Team Reference

pragma

```
#pragma GCC optimize("O3,no-stack-protector")
#pragma GCC target("sse,sse2,sse4,ssse3,popcnt,abm,mmx,a
```

Алгебра Pick

В + Γ / 2 − 1 = AREA,

где В — количество целочисленных точек внутри многоугольника, а Γ — 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: $7340033 = 7 \cdot 2^{20} + 1; w : 5(w^{2^{20}} = 1)$

```
1  // pollard begins
2
3  const int max_step = 4e5;
4
5  unsigned long long gcd(unsigned long long a,
   ↪  unsigned long long b){
6    if (!a) return 1;
7    while (a) swap(a, b%=a);
8    return b;
9  }
10
11  unsigned long long get(unsigned long long a,
    ↪  unsigned long long b){
12    if (a > b)
13      return a-b;
14    else
15      return b-a;
16  }
17
18  unsigned long long pollard(unsigned long long n){
19    unsigned long long x = (rand() + 1) % n, y = 1,
      ↪  g;
20    int stage = 2, i = 0;
21    g = gcd(get(x, y), n);
22    while (g == 1) {
23      if (i == max_step)
24        break;
25      if (i == stage) {
26        y = x;
27        stage <<= 1;
28      }
29      x = (x * (__int128)x + 1) % n;
30      i++;
31      g = gcd(get(x, y), n);
32    }
33    return g;
34  }
35
36  // pollard ends
```

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

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

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

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

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

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

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

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

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

- prime: $998244353 = 7 \cdot 17 \cdot 2^{23} + 1; w : 3^{7*17}$.

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
// sk fast allocation begins
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!

// sk fast allocation ends



// sk fast read-write begins

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;
}
```

```
59    }
60
61    template <class T>
62    inline T readInt() {
63      int s = 1, c = readChar();
64      T x = 0;
65      if (c == '-')
66        s = -1, c = getChar();
67      while ('0' <= c && c <= '9')
68        x = x * 10 + c - '0', c = getChar();
69      return s == 1 ? x : -x;
70    }
71
72    /** Write */
73
74    static int write_pos = 0;
75    static char write_buf[buf_size];
76
77    inline void writeChar( int x ) {
78      if (write_pos == buf_size)
79        fwrite(write_buf, 1, buf_size, stdout),
          ↪  write_pos = 0;
80      write_buf[write_pos++] = x;
81    }
82
83    template <class T>
84    inline void writeInt( T x, char end ) {
85      if (x < 0)
86        writeChar('-'), x = -x;
87
88      char s[24];
89      int n = 0;
90      while (x || !n)
91        s[n++] = '0' + x % 10, x /= 10;
92      while (n--)
93        writeChar(s[n]);
94      if (end)
95        writeChar(end);
96    }
97
98    inline void writeWord( const char *s ) {
99      while (*s)
100       writeChar(*s++);
101   }
102
103   struct Flusher {
104     ~Flusher() {
105       if (write_pos)
106         fwrite(write_buf, 1, write_pos, stdout),
            ↪  write_pos = 0;
107     }
108   } flusher;
109
110
111   // sk fast read-write ends

1     // extended euclid begins
2
3     int gcd (int a, int b, int & x, int & y) {
4       if (a == 0) {
5         x = 0; y = 1;
6         return b;
```

```
7       }
8       int x1, y1;
9       int d = gcd (b%a, a, x1, y1);
10      x = y1 - (b / a) * x1;
11      y = x1;
12      return d;
13    }
14
15    // extended euclid ends

1     // FFT begins
2
3     const int LOG = 19;
4     const int N = (1 << LOG);
5
6     typedef std::complex<double> cd;
7
8     int rev[N];
9     cd W[N];
10
11    void precalc() {
12      const double pi = std::acos(-1);
13      for (int i = 0; i != N; ++i)
14        W[i] = cd(std::cos(2 * pi * i / N),
          ↪  std::sin(2 * pi * i / N));
15
16      int last = 0;
17      for (int i = 1; i != N; ++i) {
18        if (i == (2 << last))
19          ++last;
20
21        rev[i] = rev[i ^ (1 << last)] | (1 << (LOG -
          ↪  1 - last));
22      }
23    }
24
25    void fft(vector<cd>& a) {
26      for (int i = 0; i != N; ++i)
27        if (i < rev[i])
28          std::swap(a[i], a[rev[i]]);
29
30      for (int lvl = 0; lvl != LOG; ++lvl)
31        for (int start = 0; start != N; start += (2
          ↪  << lvl))
32          for (int pos = 0; pos != (1 << lvl); ++pos)
            ↪  {
33            cd x = a[start + pos];
34            cd y = a[start + pos + (1 << lvl)];
35
36            y *= W[pos << (LOG - 1 - lvl)];
37
38            a[start + pos] = x + y;
39            a[start + pos + (1 << lvl)] = x - y;
40          }
41    }
42
43    void inv_fft(vector<cd>& a) {
44      fft(a);
45      std::reverse(a.begin() + 1, a.end());
46
47      for (cd& elem: a)
48        elem /= N;
```

```cpp
49    }
50
51    // FFT ends
1     // fast gauss begins
2
3     using elem_t = int;
4     // a[i][rows[i][j].first]=rows[i][j].second;
      //   b[i]=a[i][n]
5     bool gauss(vector<vector<pair<int, elem_t>>>
      →   rows, vector<elem_t> &res) {
6       int n = rows.size();
7
8       res.resize(n + 1, 0);
9       vector<int> p(n + 1);
10      iota(p.begin(), p.end(), 0);
11      vector<int> toZero(n + 1, -1);
12      vector<int> zro(n + 1);
13      vector<elem_t> a(n + 1);
14
15      // optional: sort rows
16
17      sort(p.begin(), p.begin() + n, [&rows](int i,
      →   int j) { return rows[i].size() <
      →   rows[j].size(); });
18      vector<int> invP(n + 1);
19      vector<vector<pair<int, elem_t>>> rs(n);
20      for (int i = 0; i < n; i++) {
21        invP[p[i]] = i;
22        rs[i] = rows[p[i]];
23      }
24      for (int i = 0; i < n; i++) {
25        rows[i] = rs[i];
26        for (auto& el: rows[i]) {
27          if (el.first < n) {
28            el.first = invP[el.first];
29          }
30        }
31      }
32
33
34      for (int i = 0; i < n; i++) {
35        for (auto& el: rows[i]) {
36          a[el.first] = el.second;
37        }
38        while (true) {
39          int k = -1;
40          for (auto& el: rows[i]) {
41            if (!isZero(a[el.first]) &&
      →       toZero[el.first] != -1 &&
42              (k == -1 || toZero[el.first] <
      →         toZero[k])) {
43              k = el.first;
44            }
45          }
46          if (k == -1)
47            break;
48
49          int j = toZero[k];
50          elem_t c = a[k];
51
52          for (auto el: rows[j]) {
```

