(useda));
```

```
7   memset(usedb, false, sizeof(usedb));
8   useda[v] = true;
9   for (int i = 0; i < n; i++)
10      w[i] = make_pair(a[v][i] +
         ↪  row[v] + col[i], v);
11  memset(prev, 0, sizeof(prev));
12  int pos;
13  while (true){
14      pair<pair<int, int>, int> p =
         ↪  make_pair(make_pair(1e9,
         ↪  1e9), 1e9);
15      for (int i = 0; i < n; i++)
            if (!usedb[i])
                p = min(p,
                 ↪  make_pair(w[i], i));
16      for (int i = 0; i < n; i++)
            if (!useda[i])
                row[i] += p.first.first;
17      for (int i = 0; i < n; i++)
            if (!usedb[i]){
                col[i] -= p.first.first;
                w[i].first -=
                 ↪  p.first.first;
            }
26      ans += p.first.first;
27      usedb[p.second] = true;
28      prev[p.second] = p.first.second;
         ↪  //из второй в первую
29      int x = par[p.second];
30      if (x == -1){
31          pos = p.second;
32          break;
33      }
34      useda[x] = true;
35      for (int j = 0; j < n; j++)
36          w[j] = min(w[j], {a[x][j] +
             ↪  row[x] + col[j], x});
38  }
39  while (pos != -1){
40      int nxt = par2[prev[pos]];
41      par[pos] = prev[pos];
42      par2[prev[pos]] = pos;
43      pos = nxt;
44  }
45  }
46  cout << ans << '\n';
47  for (int i = 0; i < n; i++)
48      cout << par[i] + 1 << " " << i + 1
         ↪  << '\n';
```

```
1  struct edge{
2      int from, to;
3      int c, f, num;
4      edge(int from, int to, int c, int
       ↪  num):from(from), to(to), c(c),
       ↪  f(0), num(num){}
5      edge(){}
6  };
7
8  const int max_n =  600;
9
10 edge eds[150000];
11 int num = 0;
12 int it[max_n];
13 vector<int> gr[max_n];
14 int s, t;
15 vector<int> d(max_n);
16
17 bool bfs(int k) {
18     queue<int> q;
19     q.push(s);
20     fill(d.begin(), d.end(), -1);
21     d[s] = 0;
22     while (!q.empty()) {
23         int v = q.front();
24         q.pop();
25         for (int x : gr[v])
26             if (d[eds[x].to] == -1 &&
               ↪  eds[x].c - eds[x].f >=
               ↪  (1 << k)){
27                 d[eds[x].to] = d[v] + 1;
28                 q.push(eds[x].to);
29             }
30     }
31
32     return (d[t] != -1);
33 }
34
35 int dfs(int v, int flow, int k) {
36     if (flow < (1 << k))
37         return 0;
38     if (v == t)
39         return flow;
40     for (; it[v] < gr[v].size();
       ↪  it[v]++) {
41         int num = gr[v][it[v]];
42         if (d[v] + 1 != d[num].to)
43             continue;
44         int res = dfs(eds[num].to,
           ↪  min(flow, eds[num].c -
           ↪  eds[num].f), k);
45         if (res){
46             eds[num].f += res;
47             eds[num ^ 1].f -= res;
48             return res;
49         }
50     }
51     return 0;
52 }
53
54 void add(int fr, int to, int c, int nm)
   ↪  {
55     gr[fr].push_back(num);
56     eds[num++] = edge(fr, to, c, nm);
57     gr[to].push_back(num);
58     eds[num++] = edge(to, fr, 0, nm);
       ↪  //corrected c
59 }
60
61 int ans = 0;
62     for (int k = 30; k >= 0; k--)
63         while (bfs(k)) {
64             memset(it, 0, sizeof(it));
65             while (int res = dfs(s, 1e9
               ↪  + 500, k))
66                 ans += res;
67         }
68
69
70 // decomposition
71
72 int path_num = 0;
73 vector<int> paths[550];
74 int flows[550];
75
76 int decomp(int v, int flow) {
77     if (flow < 1)
78         return 0;
79     if (v == t) {
80         path_num++;
81         flows[path_num - 1] = flow;
82         return flow;
83     }
84     for (int i = 0; i < gr[v].size();
       ↪  i++) {
85         int num = gr[v][i];
86         int res = decomp(eds[num].to,
           ↪  min(flow, eds[num].f));
87         if (res)  {
88             eds[num].f -= res;
89             paths[path_num -
               ↪  1].push_back(eds[num].num);
```

