Contents

1 Some useful stuff 1
- 1.1 Fast I/O . . . 1
- 1.2 Pragmas . . . 3

2 Data structures 3
- 2.1 Cartesian Tree . . . 3
- 2.2 Dynamic convex hull trick . . . 3
- 2.3 Fenwick tree . . . 4
- 2.4 Hash table . . . 4
- 2.5 Persistent Segment Tree . . . 4
- 2.6 Ordered set and bitset . . . 5

3 DP optimizations 5
- 3.1 Problem Statement . . . 5
- 3.2 Divide and Conquer optimization . . . 5
- 3.3 Knuth optimization . . . 5

4 Geometry 5
- 4.1 Common tangents of two circles . . . 5
- 4.2 Convex hull 3D in $O(n^2)$ . . . 6
- 4.3 Minimal covering disk . . . 6
- 4.4 Polygon tangent . . . 6
- 4.5 Rotate 3D . . . 7
- 4.6 Rotation matrix 2D . . . 7
- 4.7 Sphere distance . . . 7
- 4.8 Draw svg pictures . . . 7

5 Graphs 7
- 5.1 HLD . . . 7
- 5.2 Dinic's algorithm . . . 8
- 5.3 General matching . . . 8
- 5.4 Hungarian algorithm . . . 9
- 5.5 Kuhn and Min Vertex Covering . . . 10
- 5.6 Mincost . . . 10

6 Numeric 11
- 6.1 Burnside's lemma . . . 11
- 6.2 Chinese remainder theorem . . . 11
- 6.3 AND/OR/XOR convolution . . . 12
- 6.4 Counting size of the maximum general matching . . . 12
- 6.5 Counting number of spanning trees . . . 12
- 6.6 Some formulas . . . 12
- 6.7 Miller–Rabin primality test . . . 12
- 6.8 Taking by modullo (Inline assembler) . . . 12
- 6.9 First solution of $(p + step \cdot x) \bmod mod < l$ . . . 13
- 6.10 Multiplication by modulo in `long double` . . . 13
- 6.11 Numerical integration . . . 13
- 6.12 Pollard's rho algorithm . . . 13
- 6.13 Polynom division and inversion . . . 14
- 6.14 Polynom roots . . . 14
- 6.15 Simplex method . . . 15
- 6.16 Some integer sequences . . . 16

7 Strings 16
- 7.1 Aho-Corasick . . . 16
- 7.2 Manacher's algorithm . . . 17
- 7.3 Min Cyclic Shift O(n) . . . 18
- 7.4 Suffix array + LCP . . . 18