```cpp
53            if (isZero(a[el.first]))
54              rows[i].emplace_back(el.first, 0);
55            a[el.first] = sub(a[el.first], mult(c,
      →       el.second));
56          }
57
58          auto cond = [&a](const pair<int, elem_t>&
      →     p) { return isZero(a[p.first]); };
59
60
      →       rows[i].erase(std::remove_if(rows[i].begin(),
      →       rows[i].end(), cond), rows[i].end());
61        }
62
63        bool ok = false;
64        for (auto& el: rows[i]) {
65          if (el.first < n && !isZero(a[el.first])) {
66            toZero[el.first] = i;
67            zro[i] = el.first;
68    //        det = (det * a[el.first]) % MOD;
69
70            elem_t c = divM(1, a[el.first]);
71            for (auto& el: rows[i]) {
72              el.second = mult(a[el.first], c);
73              a[el.first] = 0;
74            }
75
76            ok = true;
77            break;
78          }
79        }
80
81        if (!ok) {
82    //        det = 0;
83          return false;
84        }
85      }
86
87      res[n] = sub(0, 1);
88      for (int i = n - 1; i >= 0; i--) {
89        int k = zro[i];
90        for (auto& el : rows[i])
91          if (el.first != k)
92            res[p[k]] = sub(res[p[k]],
      →       mult(el.second, res[p[el.first]]));
93      }
94
95      return true;
96    }
97
98    // fast gauss ends
1     // simple geometry begins
2
3     struct Point {
4       double x, y;
5       Point operator+(const Point& p) const { return
      →   {x + p.x, y + p.y}; }
6       Point operator-(const Point& p) const { return
      →   {x - p.x, y - p.y}; }
7       Point operator*(const double d) const { return
      →   {x * d, y * d}; }
```

```cpp
 8     Point rotate() const { return {y, -x}; }
 9     double operator*(const Point& p) const { return
       ↪  x * p.x + y * p.y; }
10     double operator^(const Point& p) const { return
       ↪  x * p.y - y * p.x; }
11     double dist() const { return sqrt(x * x + y *
       ↪  y); }
12   };
13
14   struct Line {
15     double a, b, c;
16     Line(const Point& p1, const Point& p2) {
17       a = p1.y - p2.y;
18       b = p2.x - p1.x;
19       c = - a * p1.x - b * p1.y;
20
21       double d = sqrt(sqr(a) + sqr(b));
22       a /= d, b /= d, c /= d;
23     }
24     bool operator||(const Line& l) const { return
       ↪  fabs(a * l.b - l.a * b) < EPS; }
25     double dist(const Point& p) const { return
       ↪  fabs(a * p.x + b * p.y + c); }
26     Point operator^(const Line& l) const {
27       return {(l.c * b - c * l.b) / (a * l.b - l.a
       ↪  * b),
28          (l.c * a - c * l.a) / (l.a * b - a *
       ↪  l.b)};
29     }
30     Point projection(const Point& p) const {
31       return p - Point{a, b} * (a * p.x + b * p.y +
       ↪  c);
32     }
33   };
34
35   struct Circle {
36     Point c;
37     double r;
38     Circle(const Point& c, double r) : c(c), r(r)
       ↪  {}
39     Circle(const Point& a, const Point& b, const
       ↪  Point& c) {
40       Point p1 = (a + b) * 0.5, p2 = (a + c) * 0.5;
41       Point q1 = p1 + (b - a).rotate(), q2 = p2 +
       ↪  (c - a).rotate();
42       this->c = Line(p1, q1) ^ Line(p2, q2);
43       r = (a - this->c).dist();
44     }
45   };
46
47   inline bool on_segment(const Point& p1, const
     ↪  Point& p2, const Point& x, bool strictly) {
48     if (fabs((p1 - x) ^ (p2 - x)) > EPS)
49       return false;
50     return (p1 - x) * (p2 - x) < (strictly ? - EPS
       ↪  : EPS);
51   }
52
53   // in case intersection is not a segment
54   inline bool intersect_segments(const Point& p1,
     ↪  const Point& p2, const Point& q1, const
     ↪  Point& q2, Point& x) {
55     Line l1(p1, p2), l2(q1, q2);
56     if (l1 || l2) return false;
57     x = l1 ^ l2;
58     return on_segment(p1, p2, x, false);
59   }
60
61   // in case circles are not equal
62   inline bool intersect_circles(const Circle& c1,
     ↪  const Circle& c2, Point& p1, Point& p2) {
63     double d = (c2.c - c1.c).dist();
64     if (d > c1.r + c2.r + EPS || d < fabs(c1.r -
       ↪  c2.r) - EPS)
65       return false;
66     double cosa = (sqr(d) + sqr(c1.r) - sqr(c2.r))
       ↪  / (2 * c1.r * d);
67     double l = c1.r * cosa, h = sqrt(sqr(c1.r) -
       ↪  sqr(l));
68     Point v = (c2.c - c1.c) * (1 / d), p = c1.c + v
       ↪  * l;
69     p1 = p + v.rotate() * h, p2 = p - v.rotate() *
       ↪  h;
70     return true;
71   }
72
73   inline bool intersect_circle_and_line(const
     ↪  Circle& c, const Line& l, Point& p1, Point&
     ↪  p2) {
74     double d = l.dist(c.c);
75     if (d > c.r + EPS)
76       return false;
77     Point p = l.projection(c.c);
78     Point n{l.b, -l.a};
79     double h = sqrt(sqr(c.r) - sqr(l.dist(c.c)));
80     p1 = p + n * h, p2 = p - n * h;
81     return true;
82   }
83
84   // simple geometry ends
85
 1   // convex hull begins
 2
 3   struct Point {
 4     int x, y;
 5     Point operator-(const Point& p) const { return
       ↪  {x - p.x, y - p.y}; }
 6     int64_t operator^(const Point& p) const {
       ↪  return x * 1ll * p.y - y * 1ll * p.x; }
 7     int64_t dist() const { return x * 1ll * x + y *
       ↪  1ll * y; }
 8     bool operator<(const Point& p) const { return x
       ↪  != p.x ? x < p.x : y < p.y; }
 9   };
10
11   // all point on convex hull are included
12   vector<Point> convex_hull(vector<Point> pt) {
13     int n = pt.size();
14     Point p0 = *std::min_element(pt.begin(),
       ↪  pt.end());
```

```cpp
15    std::sort(pt.begin(), pt.end(), [&p0](const
   ↪    Point& a, const Point& b) {
16      int64_t cp = (a - p0) ^ (b - p0);
17      return cp != 0 ? cp > 0 : (a - p0).dist() <
   ↪      (b - p0).dist();
18    });
19
20    int i = n - 1;
21    for (; i > 0 && ((pt[i] - p0) ^ (pt[i - 1] -
   ↪    p0)) == 0; i--);
22    std::reverse(pt.begin() + i, pt.end());
23
24    vector<Point> ch;
25    for (auto& p : pt) {
26      while (ch.size() > 1) {
27        auto& p1 = ch[(int) ch.size() - 1];
28        auto& p2 = ch[(int) ch.size() - 2];
29        int64_t cp = (p1 - p2) ^ (p - p1);
30        if (cp >= 0) break;
31        ch.pop_back();
32      }
33      ch.push_back(p);
34    }
35
36    return ch;
37  }
38
39  // convex hull ends