```
90                 return res;
91             }
92         }
93     return 0;
94 }
95
96 while (decomp(s, 1e9 + 5));
```

```
1  long long ans = 0;
2  int mx = 2 * n + 2;
3
4  memset(upd, 0, sizeof(upd));
5  for (int i = 0; i < mx; i++)
6      dist[i] = inf;
7  dist[st] = 0;
8  queue<int> q;
9  q.push(st);
10 upd[st] = 1;
11 while (!q.empty()){
12     int v = q.front();
13     q.pop();
14     if (upd[v]){
15         for (int x : gr[v])         {
16             edge &e = edges[x];
17             if (e.c - e.f > 0 && dist[v]
               ↪  != inf && dist[e.to] >
               ↪  dist[v] + e.w) {
18                 dist[e.to] = dist[v] +
                   ↪   e.w;
19                 if (!upd[e.to])
20                     q.push(e.to);
21                 upd[e.to] = true;
22                 p[e.to] = x;
23             }
24         }
25         upd[v] = false;
26     }
27 }
28
29 for (int i = 0; i < k; i++){
30     for (int i = 0; i < mx; i++)
31         d[i] = inf;
32     d[st] = 0;
33     memset(used, false, sizeof(used));
34     set<pair<int, int> > s;
35     s.insert(make_pair(0, st));
36     for (int i = 0; i < mx; i++){
37         int x;
38         while (!s.empty() &&
               ↪  used[(s.begin() ->
               ↪  second)]){
39             s.erase(s.begin());
40         }
41         if (s.empty())
42             break;
43         x = s.begin() -> second;
44         used[x] = true;
45         s.erase(s.begin());
```

```
46        for (int i = 0; i <
     ↪  gr[x].size(); i++){
47        edge &e = edges[gr[x][i]];
48        if (!used[e.to] && e.c - e.f
     ↪  > 0){
49            if (d[e.to] > d[x] +
         ↪  (e.c - e.f) * e.w +
         ↪  dist[x] -
         ↪  dist[e.to]){
50            d[e.to] = d[x] +
             ↪  (e.c - e.f) *
             ↪  e.w + dist[x] -
             ↪  dist[e.to];
51            p[e.to] = gr[x][i];
52
         ↪  s.insert(make_pair(d[e.to],
         ↪  e.to));
53            }
54        }
55      }
56      dist[x] += d[x];
57    }
58    int pos = t;
59    while (pos != st){
60          int id = p[pos];
61          edges[id].f += 1;
62      edges[id ^ 1].f -= 1;
63      pos = edges[id].from;
64    }
65 }
```

```
1 string min_cyclic_shift (string s) {
2        s += s;
3        int n = (int) s.length();
4        int i=0, ans=0;
5        while (i < n/2) {
6              ans = i;
7              int j=i+1, k=i;
8              while (j < n && s[k] <=
           ↪  s[j]) {
9                  if (s[k] < s[j])
10                     k = i;
11                 else
12                     ++k;
13                 ++j;
14             }
15             while (i <= k)  i += j -
           ↪  k;
16         }
17         return s.substr (ans, n/2);
18 }
```

Sum over subsets

```
1 for(int i = 0; i<(1<<N); ++i)
2        F[i] = A[i];
3 for(int i = 0;i < N; ++i) for(int mask =
   ↪  0; mask < (1<<N); ++mask){
4        if(mask & (1<<i))
5                F[mask] +=
               ↪  F[mask^(1<<i)];
6 }
```