1 Some useful stuff

1.1 Fast I/O

```
#include <cassert>
#include <cstdio>
#include <algorithm>

/** Fast allocation */

#ifdef FAST_ALLOCATOR_MEMORY
int allocator_pos = 0;
    char allocator_memory[(int)FAST_ALLOCATOR_MEMORY];
    inline void * operator new ( size_t n ) {
        char *res = allocator_memory + allocator_pos;
        allocator_pos += n;
        assert(allocator_pos <= (int)FAST_ALLOCATOR_MEMORY);
        return (void *)res;
    }
    inline void operator delete ( void * ) noexcept { }
    //inline void * operator new [] ( size_t ) { assert(0); }
    //inline void operator delete [] ( void * ) { assert(0); }
#endif

/** Fast input-output */

template <class T = int> inline T readInt();
inline double readDouble();
inline int readUInt();
inline int readChar(); // first non-blank character
inline void readWord( char *s );
inline bool readLine( char *s ); // do not save '\n'
inline bool isEof();
inline int getChar();
inline int peekChar();
inline bool seekEof();
inline void skipBlanks();

template <class T> inline void writeInt( T x, char end = 0, int len = -1 );
inline void writeChar( int x );
inline void writeWord( const char *s );
inline void writeDouble( double x, int len = 0 ); // works correct only for |x| < 2^{63}
inline void flush();

static struct buffer_flusher_t {
    ~buffer_flusher_t() {
        flush();
    }
} buffer_flusher;

/** Read */

static const int buf_size = 4096;
```

```
static unsigned char buf[buf_size];
static int buf_len = 0, buf_pos = 0;

inline bool isEof() {
    if (buf_pos == buf_len) {
        buf_pos = 0, buf_len = fread(buf, 1,
↪   buf_size, stdin);
        if (buf_pos == buf_len)
            return 1;
    }
    return 0;
}

inline int getChar() {
    return isEof() ? -1 : buf[buf_pos++];
}

inline int peekChar() {
    return isEof() ? -1 : buf[buf_pos];
}

inline bool seekEof() {
    int c;
    while ((c = peekChar()) != -1 && c <= 32)
        buf_pos++;
    return c == -1;
}

inline void skipBlanks() {
    while (!isEof() && buf[buf_pos] <= 32U)
        buf_pos++;
}

inline int readChar() {
    int c = getChar();
    while (c != -1 && c <= 32)
        c = getChar();
    return c;
}

inline int readUInt() {
    int c = readChar(), x = 0;
    while ('0' <= c && c <= '9')
        x = x * 10 + c - '0', c = getChar();
    return x;
}

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

inline double readDouble() {
    int s = 1, c = readChar();
    double x = 0;
    if (c == '-')
        s = -1, c = getChar();
    while ('0' <= c && c <= '9')
        x = x * 10 + c - '0', c = getChar();
    if (c == '.') {
        c = getChar();
        double coef = 1;
        while ('0' <= c && c <= '9')
            x += (c - '0') * (coef *= 1e-1), c =
↪   getChar();
    }
    return s == 1 ? x : -x;
}

inline void readWord( char *s ) {
    int c = readChar();
    while (c > 32)
        *s++ = c, c = getChar();
    *s = 0;
}

inline bool readLine( char *s ) {
    int c = getChar();
    while (c != '\n' && c != -1)
        *s++ = c, c = getChar();
    *s = 0;
    return c != -1;
}

/** Write */

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

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

inline void flush() {
    if (write_buf_pos) {
        fwrite(write_buf, 1, write_buf_pos,
↪   stdout), write_buf_pos = 0;
        fflush(stdout);
    }
}

template <class T>
inline void writeInt( T x, char end, int
↪   output_len ) {
    if (x < 0)
        writeChar('-'), x = -x;

    char s[24];
    int n = 0;
    while (x || !n)
        s[n++] = '0' + x % 10, x /= 10;
    while (n < output_len)
        s[n++] = '0';
```

```
    while (n--)
        writeChar(s[n]);
    if (end)
        writeChar(end);
}

inline void writeWord( const char *s ) {
    while (*s)
        writeChar(*s++);
}

inline void writeDouble( double x, int output_len
↪   ) {
    if (x < 0)
        writeChar('-'), x = -x;
    assert(x <= (1LLU << 63) - 1);
    long long t = (long long)x;
    writeInt(t), x -= t;
    writeChar('.');
    for (int i = output_len - 1; i > 0; i--) {
        x *= 10;
        t = std::min(9, (int)x);
        writeChar('0' + t), x -= t;
    }
    x *= 10;
    t = std::min(9, (int)(x + 0.5));
    writeChar('0' + t);
}

// 10M int [0..1e9)
// cin 3.02
// scanf 1.2
// cin sync_with_stdio(false) 0.71
// fastRead getchar 0.53
// fastRead fread 0.15
```

1.2 Pragmas

```
#pragma GCC optimize(
↪   "Ofast,no-stack-protector,unroll-loops,fast-math")
#pragma GCC target(
↪   "sse,sse2,sse3,ssse3,sse4.1,sse4.2,popcnt,tune=native")
#pragma GCC target("avx,avx2")
```

2 Data structures

2.1 Cartesian Tree

```
struct Node {
  int x, y, sz = 1;
  Node *l = nullptr;
  Node *r = nullptr;
  Node(int x) : x(x), y(rand()) {}
};

int get_sz(Node *root) {
  return (root == nullptr ? 0 : root->sz);
}

void update(Node *root) {
  root->sz = 1 + get_sz(root->l) +
↪   get_sz(root->r);
}

pair<Node *, Node *> split(Node *&root, int x) {
  if (!root) {
    return {nullptr, nullptr};
  }
  if (root->x < x) {
    auto t = split(root->r, x);
    root->r = t.first;
    update(root);
    return {root, t.second};
  }
  auto t = split(root->l, x);
  root->l = t.second;
  update(root);
  return {t.first, root};
}

Node *mergeTree(Node *a, Node *b) {
  if (!a || !b)
    return a ? a : b;
  if (a->y < b->y) {
    a->r = mergeTree(a->r, b);
    update(a);
    return a;
  }
  b->l = mergeTree(a, b->l);
  update(b);
  return b;
}

void ins(Node *&root, int x) {
  Node *nn = new Node(x);
  auto a = split(root, x);
  root = mergeTree(mergeTree(a.first, nn),
↪   a.second);
}

void del(Node *&root, int x) {
  auto a = split(root, x + 1);
  auto b = split(a.first, x);
  root = mergeTree(b.first, a.second);
}
```