1   // convex hull trick begins
2
3   typedef long long ftype;
4   typedef complex<ftype> point;
5   #define x real
6   #define y imag
7
8   ftype dot(point const& a, point const& b) {
9     return (conj(a) * b).x();
10  }
11
12  ftype f(point const& a, int x) {
13    return dot(a, {compressed[x], 1});
14    //return dot(a, {x, 1});
15  }
16
17  int pos = 0;
18
19  // (x, y) -> (k, b) -> kb + x
20  struct li_chao { // for min
21    vector<point> line;
22
23    li_chao(int maxn) {
24      line.resize(4 * maxn, {0, inf});
25    }
26
27    void add_line(int v, int l, int r, int a, int
   ↪    b, point nw) {
28      if (r <= a || b <= l) return; // remove if no
   ↪      [a, b) query
29
30      int m = (l + r) >> 1;
31
32      if (!(a <= l && r <= b)) { // remove if no
   ↪      [a, b) query
33        add_line(v + v + 1, l, m, a, b, nw);
34        add_line(v + v + 2, m, r, a, b, nw);
35        return;
36      }
37
38      bool lef = f(nw, l) < f(line[v], l);
39      bool mid = f(nw, m) < f(line[v], m);
40
41      if (mid) swap(line[v], nw);
42
43      if (l == r - 1)
44        return;
45
46      if (lef != mid)
47        add_line(v + v + 1, l, m, a, b, nw);
48      else
49        add_line(v + v + 2, m, r, a, b, nw);
50    }
51
52    ftype get(int v, int l, int r, int x) {
53      if(l == r - 1)
54        return f(line[v], x);
55      int m = (l + r) / 2;
56      if(x < m) {
57        return min(f(line[v], x), get(v + v + 1, l,
   ↪        m, x));
58      } else {
59        return min(f(line[v], x), get(v + v + 2, m,
   ↪        r, x));
60      }
61    }
62
63  } cdt(maxn);
64
65  // convex hull with stack
66
67  ftype cross(point a, point b) {
68    return (conj(a) * b).y();
69  }
70
71  vector<point> hull, vecs;
72
73  void add_line(ftype k, ftype b) {
74    point nw = {k, b};
75    while(!vecs.empty() && dot(vecs.back(), nw -
   ↪    hull.back()) < 0) {
76      hull.pop_back();
77      vecs.pop_back();
78    }
79    if(!hull.empty()) {
80      vecs.push_back(1i * (nw - hull.back()));
81    }
82    hull.push_back(nw);
83  }
84
85  int get(ftype x) {
86    point query = {x, 1};
87    auto it = lower_bound(vecs.begin(), vecs.end(),
   ↪    query, [](point a, point b) {
```

```cpp
88      return cross(a, b) > 0;
89    });
90    return dot(query, hull[it - vecs.begin()]);
91  }
92
93  // convex hull trick ends

1   // heavy-light begins
2
3   int sz[maxn];
4
5   void dfs_sz(int v, int par = -1) {
6     sz[v] = 1;
7     for (int x : gr[v])
8       if (x != par) {
9         dfs_sz(x, v);
10        sz[v] += sz[x];
11      }
12    for (int i = 0; i < gr[v].size(); i++)
13      if (gr[v][i] != par)
14      if (sz[gr[v][i]] * 2 >= sz[v]) {
15        swap(gr[v][i], gr[v][0]);
16        break;
17      }
18  }
19
20  int rev[maxn];
21  int t_in[maxn];
22  int upper[maxn];;
23  int par[maxn];
24  int dep[maxn];
25
26  int T = 0;
27
28  void dfs_build(int v, int uppr, int pr = -1) {
29    rev[T] = v;
30    t_in[v] = T++;
31    dep[v] = pr == -1 ? 0 : dep[pr] + 1;
32    par[v] = pr;
33    upper[v] = uppr;
34
35    bool first = true;
36
37    for (int x : gr[v])
38      if (x != pr) {
39        dfs_build(x, first ? upper[v] : x, v);
40        first = false;
41      }
42  }
43
44  struct interval {
45    int l;
46    int r;
47    bool inv; // should direction be reversed
48  };
49
50  // node-weighted hld
51  vector<interval> get_path(int a, int b) {
52    vector<interval> front;
53    vector<interval> back;
54
55    while (upper[a] != upper[b]) {
```

```cpp
56      if (dep[upper[a]] > dep[upper[b]]) {
57        front.push_back({t_in[upper[a]], t_in[a],
          ↪   true});
58        a = par[upper[a]];
59      } else {
60        back.push_back({t_in[upper[b]], t_in[b],
          ↪   false});
61        b = par[upper[b]];
62      }
63    }
64
65    front.push_back({min(t_in[a], t_in[b]),
      ↪   max(t_in[a], t_in[b]), t_in[a] > t_in[b]});
66    // for edge-weighted hld add:
      ↪   "front.back().l++;"
67    front.insert(front.end(), back.rbegin(),
      ↪   back.rend());
68
69    return front;
70  }
71
72  // heavy-light ends

1   // max flow begins
2
3   struct edge{
4     int from, to;
5     int c, f, num;
6     edge(int from, int to, int c, int
        ↪   num):from(from), to(to), c(c), f(0),
        ↪   num(num){}
7     edge(){}
8   };
9
10  const int max_n =  600;
11
12  edge eds[150000];
13  int num = 0;
14  int it[max_n];
15  vector<int> gr[max_n];
16  int s, t;
17  vector<int> d(max_n);
18
19  bool bfs(int k) {
20    queue<int> q;
21    q.push(s);
22    fill(d.begin(), d.end(), -1);
23    d[s] = 0;
24    while (!q.empty()) {
25      int v = q.front();
26      q.pop();
27      for (int x : gr[v]) {
28        int to = eds[x].to;
29        if (d[to] == -1 && eds[x].c - eds[x].f >=
          ↪   (1 << k)){
30          d[to] = d[v] + 1;
31          q.push(to);
32        }
33      }
34    }
35
36    return (d[t] != -1);
```

```
37    }
38
39    int dfs(int v, int flow, int k) {
40      if (flow < (1 << k))
41        return 0;
42      if (v == t)
43        return flow;
44      for (; it[v] < gr[v].size(); it[v]++) {
45        int num = gr[v][it[v]];
46        if (d[v] + 1 != d[num].to)
47          continue;
48        int res = dfs(eds[num].to, min(flow,
         ↪ eds[num].c - eds[num].f), k);
49        if (res){
50          eds[num].f += res;
51          eds[num ^ 1].f -= res;
52          return res;
53        }
54      }
55      return 0;
56    }
57
58    void add(int fr, int to, int c, int nm) {
59      gr[fr].push_back(num);
60      eds[num++] = edge(fr, to, c, nm);
61      gr[to].push_back(num);
62      eds[num++] = edge(to, fr, 0, nm); //corrected to c
63    }
64
65    int ans = 0;
66      for (int k = 30; k >= 0; k--)
67        while (bfs(k)) {
68          memset(it, 0, sizeof(it));
69          while (int res = dfs(s, 1e9 + 500, k))
70            ans += res;
71        }
72
73
74    // decomposition
75
76    int path_num = 0;
77    vector<int> paths[max_n];
78    int flows[max_n];
79
80    int decomp(int v, int flow) {
81      if (flow < 1)
82        return 0;
83      if (v == t) {
84        path_num++;
85        flows[path_num - 1] = flow;
86        return flow;
87      }
88      for (int i = 0; i < gr[v].size(); i++) {
89        int num = gr[v][i];
90        int res = decomp(eds[num].to, min(flow,
         ↪ eds[num].f));
91        if (res)  {
92          eds[num].f -= res;
93          paths[path_num -
           ↪ 1].push_back(eds[num].num);
94          return res;
```

```
95          }
96        }
97        return 0;
98    }
99
100   while (decomp(s, 1e9 + 5));
101
102   // max flow ends