```
1  vector<int> getZ(string s){
2      vector<int> z;
3      z.resize(s.size(), 0);
4      int l = 0, r = 0;
5      for (int i = 1; i < s.size(); i++){
6          if (i <= r)
7              z[i] = min(r - i + 1, z[i -
                   ↪ l]);
8          while (i + z[i] < s.size() &&
               ↪ s[z[i]] == s[i + z[i]])
9              z[i]++;
10         if (i + z[i] - 1 > r){
11             r = i + z[i] - 1;
12             l = i;
13         }
14     }
15     return z;
16 }
17
18 vector<int> getP(string s){
19     vector<int> p;
20     p.resize(s.size(), 0);
21     int k = 0;
22     for (int i = 1; i < s.size(); i++){
23         while (k > 0 && s[i] == s[k])
24             k = p[k - 1];
25         if (s[i] == s[k])
26             k++;
27         p[i] = k;
28     }
29     return p;
30 }
31
32 vector<int> getH(string s){
33     vector<int> h;
34     h.resize(s.size() + 1, 0);
35     for (int i = 0; i < s.size(); i++)
36         h[i + 1] = ((h[i] * 1ll * pow) +
               ↪ s[i] - 'a' + 1) % mod;
37     return h;
38 }
39
40 int getHash(vector<int> &h, int l, int
       ↪ r){
41     int res = (h[r + 1] - h[l] * p[r - l
           ↪ + 1]) % mod;
42     if (res < 0)
43         res += mod;
44     return res;
45 }
```

suf array + lcp

```
1  // O(n)
2  #define forn(i, n) for (int i = 0; i <
       ↪ (int)(n); i++)
3
4  typedef vector<char> bits;
5
6  template<const int end>
7  void getBuckets(int *s, int *bkt, int n,
       ↪ int K) {
8      fill(bkt, bkt + K + 1, 0);
9      forn(i, n) bkt[s[i] + !end]++;
10     forn(i, K) bkt[i + 1] += bkt[i];
11 }
12 void induceSAl(bits &t, int *SA, int *s,
       ↪ int *bkt, int n, int K) {
13     getBuckets<0>(s, bkt, n, K);
14     forn(i, n) {
15         int j = SA[i] - 1;
16         if (j >= 0 && !t[j])
17             SA[bkt[s[j]]++] = j;
18     }
19 }
20 void induceSAs(bits &t, int *SA, int *s,
       ↪ int *bkt, int n, int K) {
21     getBuckets<1>(s, bkt, n, K);
22     for (int i = n - 1; i >= 0; i--) {
23         int j = SA[i] - 1;
24         if (j >= 0 && t[j])
25             SA[--bkt[s[j]]] = j;
26     }
27 }
28
29 void SA_IS(int *s, int *SA, int n, int
       ↪ K) { // require last symbol is 0
30 #define isLMS(i) (i && t[i] && !t[i-1])
31     int i, j;
32     bits t(n);
33     t[n-1] = 1;
34     for (i = n - 3; i >= 0; i--)
35         t[i] = (s[i]<s[i+1] ||
               ↪ (s[i]==s[i+1] &&
               ↪ t[i+1]==1));
36     int bkt[K + 1];
37     getBuckets<1>(s, bkt, n, K);
38     fill(SA, SA + n, -1);
39     forn(i, n)
40         if (isLMS(i))
41             SA[--bkt[s[i]]] = i;
42     induceSAl(t, SA, s, bkt, n, K);
```