2.2 Dynamic convex hull trick

```
const ll is_query = -(1LL << 62);

struct Line {
  ll m, b;
  mutable function<const Line *()> succ;

  bool operator<(const Line &rhs) const {
    if (rhs.b != is_query)
      return m < rhs.m;
    const Line *s = succ();
    if (!s)
      return 0;
    ll x = rhs.m;
    return b - s->b < (s->m - m) * x;
  }
};
```

```
struct HullDynamic : public multiset<Line> {
	bool bad(iterator y) {
		auto z = next(y);
		if (y == begin()) {
			if (z == end())
				return 0;
			return y->m == z->m && y->b <= z->b;
		}
		auto x = prev(y);
		if (z == end())
			return y->m == x->m && y->b <= x->b;
		return (x->b - y->b) * (z->m - y->m) >= (y->b
		↪ - z->b) * (y->m - x->m);
	}

	void insert_line(ll m, ll b) {
		auto y = insert({m, b});
		y->succ = [=] { return next(y) == end() ? 0 :
		↪ &*next(y); };
		if (bad(y)) {
			erase(y);
			return;
		}
		while (next(y) != end() && bad(next(y)))
			erase(next(y));
		while (y != begin() && bad(prev(y)))
			erase(prev(y));
	}

	ll eval(ll x) {
		auto l = *lower_bound((Line){x, is_query});
		return l.m * x + l.b;
	}
};
```

2.3 Fenwick tree

```
struct FT {
	vector<ll> s;
	FT(int n) : s(n) {}
	void update(int pos, ll dif) { // a[pos] +=
	↪ dif
		for (; pos < sz(s); pos |= pos + 1) s[pos] +=
		↪ dif;
	}
	ll query(int pos) { // sum of values in [0,
	↪ pos)
		ll res = 0;
		for (; pos > 0; pos &= pos - 1) res +=
		↪ s[pos-1];
		return res;
	}
	int lower_bound(ll sum) {// min pos st sum of
	↪ [0, pos] >= sum
		// Returns n if no sum is >= sum, or -1 if
		↪ empty sum is.
		if (sum <= 0) return -1;
		int pos = 0;
		for (int pw = 1 << 25; pw; pw >>= 1) {
			if (pos + pw <= sz(s) && s[pos + pw-1] <
			↪ sum)
				pos += pw, sum -= s[pos-1];
		}
		return pos;
	}
};
```

2.4 Hash table

```
template <const int max_size, class HashType,
↪ class Data,
					const Data default_value>
struct hashTable {
	HashType hash[max_size];
	Data f[max_size];
	int size;

	int position(HashType H) const {
		int i = H % max_size;
		while (hash[i] && hash[i] != H)
			if (++i == max_size)
				i = 0;
		return i;
	}

	Data &operator[](HashType H) {
		assert(H != 0);
		int i = position(H);
		if (!hash[i]) {
			hash[i] = H;
			f[i] = default_value;
			size++;
		}
		return f[i];
	}
};

hashTable<13, int, int, 0> h;
```

2.5 Persistent Segment Tree

```
constexpr int N = 1e5 + 7;

struct Node {
  int x;
  int l, r;
  Node *L, *R;

  int size() { return r - l; }
  bool have(int i) { return l <= i && i < r; }
  void upd() { x = L->x + R->x; }

  Node() {}
  Node(int _x, int i) : x(_x), l(i), r(i + 1),
  ↪ L(nullptr), R(nullptr) {}
  Node(Node *_L, Node *_R) : L(_L), R(_R) {
  ↪ upd(); }
} tree[N];

Node *upd(Node *t, int i, int x) {
  if (!t->have(i)) {
    return t;
  } else if (t->size() == 1) {
```

```
    return new Node(x, i);
  } else {
    return new Node(upd(t->L, i, x), upd(t->R, i,
↪   x));
  }
}

int get(Node *t, int ql, int qr) {
  if (t->r <= ql || qr <= t->l) {
    return 0;
  } else if (ql <= t->l && t->r <= qr) {
    return t->x;
  } else {
    return get(t->L, ql, qr) + get(t->R, ql, qr);
  }
}
```