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

```cpp
51        if (d[e.to] > d[x] + (e.c - e.f) * e.w +
      ↪ dist[x] - dist[e.to]){
52          d[e.to] = d[x] + (e.c - e.f) * e.w +
        ↪ dist[x] - dist[e.to];
53          p[e.to] = gr[x][i];
54          s.insert(make_pair(d[e.to], e.to));
55        }
56      }
57    }
58    dist[x] += d[x];
59  }
60  int pos = t;
61  while (pos != st){
62    int id = p[pos];
63    edges[id].f += 1;
64    edges[id ^ 1].f -= 1;
65    pos = edges[id].from;
66  }
67 }
68
69 // min-cost flow ends

1 // bad hungarian begins
2
3 fill(par, par + 301, -1);
4 fill(par2, par2 + 301, -1);
5
6 int ans = 0;
7 for (int v = 0; v < n; v++){
8   memset(useda, false, sizeof(useda));
9   memset(usedb, false, sizeof(usedb));
10  useda[v] = true;
11  for (int i = 0; i < n; i++)
12    w[i] = make_pair(a[v][i] + row[v] + col[i],
      ↪ v);
13  memset(prev, 0, sizeof(prev));
14  int pos;
15  while (true){
16    pair<pair<int, int>, int> p =
      ↪ make_pair(make_pair(1e9, 1e9), 1e9);
17    for (int i = 0; i < n; i++)
18      if (!usedb[i])
19        p = min(p, make_pair(w[i], i));
20    for (int i = 0; i < n; i++)
21      if (!useda[i])
22        row[i] += p.first.first;
23    for (int i = 0; i < n; i++)
24      if (!usedb[i]){
25        col[i] -= p.first.first;
26        w[i].first -= p.first.first;
27      }
28    ans += p.first.first;
29    usedb[p.second] = true;
30    prev[p.second] = p.first.second; //из второй
      ↪ в первую
31    int x = par[p.second];
32    if (x == -1){
33      pos = p.second;
34      break;
35    }
36    useda[x] = true;
37    for (int j = 0; j < n; j++)
38      w[j] = min(w[j], {a[x][j] + row[x] +
        ↪ col[j], x});
39    }
40  }
41  while (pos != -1){
42    int nxt = par2[prev[pos]];
43    par[pos] = prev[pos];
44    par2[prev[pos]] = pos;
45    pos = nxt;
46  }
47 }
48 cout << ans << "\n";
49 for (int i = 0; i < n; i++)
50   cout << par[i] + 1 << " " << i + 1 << "\n";
51
52 // bad hungarian ends

1 // Edmonds O(n^3) begins
2
3 vector<int> gr[MAXN];
4 int match[MAXN], p[MAXN], base[MAXN], q[MAXN];
5 bool used[MAXN], blossom[MAXN];
6 int mark[MAXN];
7 int C = 1;
8
9 int lca(int a, int b) {
10   C++;
11   for (;;) {
12     a = base[a];
13     mark[a] = C;
14     if (match[a] == -1) break;
15     a = p[match[a]];
16   }
17
18   for (;;) {
19     b = base[b];
20     if (mark[b] == C) return b;
21     b = p[match[b]];
22   }
23 }
24
25 void mark_path(int v, int b, int children) {
26   while (base[v] != b) {
27     blossom[base[v]] = blossom[base[match[v]]] =
        ↪ true;
28     p[v] = children;
29     children = match[v];
30     v = p[match[v]];
31   }
32 }
33
34 int find_path(int root) {
35   memset(used, 0, sizeof(used));
36   memset(p, -1, sizeof p);
37   for (int i = 0; i < N; i++)
38     base[i] = i;
39
40   used[root] = true;
41   int qh = 0, qt = 0;
42   q[qt++] = root;
43   while (qh < qt) {
44     int v = q[qh++];
```

```
45      for (int to : gr[v]) {
46        if (base[v] == base[to] || match[v] == to)
          ↪  continue;
47        if (to == root || match[to] != -1 &&
          ↪  p[match[to]] != -1) {
48          int curbase = lca(v, to);
49          memset(blossom, 0, sizeof(blossom));
50          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    }

86  // Edmonds O(n^3) ends

1   // string basis begins
2
3   vector<int> getZ(string s){
4     vector<int> z;
5     z.resize(s.size(), 0);
6     int l = 0, r = 0;
7     for (int i = 1; i < s.size(); i++){
8       if (i <= r)
9         z[i] = min(r - i + 1, z[i - l]);
10      while (i + z[i] < s.size() && s[z[i]] == s[i
          ↪  + z[i]])
11        z[i]++;
12      if (i + z[i] - 1 > r){
13        r = i + z[i] - 1;
14        l = i;
15      }
16    }
```

```
17    return z;
18  }

19
20  vector<int> getP(string s){
21    vector<int> p;
22    p.resize(s.size(), 0);
23    int k = 0;
24    for (int i = 1; i < s.size(); i++){
25      while (k > 0 && s[i] == s[k])
26        k = p[k - 1];
27      if (s[i] == s[k])
28        k++;
29      p[i] = k;
30    }
31    return p;
32  }

33
34  vector<int> getH(string s){
35    vector<int> h;
36    h.resize(s.size() + 1, 0);
37    for (int i = 0; i < s.size(); i++)
38      h[i + 1] = ((h[i] * 1ll * pow) + s[i] - 'a' +
          ↪  1) % mod;
39    return h;
40  }

41
42  int getHash(vector<int> &h, int l, int r){
43    int res = (h[r + 1] - h[l] * p[r - l + 1]) %
          ↪  mod;
44    if (res < 0)
45      res += mod;
46    return res;
47  }

48
49  // string basis ends


1   // min cyclic shift begins
2
3   string min_cyclic_shift (string s) {
4     s += s;
5     int n = (int) s.length();
6     int i=0, ans=0;
7     while (i < n/2) {
8       ans = i;
9       int j=i+1, k=i;
10      while (j < n && s[k] <= s[j]) {
11        if (s[k] < s[j])
12          k = i;
13        else
14          ++k;
15        ++j;
16      }
17      while (i <= k)  i += j - k;
18    }
19    return s.substr (ans, n/2);
20  }

21
22  // min cyclic shift ends

1   // suffix array O(n) begins
2
3   typedef vector<char> bits;
```

```
4
5   template<const int end>
6   void getBuckets(int *s, int *bkt, int n, int K) {
7     fill(bkt, bkt + K + 1, 0);
8     forn(i, n) bkt[s[i] + !end]++;
9     forn(i, K) bkt[i + 1] += bkt[i];
10  }
11  void induceSAl(bits &t, int *SA, int *s, int
    ↪  *bkt, int n, int K) {
12    getBuckets<0>(s, bkt, n, K);
13    forn(i, n) {
14      int j = SA[i] - 1;
15      if (j >= 0 && !t[j])
16        SA[bkt[s[j]]++] = j;
17    }
18  }
19  void induceSAs(bits &t, int *SA, int *s, int
    ↪  *bkt, int n, int K) {
20    getBuckets<1>(s, bkt, n, K);
21    for (int i = n - 1; i >= 0; i--) {
22      int j = SA[i] - 1;
23      if (j >= 0 && t[j])
24        SA[--bkt[s[j]]] = j;
25    }
26  }
27
28  void SA_IS(int *s, int *SA, int n, int K) { //
    ↪  require last symbol is 0
29  #define isLMS(i) (i && t[i] && !t[i-1])
30    int i, j;
31    bits t(n);
32    t[n-1] = 1;
33    for (i = n - 3; i >= 0; i--)
34      t[i] = (s[i]<s[i+1] || (s[i]==s[i+1] &&
         ↪  t[i+1]==1));
35    int bkt[K + 1];
36    getBuckets<1>(s, bkt, n, K);
37    fill(SA, SA + n, -1);
38    forn(i, n)
39      if (isLMS(i))
40        SA[--bkt[s[i]]] = i;
41    induceSAl(t, SA, s, bkt, n, K);
42    induceSAs(t, SA, s, bkt, n, K);
43    int n1 = 0;
44    forn(i, n)
45      if (isLMS(SA[i]))
46        SA[n1++] = SA[i];
47    fill(SA + n1, SA + n, -1);
48    int name = 0, prev = -1;
49    forn(i, n1) {
50      int pos = SA[i];
51      bool diff = false;
52      for (int d = 0; d < n; d++)
53        if (prev == -1 || s[pos+d] != s[prev+d] ||
           ↪  t[pos+d] != t[prev+d])
54          diff = true, d = n;
55        else if (d > 0 && (isLMS(pos+d) ||
           ↪  isLMS(prev+d)))
56          d = n;
57      if (diff)
58        name++, prev = pos;
59      SA[n1 + (pos >> 1)] = name - 1;
60    }
61    for (i = n - 1, j = n - 1; i >= n1; i--)
62      if (SA[i] >= 0)
63        SA[j--] = SA[i];
64    int *s1 = SA + n - n1;
65    if (name < n1)
66      SA_IS(s1, SA, n1, name - 1);
67    else
68      forn(i, n1)
69        SA[s1[i]] = i;
70    getBuckets<1>(s, bkt, n, K);
71    for (i = 1, j = 0; i < n; i++)
72      if (isLMS(i))
73        s1[j++] = i;
74    forn(i, n1)
75      SA[i] = s1[SA[i]];
76    fill(SA + n1, SA + n, -1);
77    for (i = n1 - 1; i >= 0; i--) {
78      j = SA[i], SA[i] = -1;
79      SA[--bkt[s[j]]] = j;
80    }
81    induceSAl(t, SA, s, bkt, n, K);
82    induceSAs(t, SA, s, bkt, n, K);
83  }
84  // suffix array O(n) ends
85
86  // suffix array O(n log n) begins
87  string str;
88  int N, m, SA [MAX_N], LCP [MAX_N];
89  int x [MAX_N], y [MAX_N], w [MAX_N], c [MAX_N];
90
91  inline bool cmp (const int a, const int b, const
    ↪  int l) { return (y [a] == y [b] && y [a + l]
    ↪  == y [b + l]); }
92
93  void Sort () {
94    for (int i = 0; i < m; ++i) w[i] = 0;
95    for (int i = 0; i < N; ++i) ++w[x[y[i]]];
96    for (int i = 0; i < m - 1; ++i) w[i + 1] +=
       ↪  w[i];
97    for (int i = N - 1; i >= 0; --i)
       ↪  SA[--w[x[y[i]]]] = y[i];
98  }
99
100 void DA () {
101   for (int i = 0; i < N; ++i) x[i] = str[i], y[i]
       ↪  = i;
102   Sort ();
103   for (int i, j = 1, p = 1; p < N; j <<= 1, m =
       ↪  p) {
104     for (p = 0, i = N - j; i < N; i++) y[p++] =
         ↪  i;
105     for (int k = 0; k < N; ++k) if (SA[k] >= j)
         ↪  y[p++] = SA[k] - j;
106     Sort();
107     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++;
108   }
109 }
```

```
110   // common for all algorithms
111   void kasaiLCP () {
112     for (int i = 0; i < N; i++) c[SA[i]] = i;
113     for (int i = 0, j, k = 0; i < N; LCP [c[i++]] =
114       ↪ k)
115       if (c [i] > 0) for (k ? k-- : 0, j = SA[c[i]
116         ↪ - 1]; str[i + k] == str[j + k]; k++);
117       else k = 0;
      }
118
119   void suffixArray () { // require last symbol is
      ↪ char(0)
120     m = 256;
121     N = str.size();
122     DA ();
123     kasaiLCP ();
124   }
125   // suffix array O(n log n) ends


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