```
43        induceSAs(t, SA, s, bkt, n, K);
44        int n1 = 0;
45        forn(i, n)
46            if (isLMS(SA[i]))
47                SA[n1++] = SA[i];
48        fill(SA + n1, SA + n, -1);
49        int name = 0, prev = -1;
50        forn(i, n1) {
51            int pos = SA[i];
52            bool diff = false;
53            for (int d = 0; d < n; d++)
54                if (prev == -1 || s[pos+d]
                    != s[prev+d] || t[pos+d]
                    != t[prev+d])
55                    diff = true, d = n;
56                else if (d > 0 &&
                    (isLMS(pos+d) ||
                    isLMS(prev+d)))
57                    d = n;
58            if (diff)
59                name++, prev = pos;
60            SA[n1 + (pos >> 1)] = name - 1;
61        }
62    for (i = n - 1, j = n - 1; i >= n1;
        i--)
63        if (SA[i] >= 0)
64            SA[j--] = SA[i];
65    int *s1 = SA + n - n1;
66    if (name < n1)
67        SA_IS(s1, SA, n1, name - 1);
68    else
69        forn(i, n1)
70            SA[s1[i]] = i;
71    getBuckets<1>(s, bkt, n, K);
72    for (i = 1, j = 0; i < n; i++)
73        if (isLMS(i))
74            s1[j++] = i;
75    forn(i, n1)
76        SA[i] = s1[SA[i]];
77    fill(SA + n1, SA + n, -1);
78    for (i = n1 - 1; i >= 0; i--) {
79        j = SA[i], SA[i] = -1;
80        SA[--bkt[s[j]]] = j;
81    }
82    induceSAl(t, SA, s, bkt, n, K);
83    induceSAs(t, SA, s, bkt, n, K);
84 }
85 // O(n) ended
86
87 // O(n logn)
88 string str;
```

```
89 int N, m, SA [MAX_N], LCP [MAX_N];
90 int x [MAX_N], y [MAX_N], w [MAX_N], c
    [MAX_N];
91
92 inline bool cmp (const int a, const int
    b, const int l) { return (y [a] == y
    [b] && y [a + l] == y [b + l]); }
93
94 void Sort () {
95    for (int i = 0; i < m; ++i) w[i] =
        0;
96    for (int i = 0; i < N; ++i)
        ++w[x[y[i]]];
97    for (int i = 0; i < m - 1; ++i) w[i
        + 1] += w[i];
98    for (int i = N - 1; i >= 0; --i)
        SA[--w[x[y[i]]]] = y[i];
99 }
100
101 void DA () {
102    for (int i = 0; i < N; ++i) x[i] =
        str[i], y[i] = i;
103    Sort ();
104    for (int i, j = 1, p = 1; p < N; j
        <<= 1, m = p) {
105        for (p = 0, i = N - j; i < N;
            i++) y[p++] = i;
106        for (int k = 0; k < N; ++k) if
            (SA[k] >= j) y[p++] = SA[k]
            - j;
107        Sort();
108        for (swap (x, y), p = 1,
            x[SA[0]] = 0, i = 1; i < N;
            ++i) x[SA [i]] = cmp (SA[i -
            1], SA[i], j) ? p - 1 : p++;
109    }
110 }
111
112 void kasaiLCP () {
113    for (int i = 0; i < N; i++) c[SA[i]]
        = i;
114    for (int i = 0, j, k = 0; i < N; LCP
        [c[i++]] = k)
115        if (c [i] > 0) for (k ? k-- : 0,
            j = SA[c[i] - 1]; str[i + k]
            == str[j + k]; k++);
116        else k = 0;
117 }
118
119 void suffixArray () { // require last
    symbol is char(0)
```

```
120      m = 256;
121      N = str.size();
122      DA ();
123      kasaiLCP ();
124  }
125  // O(n logn) ended
```

```
1   int num = 0;
2   long long phi = n, nn = n;
3   for (long long x:primes){
4       if (x*x>nn)
5           break;
6       if (nn % x == 0){
7           while (nn % x == 0)
8               nn /= x;
9           phi -= phi/x;
10          num++;
11      }
12  }
13  if (nn != 1){
14      phi -= phi/nn;
15      num++;
16  }
17  if (!((num == 1 && n % 2 != 0) || n == 4
    ↪  || n == 2 || (num == 2 && n % 2 == 0
    ↪  && n % 4 != 0))){
18      cout << "-1\n";
19      continue;
20  }
21  vector<long long> v;
22  long long pp = phi;
23  for (long long x:primes){
24      if (x*x>pp)
25          break;
26      if (pp % x == 0){
27          while (pp % x == 0)
28              pp /= x;
29          v.push_back(x);
30      }
31  }
32  if (pp != 1){
33      v.push_back(pp);
34  }
35  while (true){
36      long long a = primes[rand()%5000]%n;
37      if (gcd(a, n) != 1)
38          continue;
39      bool bb = false;
40      for (long long x:v)
41          if (pow(a, phi/x) == 1){
42              bb = true;
43              break;
44          }
45      if (!bb){
46          cout << a << "\n";
47          break;
48      }
49  }
```