```
42          while (cur != -1 && v[cur].go.count(c) !=
            ↪ 0 && v[cur].go[c] == a){
43            v[cur].go[c] = b;
44            //v[a].sum_in -= v[cur].sum_in;
45            //v[b].sum_in += v[cur].sum_in;
46            cur = v[cur].suff;
47          }
48          v[a].suff = b;
49        }
50      return;
51      }
52    }
53  }
54
55  // bad suffix automaton ends

1   // pollard begins
2
3   const int max_step = 4e5;
4
5   unsigned long long gcd(unsigned long long a,
    ↪ unsigned long long b){
6     if (!a) return 1;
7     while (a) swap(a, b%=a);
8     return b;
9   }
10
11  unsigned long long get(unsigned long long a,
    ↪ unsigned long long b){
12    if (a > b)
13      return a-b;
14    else
15      return b-a;
16  }
17
18  unsigned long long pollard(unsigned long long n){
19    unsigned long long x = (rand() + 1) % n, y = 1,
      ↪ g;
20    int stage = 2, i = 0;
21    g = gcd(get(x, y), n);
22    while (g == 1) {
23      if (i == max_step)
24        break;
25      if (i == stage) {
26        y = x;
27        stage <<= 1;
28      }
29      x = (x * (__int128)x + 1) % n;
30      i++;
31      g = gcd(get(x, y), n);
32    }
33    return g;
34  }
35
36  // pollard ends

1   // linear sieve begins
2
3   const int N = 1000000;
4
5   int pr[N + 1], sz = 0;
```

```cpp
/* 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);
}

// linear sieve ends

// discrete log in sqrt(p) begins

int k = sqrt((double)p) + 2;

for (int i = k; i >= 1; i--)
  mp[bin(b, (i * 1ll * k) % (p-1), p)] = i;

bool answered = false;
int ans = INT32_MAX;
for (int i = 0; i <= k; i++){
  int sum = (n * 1ll * bin(b, i, p)) % p;
  if (mp.count(sum) != 0){
    int an = mp[sum] * 1ll * k - i;
    if (an < p)
      ans = min(an, ans);
  }
}

// discrete log in sqrt(p) ends

// prime roots mod n begins

int num = 0;
long long phi = n, nn = n;
for (long long x:primes){
  if (x*x>nn)
    break;
  if (nn % x == 0){
    while (nn % x == 0)
      nn /= x;
    phi -= phi/x;
    num++;
  }
}
if (nn != 1){
  phi -= phi/nn;
  num++;
}
if (!((num == 1 && n % 2 != 0) || n == 4 || n ==
↪   2 || (num == 2 && n % 2 == 0 && n % 4 != 0))))
↪   {
  cout << "-1\n";
```

```cpp
  continue;
}
vector<long long> v;
long long pp = phi;
for (long long x:primes){
  if (x*x>pp)
    break;
  if (pp % x == 0){
    while (pp % x == 0)
      pp /= x;
    v.push_back(x);
  }
}
if (pp != 1){
  v.push_back(pp);
}
while (true){
  long long a = primes[rand()%5000]%n;
  if (gcd(a, n) != 1)
    continue;
  bool bb = false;
  for (long long x:v)
    if (pow(a, phi/x) == 1){
      bb = true;
      break;
    }
  if (!bb){
    cout << a << "\n";
    break;
  }
}

// prime roots mod n ends

// simplex begins

const double EPS = 1e-9;

typedef vector<double> vdbl;

// n variables, m inequalities
// Ax <= b, c*x -> max, x >= 0
double simplex( int n, int m, const vector<vdbl>
↪   &a0, const vdbl &b, const vdbl &c, vdbl &x )
↪   {
  // Ax + Ez = b, A[m]*x -> max
  // x = 0, z = b, x >= 0, z >= 0
  vector<vdbl> a(m + 2, vdbl(n + m + 2));
  vector<int> p(m);
  forn(i, n)
    a[m + 1][i] = c[i];
  forn(i, m) {
    forn(j, n)
      a[i][j] = a0[i][j];
    a[i][n + i] = 1;
    a[i][m + n] = -1;
    a[i][m + n + 1] = b[i];
    p[i] = n + i;
  }

  // basis: enter "j", leave "ind+n"
  auto pivot = [&]( int j, int ind ) {
```

```
27    double coef = a[ind][j];
28    assert(fabs(coef) > EPS);
29    forn(col, n + m + 2)
30      a[ind][col] /= coef;
31    forn(row, m + 2)
32      if (row != ind && fabs(a[row][j]) > EPS) {
33        coef = a[row][j];
34        forn(col, n + m + 2)
35          a[row][col] -= a[ind][col] * coef;
36        a[row][j] = 0; // reduce precision error
37      }
38    p[ind] = j;
39  };
40
41  // the Simplex itself
42  auto iterate = [&]( int nn ) {
43    for (int run = 1; run; ) {
44      run = 0;
45      forn(j, nn) {
46        if (a[m][j] > EPS) { // strictly positive
47          run = 1;
48          double mi = INFINITY, t;
49          int ind = -1;
50          forn(i, m)
51            if (a[i][j] > EPS && (t = a[i][n + m
                ↪ + 1] / a[i][j]) < mi - EPS)
52              mi = t, ind = i;
53          if (ind == -1)
54            return false;
55          pivot(j, ind);
56        }
57      }
58    }
59    return true;
60  };
61
62  int mi = min_element(b.begin(), b.end()) -
        ↪ b.begin();
63  if (b[mi] < -EPS) {
64    a[m][n + m] = -1;
65    pivot(n + m, mi);
66    assert(iterate(n + m + 1));
67    if (a[m][m + n + 1] > EPS) // optimal value
        ↪ is positive
68      return NAN;
69    forn(i, m)
70      if (p[i] == m + n) {
71        int j = 0;
72        while (find(p.begin(), p.end(), j) !=
            ↪ p.end() || fabs(a[i][j]) < EPS)
73          j++, assert(j < m + n);
74        pivot(j, i);
75      }
76  }
77  swap(a[m], a[m + 1]);
78  if (!iterate(n + m))
79    return INFINITY;
80  x = vdbl(n, 0);
81  forn(i, m)
82    if (p[i] < n)
83      x[p[i]] = a[i][n + m + 1];
84  return -a[m][n + m + 1];
85 }
86
87 // simplex usage:
88 vdbl x(n);
89 double result = simplex(n, m, a, b, c, x);
90 if (isinf(result))
91   puts("Unbounded");
92 else if (isnan(result))
93   puts("No solution");
94 else {
95   printf("%.9f :", result);
96   forn(i, n)
97     printf(" %.9f", x[i]);
98   puts("");
99 }
100
101 // simplex ends
```

```
// sum over subsets begins
// fast subset convolution O(n 2^n)
for(int i = 0; i < (1<<N); ++i)
  F[i] = A[i];
for(int i = 0; i < N; ++i) for(int mask = 0; mask
    ↪ < (1<<N); ++mask){
  if(mask & (1<<i))
    F[mask] += F[mask^(1<<i)];
}
// sum over subsets ends
```

```
// algebra begins

Pick
B + Г / 2 - 1,
где B - количество целочисленных точек внутри
    ↪ многоугольника, а Г - количество
    ↪ целочисленных точек на границе
    ↪ многоугольника.

Newton
x_{i+1}=x_i-f(x_i)/f'(x_i)

Catalan
C_n=\sum\limits_{k=0}^{n-1} C_{k} C_{n-1-k}
C_i=\frac 1 {n + 1} \binom {2n} {n}

G_N:=2^{n(n-1)/2}
Количество связных помеченных графов
Conn_N = G_N - \frac 1 N \sum\limits_{K=1}^{N-1}
    ↪ K \binom N K Conn_K G_{N-K}

Количество помеченных графов с K компонентами
    ↪ связности
D[N][K]=\sum\limits_{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
```

```
26    a=a^2
27  return a==1 ? PRIME : NOT PRIME
28
```

Интегрирование по формуле Симпсона

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

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

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

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

Погрешность имеет порядок уменьшения как $O(n^4)$.

```
38  // algebra ends

1   // wavelet tree begins
2
3   struct wavelet_tree{
4     int lo, hi;
5     wavelet_tree *l, *r;
6     vi b;
7
8     //nos are in range [x,y]
9     //array indices are [from, to)
10    wavelet_tree(int *from, int *to, int x, int y){
11      lo = x, hi = y;
12      if(lo == hi or from >= to) return;
13
14      int mid = (lo+hi)/2;
15      auto f = [mid](int x){
16        return x <= mid;
17      };
18      b.reserve(to-from+1);
19      b.pb(0);
20      //b[i] = no of elements from first "i"
        ↪   elements that go to left node
21      for(auto it = from; it != to; it++)
22        b.pb(b.back() + f(*it));
23
24      //see how lambda function is used here
25      auto pivot = stable_partition(from, to, f);
26      l = new wavelet_tree(from, pivot, lo, mid);
27      r = new wavelet_tree(pivot, to, mid+1, hi);
28    }
29
30    //kth smallest element in [l, r]
31    int kth(int l, int r, int k){
32      if(l > r) return 0;
33      if(lo == hi) return lo;
34      //how many nos are there in left node from
        ↪   [l, r]
35      int inleft = b[r] - b[l-1];
36      int lb = b[l-1]; //amt of nos from first
        ↪   (l-1) nos that go in left
37      int rb = b[r]; //amt of nos from first (r)
        ↪   nos that go in left
38      //so [lb+1, rb] represents nos from [l, r]
        ↪   that go to left
39      if(k <= inleft) return this->l->kth(lb+1, rb
        ↪   , k);
```

```
40
41      //(l-1-lb) is amt of nos from first (l-1) nos
        ↪   that go to right
42      //(r-rb) is amt of nos from first (r) nos
        ↪   that go to right
43      //so [l-lb, r-rb] represents nos from [l, r]
        ↪   that go to right
44      return this->r->kth(l-lb, r-rb, k-inleft);
45    }
46  };
47  // wavelet tree ends

1   // berlecamp-massey begins
2
3   const int SZ = MAXN;
4
5   ll qp(ll a, ll b) {
6     ll x = 1;
7     a %= MOD;
8     while (b) {
9       if (b & 1) x = x * a % MOD;
10      a = a * a % MOD;
11      b >>= 1;
12    }
13    return x;
14  }
15
16  namespace linear_seq {
17    inline vector<int> BM(vector<int> x) {
18      vector<int> ls, cur;
19      int lf, ld;
20      for (int i = 0; i < int(x.size()); ++i) {
21        ll t = 0;
22        for (int j = 0; j < int(cur.size()); ++j)
23          t = (t + x[i - j - 1] * (ll) cur[j]) %
            ↪   MOD;
24        if ((t - x[i]) % MOD == 0) continue;
25        if (!cur.size()) {
26          cur.resize(i + 1);
27          lf = i;
28          ld = (t - x[i]) % MOD;
29          continue;
30        }
31        ll k = -(x[i] - t) * qp(ld, MOD - 2) % MOD;
32        vector<int> c(i - lf - 1);
33        c.pb(k);
34        for (int j = 0; j < int(ls.size()); ++j)
35          c.pb(-ls[j] * k % MOD);
36        if (c.size() < cur.size())
            ↪   c.resize(cur.size());
37        for (int j = 0; j < int(cur.size()); ++j)
38          c[j] = (c[j] + cur[j]) % MOD;
39        if (i - lf + (int) ls.size() >= (int)
            ↪   cur.size())
40          ls = cur, lf = i, ld = (t - x[i]) % MOD;
41        cur = c;
42      }
43      for (int i = 0; i < int(cur.size()); ++i)
44        cur[i] = (cur[i] % MOD + MOD) % MOD;
45      return cur;
46    }
```