```
1   const int LOG = 19;
2   const int N = (1 << LOG);
3
4   typedef std::complex<double> cd;
5
6   int rev[N];
7   cd W[N];
8
9   void precalc() {
10      const double pi = std::acos(-1);
11      for (int i = 0; i != N; ++i)
12          W[i] = cd(std::cos(2 * pi * i /
            ↪   N), std::sin(2 * pi * i /
            ↪   N));
13
14      int last = 0;
15      for (int i = 1; i != N; ++i) {
16          if (i == (2 << last))
17              ++last;
18
19          rev[i] = rev[i ^ (1 << last)] |
            ↪   (1 << (LOG - 1 - last));
20      }
21  }
22
23  void fft(vector<cd>& a) {
24      for (int i = 0; i != N; ++i)
25          if (i < rev[i])
26              std::swap(a[i], a[rev[i]]);
27
28      for (int lvl = 0; lvl != LOG; ++lvl)
29          for (int start = 0; start != N;
            ↪   start += (2 << lvl))
30              for (int pos = 0; pos != (1
                ↪   << lvl); ++pos) {
31                  cd x = a[start + pos];
32                  cd y = a[start + pos +
                    ↪   (1 << lvl)];
33
34                  y *= W[pos << (LOG - 1 -
                    ↪   lvl)];
35
36                  a[start + pos] = x + y;
37                  a[start + pos + (1 <<
                    ↪   lvl)] = x - y;
38              }
39  }
40
41  void inv_fft(vector<cd>& a) {
42      fft(a);
```

```
43      std::reverse(a.begin() + 1,
        ↪   a.end());
44
45      for (cd& elem: a)
46          elem /= N;
47  }
```

```
1   void fast_fourier(vector<int>& a) { //
    ↪   AND-FFT.
2       for (int k = 1; k < SZ(a); k *= 2)
3           for (int start = 0; start < (1
            ↪   << K); start += 2 * k) {
4               for (int off = 0; off < k;
                ↪   ++off) {
5                   int a_val = a[start +
                    ↪   off];
6                   int b_val = a[start + k
                    ↪   + off];
7
8                   a[start + off] = b_val;
9                   a[start + k + off] =
                    ↪   add(a_val, b_val);
10              }
11          }
12  }
13
14  void inverse_fast_fourier(vector<int>&
    ↪   a) {
15      for (int k = 1; k < SZ(a); k *= 2)
16          for (int start = 0; start < (1
            ↪   << K); start += 2 * k) {
17              for (int off = 0; off < k;
                ↪   ++off) {
18                  int a_val = a[start +
                    ↪   off];
19                  int b_val = a[start + k
                    ↪   + off];
20
21                  a[start + off] =
                    ↪   sub(b_val, a_val);
22                  a[start + k + off] =
                    ↪   a_val;
23              }
24          }
25  }
```

```cpp
struct Edge {
    int a;
    int b;
    int cost;
};

vector<int> negative_cycle(int n,
    vector<Edge> &edges) {
    // O(nm), return ids of edges in
    //    negative cycle

    vector<int> d(n);
    vector<int> p(n, -1); // last edge
    //    ids

    const int inf = 1e9;

    int x = -1;
    for (int i = 0; i < n; i++) {
        x = -1;
        for (int j = 0; j <
            edges.size(); j++) {
            Edge &e = edges[j];

            if (d[e.b] > d[e.a] +
                e.cost) {
                d[e.b] = max(-inf,
                    d[e.a] + e.cost);
                p[e.b] = j;
                x = e.b;
            }
        }
    }

    if (x == -1)
        return vector<int>(); // no
        //    negative cycle

    for (int i = 0; i < n; i++)
        x = edges[p[x]].a;

    vector<int> result;
    for (int cur = x; ; cur =
        edges[p[cur]].a) {
        if (cur == x && result.size() >
            0) break;
        result.push_back(p[cur]);
    }
    reverse(result.begin(),
        result.end());