```cpp
47
48      int m;
49      ll a[SZ], h[SZ], t_[SZ], s[SZ], t[SZ];
50
51      inline void mull(ll* p, ll* q) {
52        for (int i = 0; i < m + m; ++i) t_[i] = 0;
53        for (int i = 0; i < m; ++i)
54          if (p[i])
55            for (int j = 0; j < m; ++j)
56              t_[i + j] = (t_[i + j] + p[i] * q[j]) %
              ↪   MOD;
57        for (int i = m + m - 1; i >= m; --i)
58          if (t_[i])
59            for (int j = m - 1; ~j; --j)
60              t_[i - j - 1] = (t_[i - j - 1] + t_[i]
              ↪   * h[j]) % MOD;
61        for (int i = 0; i < m; ++i) p[i] = t_[i];
62      }
63
64      inline ll calc(ll K) {
65        for (int i = m; ~i; --i)
66          s[i] = t[i] = 0;
67        s[0] = 1;
68        if (m != 1) t[1] = 1; else t[0] = h[0];
69        while (K) {
70          if (K & 1) mull(s, t);
71          mull(t, t);
72          K >>= 1;
73        }
74        ll su = 0;
75        for (int i = 0; i < m; ++i) su = (su + s[i] *
          ↪   a[i]) % MOD;
76        return (su % MOD + MOD) % MOD;
77      }
78
79      inline int work(vector<int> x, ll n) {
80        if (n < int(x.size())) return x[n];
81        vector<int> v = BM(x);
82        m = v.size();
83        if (!m) return 0;
84        for (int i = 0; i < m; ++i) h[i] = v[i], a[i]
          ↪   = x[i];
85        return calc(n);
86      }
87
88      //b=a0/(1-p)
89      inline void calc_generating_function(const
        ↪   vector<int>& b, vector<int>& p,
        ↪   vector<int>& a0) {
90        p = BM(b);
91        a0.resize(p.size());
92        for (int i = 0; i < a0.size(); ++i) {
93          a0[i] = b[i];
94          for (int j = 0; j < i; ++j) {
95            a0[i] += MOD - (p[j] * 1ll * b[i - j -
            ↪   1]) % MOD;
96            if (a0[i] > MOD) {
97              a0[i] -= MOD;
98            }
99          }
100       }
101     }
102   }
103
104   // berlecamp-massey ends

1     // AND-FFT begins
2
3     void fast_fourier(vector<int>& a) { // AND-FFT.
4       for (int k = 1; k < SZ(a); k *= 2)
5         for (int start = 0; start < (1 << K); start +=
        ↪   2 * k) {
6           for (int off = 0; off < k; ++off) {
7             int a_val = a[start + off];
8             int b_val = a[start + k + off];
9
10            a[start + off] = b_val;
11            a[start + k + off] = add(a_val, b_val);
12          }
13        }
14    }
15
16    void inverse_fast_fourier(vector<int>& a) {
17      for (int k = 1; k < SZ(a); k *= 2)
18        for (int start = 0; start < (1 << K); start +=
        ↪   2 * k) {
19          for (int off = 0; off < k; ++off) {
20            int a_val = a[start + off];
21            int b_val = a[start + k + off];
22
23            a[start + off] = sub(b_val, a_val);
24            a[start + k + off] = a_val;
25          }
26        }
27    }
28
29    // AND-FFT ends

1     // 2-chinese begins
2
3     template <typename Info>
4     class DSU {
5       public:
6       DSU ( int n ) : jump (new int[n]), rank (new
        ↪   int [n]), info (new Info [n]) {
7         for (int i = 0; i < n; i++) {
8           jump[i] = i;
9           rank[i] = 0;
10        }
11      }
12      Info& operator [] ( int x ) {
13        return info[get (x)];
14      }
15      void merge ( int a, int b, const Info &comment
        ↪   ) {
16        a = get (a);
17        b = get (b);
18        if (rank[a] <= rank[b]) {
19          jump[a] = b;
20          rank[b] += rank[a] == rank[b];
21          info[b] = comment;
22        } else {
23          jump[b] = a;
```

```
24      info[a] = comment;
25      }
26    }
27  private:
28    int *jump, *rank;
29    Info *info;
30
31    int get ( int x ) {
32      return jump[x] == x ? x : (jump[x] = get
          ↪   (jump[x]));
33    }
34  };
35
36
37  struct Treap {
38    int value, add;
39    int source, target, height;
40    int min_value, min_path;
41
42    Treap *left, *right;
43
44    Treap ( int _source, int _target, int _value )
          ↪   : value (_value), add (0), source
          ↪   (_source), target (_target) {
45    height = rand ();
46    min_value = value, min_path = 0;
47    left = right = 0;
48    }
49
50    Treap& operator += ( int sub ) {
51    add += sub;
52    return *this;
53    }
54
55    void push () {
56    if (!add)
57      return;
58    if (left) {
59      left->add += add;
60    }
61    if (right) {
62      right->add += add;
63    }
64    value += add;
65    min_value += add;
66    add = 0;
67    }
68
69    void recalc () {
70    min_value = value;
71    min_path = 0;
72    if (left && left->min_value + left->add <
        ↪   min_value) {
73      min_value = left->min_value + left->add;
74      min_path = -1;
75    }
76    if (right && right->min_value + right->add <
        ↪   min_value) {
77      min_value = right->min_value + right->add;
78      min_path = +1;
79    }
80    }
81  };
82
83  Treap* treap_merge ( Treap *x, Treap *y ) {
84    if (!x)
85    return y;
86    if (!y)
87    return x;
88    if (x->height < y->height) {
89    x->push ();
90    x->right = treap_merge (x->right, y);
91    x->recalc ();
92    return x;
93    } else {
94    y->push ();
95    y->left = treap_merge (x, y->left);
96    y->recalc ();
97    return y;
98    }
99  }
100
101 Treap* treap_getmin ( Treap *x, int &source, int
        ↪   &target, int &value ) {
102    assert (x);
103    x->push ();
104    if (x->min_path == 0) {
105    // memory leak, sorry
106    source = x->source;
107    target = x->target;
108    value = x->value + x->add;
109    return treap_merge (x->left, x->right);
110    } else if (x->min_path == -1) {
111    x->left = treap_getmin (x->left, source,
          ↪   target, value);
112    value += x->add;
113    x->recalc ();
114    return x;
115    } else if (x->min_path == +1) {
116    x->right = treap_getmin (x->right, source,
          ↪   target, value);
117    value += x->add;
118    x->recalc ();
119    return x;
120    } else
121    assert (0);
122  }
123
124 Treap* treap_add ( Treap *x, int add ) {
125    if (!x)
126    return 0;
127    return &((*x) += add);
128  }
129
130
131 int main () {
132    int n, m;
133    while (scanf ('"%d%d"', &n, &m) == 2) {
134    Treap * g[n + 1];
135    for (int i = 0; i <= n; i++)
136      g[i] = 0;
137    for (int i = 1; i <= n; i++) {
```