    return result;
}

vector<int> min_avg_cycle(int n,
    vector<Edge> &edges) {
    const int inf = 1e3;

    for (auto &e : edges)
        e.cost *= n * n;

    int l = -inf;
    int r = inf;
    while (l + 1 < r) {
        int m = (l + r) / 2;
        for (auto &e : edges)
            e.cost -= m;

        if (negative_cycle(n,
            edges).empty())
            l = m;
        else
            r = m;

        for (auto &e : edges)
            e.cost += m;
    }

    if (r >= 0) // if only negative
    //    needed
        return vector<int>();

    for (auto &e : edges)
        e.cost -= r;

    vector<int> result =
        negative_cycle(n, edges);

    for (auto &e : edges)
        e.cost += r;

    for (auto &e : edges)
        e.cost /= n * n;

    return result;
}

struct edge {
    int from, to;
    int c, f, cost;
};
```

```cpp
89
90  const int max_n = 200;
91
92  vector<int> gr[max_n];
93  vector<edge> edges;
94
95  void add(int fr, int to, int c, int
    ↪  cost) {
96      gr[fr].push_back(edges.size());
97      edges.push_back({fr, to, c, 0,
        ↪  cost});
98      gr[to].push_back(edges.size());
99      edges.push_back({to, fr, 0, 0,
        ↪  -cost}); // single
100 }
101
102 void calc_min_circulation(int n) {
103     while (true) {
104         vector<Edge> eds;
105         vector<int> origin;
106
107         for (int i = 0; i <
            ↪  edges.size(); i++) {
108             edge &e = edges[i];
109             if (e.c - e.f > 0) {
110                 eds.push_back({e.from,
                    ↪  e.to, e.cost});
111                 origin.push_back(i);
112             }
113         }
114
115         vector<int> cycle =
            ↪  negative_cycle(n, eds);
116
117         if (cycle.empty())
118             break;
119
120         for (auto id : cycle) {
121             int x = origin[id];
122             edges[x].f += 1;
123             edges[x ^ 1].f -= 1;
124         }
125     }
126 }
```

Добавим к нашему графу вершину s и рёбра из неё во все остальные вершины. Запустим алгоритм Форда-Беллмана и попросим его построить нам квадратную матрицу со следующим условием: d[i][u] - длина минимального пути от s до u ровно из i ребер. Тогда длина оптимального цикла μ* минимального среднего веса вычисляется как minumaxkd[n][u]−d[k][u]n−k.

Достаточно будет доказать это правило для μ*=0, так как для других μ* можно просто отнять эту величину от всех ребер и получить снова случай с μ*=0.

Чтобы найти цикл после построения матрицы d[k][u], запомним, при каких u и k достигается оптимальное значение μ*, и, используя d[n][u], поднимемся по указателям предков. Как только мы попадем в уже посещенную вершину - мы нашли цикл минимального среднего веса.

Этот алгоритм работает за O(VE)

```
func findMinCycle(Graph G)
    // вводим мнимую вершину s, от
    ↪  которой проведём рёбра нулевого
    ↪  веса в каждую вершину графа
    Node s
    Edge[] e
    insert(s)
    i = 0
    for u in G
        e[i].begin = s
        e[i].end = u
        e[i].weight = 0
        i++
    // строим матрицу кратчайших
    ↪  расстояний, запустив алгоритм
    ↪  Форда-Беллмана из вершины s
    fordBellman(s)
    // m - длина оптимального цикла
```

```
23    m = minumaxk((d[n][u] - d[k][u]) /
      ↪  (n - k))
```

DM

Кол-во корневых деревьев:

$t(G) = \frac{1}{n}\lambda_2 \ldots \lambda_n \ (\lambda_1 = 0)$

Кол-во эйлеровых циклов:

$e(D) = t^-(D, x) \cdot \prod_{y \in D} (outdeg(y) - 1)!$

Наличие совершенного паросочетания:

T – матрица с нулями на диагонали. Если есть ребро (i, j), то $a_{i,j} := x_{i,j}$, $a_{j,i} = -x_{i,j}$

$\det(T) = 0 \Leftrightarrow$ нет совершенного паросочетания.

Whitespace code FFT