```
138      int a;
139      assert (scanf ('"%d"', &a) == 1);
140      g[i] = treap_merge (g[i], new Treap (i, 0,
         ↪  a));
141    }
142    n++;
143    for (int i = 0; i < m; i++) {
144      int a, b, c;
145      assert (scanf ('"%d%d%d"', &a, &b, &c) == 3);
146      g[b] = treap_merge (g[b], new Treap (b, a,
         ↪  c));
147    }
148    DSU <pair <int, Treap*> > dsu (n + 1);
149    for (int i = 0; i < n; i++) {
150      dsu[i] = make_pair (i, g[i]);
151    }
152
153    int ans = 0, k = n;
154    int jump[2 * n], jump_from[2 * n], parent[2 *
         ↪  n], c[n];
155    vector <int> children[2 * n];
156    memset (c, 0, sizeof (c[0]) * n);
157    memset (parent, -1, sizeof (parent[0]) * 2 *
         ↪  n);
158    vector <int> finish;
159    for (int i = 0; i < n; i++) {
160      if (dsu[i].first == 0)
161      continue;
162      int u = i;
163      c[u] = 1;
164      while (true) {
165      int source, target, value;
166      dsu[u].second = treap_getmin (dsu[u].second,
         ↪  source, target, value);
167      if (dsu[target] == dsu[u])
168        continue;
169      treap_add (dsu[u].second, -value);
170      ans += value;
171      jump_from[dsu[u].first] = source;
172      jump[dsu[u].first] = target;
173      if (dsu[target].first == 0)
174        break;
175      if (!c[target]) {
176        c[target] = 1;
177        u = target;
178        continue;
179      }
180      assert (k < 2 * n);
181      int node = k++, t = target;
182      parent[dsu[u].first] = node;
183      children[node].push_back (dsu[u].first);
184      dsu[u].first = node;
185      Treap *v = dsu[u].second;
186      while (dsu[t].first != node) {
187        int next = jump[dsu[t].first];
188        parent[dsu[t].first] = node;
189        children[node].push_back (dsu[t].first);
190        v = treap_merge (v, dsu[t].second);
191        dsu.merge (u, t, make_pair (node, v));
192        t = next;
193      }
194      }
195      u = i;
196      while (dsu[u].first) {
197      int next = jump[dsu[u].first];
198      finish.push_back (dsu[u].first);
199      dsu.merge (u, 0, make_pair (0, (Treap *)0));
200      u = next;
201      }
202    }
203    bool ok[k];
204    int res[n];
205    memset (ok, 0, sizeof (ok[0]) * k);
206    memset (res, -1, sizeof (res[0]) * n);
207    function <void (int, int)> add_edge = [&ok,
         ↪  &parent, &res, &n] ( int a, int b ) {
208      assert (0 <= a && a < n);
209      assert (0 <= b && b < n);
210      assert (res[a] == -1);
211      res[a] = b;
212      while (a != -1 && !ok[a]) {
213      ok[a] = true;
214      a = parent[a];
215      }
216    };
217    function <void (int)> reach = [&ok, &reach,
         ↪  &children, &jump, &jump_from, &add_edge](
         ↪   int u ) {
218      if (!ok[u])
219      add_edge (jump_from[u], jump[u]);
220      for (auto x : children[u])
221      reach (x);
222    };
223    for (auto x : finish)
224      reach (x);
225    printf ('"%d\n"', ans);
226    for (int i = 1; i < n; i++)
227      printf ('"%d%c"', res[i] ? res[i] : -1, '"\n "'[i
         ↪  < n - 1]);
228  }
229  return 0;
230 }
231
232 // 2-chinese ends

1   // slow min circulation begins
2
3   struct Edge {
4     int a;
5     int b;
6     int cost;
7   };
8
9   vector<int> negative_cycle(int n, vector<Edge>
     ↪  &edges) {
10    // O(nm), return ids of edges in negative cycle
11
12    vector<int> d(n);
13    vector<int> p(n, -1); // last edge ids
14
15    const int inf = 1e9;
16
17    int x = -1;
```

```
18    for (int i = 0; i < n; i++) {
19      x = -1;
20      for (int j = 0; j < edges.size(); j++) {
21        Edge &e = edges[j];
22
23        if (d[e.b] > d[e.a] + e.cost) {
24          d[e.b] = max(-inf, d[e.a] + e.cost);
25          p[e.b] = j;
26          x = e.b;
27        }
28      }
29    }
30
31    if (x == -1)
32      return vector<int>(); // no negative cycle
33
34    for (int i = 0; i < n; i++)
35      x = edges[p[x]].a;
36
37    vector<int> result;
38    for (int cur = x; ; cur = edges[p[cur]].a) {
39      if (cur == x && result.size() > 0) break;
40      result.push_back(p[cur]);
41    }
42    reverse(result.begin(), result.end());
43
44    return result;
45  }
46
47  vector<int> min_avg_cycle(int n, vector<Edge>
     ↪ &edges) {
48    const int inf = 1e3;
49
50    for (auto &e : edges)
51      e.cost *= n * n;
52
53    int l = -inf;
54    int r = inf;
55    while (l + 1 < r) {
56      int m = (l + r) / 2;
57      for (auto &e : edges)
58        e.cost -= m;
59
60      if (negative_cycle(n, edges).empty())
61        l = m;
62      else
63        r = m;
64
65      for (auto &e : edges)
66        e.cost += m;
67    }
68
69    if (r >= 0) // if only negative needed
70      return vector<int>();
71
72    for (auto &e : edges)
73      e.cost -= r;
74
75    vector<int> result = negative_cycle(n, edges);
76
77
78      for (auto &e : edges)
79        e.cost += r;
80
81      for (auto &e : edges)
82        e.cost /= n * n;
83
84      return result;
85  }
86
87  struct edge {
88    int from, to;
89    int c, f, cost;
90  };
91
92  const int max_n = 200;
93
94  vector<int> gr[max_n];
95  vector<edge> edges;
96
97  void add(int fr, int to, int c, int cost) {
98    gr[fr].push_back(edges.size());
99    edges.push_back({fr, to, c, 0, cost});
100   gr[to].push_back(edges.size());
101   edges.push_back({to, fr, 0, 0, -cost}); //
     ↪ single
102 }
103
104 void calc_min_circulation(int n) {
105   while (true) {
106     vector<Edge> eds;
107     vector<int> origin;
108
109     for (int i = 0; i < edges.size(); i++) {
110       edge &e = edges[i];
111       if (e.c - e.f > 0) {
112         eds.push_back({e.from, e.to, e.cost});
113         origin.push_back(i);
114       }
115     }
116
117     vector<int> cycle = negative_cycle(n, eds);
118
119     if (cycle.empty())
120       break;
121
122     for (auto id : cycle) {
123       int x = origin[id];
124       edges[x].f += 1;
125       edges[x ^ 1].f -= 1;
126     }
127   }
128 }
129
130 // slow min circulation ends


1   // fast hashtable begins
2
3   #include <ext/pb_ds/assoc_container.hpp>
4   using namespace __gnu_pbds;
5   gp_hash_table<int, int> table;
6
```

```
7   const int RANDOM =
    ↪   chrono::high_resolution_clock::now().time_since_epoch().count();
8   struct chash {
9     int operator()(int x) { return hash<int>{}(x ^
      ↪   RANDOM); }
10  };
11  gp_hash_table<key, int, chash> table;
12
13  // fast hashtable ends
```

DM

Кол-во (корневых) деревьев:
$t(G) = \frac{1}{n}\lambda_2 \ldots \lambda_n \ (\lambda_1 = 0)$

Кол-во эйлеровых циклов:
$e(D) = t^-(D, x) \cdot \prod\limits_{y \in D} (outdeg(y) - 1)!$

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.

```
xmodmap -e 'keycode 94='
setxkbmap us
```