2.6 Ordered set and bitset

```
#include <ext/pb_ds/assoc_container.hpp>
using namespace __gnu_pbds;

template <typename T> using ordered_set = tree<T,
↪   null_type, less<T>, rb_tree_tag,
↪   tree_order_statistics_node_update>;
template <typename K, typename V> using
↪   ordered_map = tree<K, V, less<K>,
↪   rb_tree_tag,
↪   tree_order_statistics_node_update>;

// HOW TO USE ::
// -- order_of_key(10) returns the number of
↪   elements in set/map strictly less than 10
// -- *find_by_order(10) returns 10-th smallest
↪   element in set/map (0-based)

bitset<N> a;
for (int i = a._Find_first(); i != a.size(); i =
↪   a._Find_next(i)) {
| cout << i << endl;
}
```

3 DP optimizations

3.1 Problem Statement

```
// x[] | sorted array of N points
// Question: min cost to cover them with K
↪   segments (cost(seg)=len^2(seg))
int cost(int i, int j) {
    return (x[j] - x[i]) * (x[j] - x[i]);
}
// f[k][n] --- k segments, n first points
for (int k = 1; k <= K; k++)
    for (int n = 1; n <= N; n++)
        for (int i = 0; i < n; i++)
            f[k][n] = min(f[k][n], f[k - 1][i] +
↪   cost(i, n));
```

3.2 Divide and Conquer optimization

```
// [l, r] -- count f[k][l..r]
// [L, R] -- i = L..R
void solve(int l, int r, int L, int R, int k) {
    if (l > r) return;  // empty segment
    int m = (l + r) / 2, opt = L;
    // find answer for f[k][m]
    for (int i = L; i <= min(R, m); i++) {
        int val = f[k - 1][i] + cost(i, m);
        if (val < f[k][m])
            f[k][m] = val, opt = i;
    }
    solve(l, m - 1, L, opt, k);
    solve(m + 1, r, opt, R, k);
}
for (int k = 1; k <= K; k++)
    solve(1, n, 0, n - 1, k);
```

3.3 Knuth optimization

```
// opt[k-1][n] <= opt[k][n] <= opt[k][n+1]
for (int k = 1; k <= K; k++)
    for (int n = N; n >= 1; n--) {
        f[k][n] = +INF;
        for (int i = opt[k - 1][n]; i <= opt[k][n
↪   + 1]; i++) {
            int val = f[k - 1][i] + cost(i, n);
            if (val < f[k][n]) f[k][n] = val,
↪   opt[k][n] = i;
        }
    }
```

4 Geometry

4.1 Common tangents of two circles

```
vector<Line> commonTangents(pt A, dbl rA, pt B,
↪   dbl rB) {
| vector<Line> res;
| pt C = B - A;
| dbl z = C.len2();
| for (int i = -1; i <= 1; i += 2) {
| | for (int j = -1; j <= 1; j += 2) {
| | | dbl r = rB * j - rA * i;
| | | dbl d = z - r * r;
| | | if (ls(d, 0))
| | | | continue;
| | | d = sqrt(max(0.01, d));
| | | pt magic = pt(r, d) / z;
| | | pt v(magic % C, magic * C);
| | | dbl CC = (rA * i - v % A) / v.len2();
| | | pt O = v * -CC;
| | | res.pb(Line(O, O + v.rotate()));
| | }
| }
| return res;
}

// HOW TO USE ::
// --      *D*---------------*F*
// --     *...*-             -*...*
// --    *.....* -         - *.....*
// --   *.......*  -     -  *.......*
// --   *...A...*    --    *...B...*
// --   *.......*  -     -  *.......*
```

```
// --    *.....* -        - *.....*
// --     *...*-           -*...*
// --      *C*---------------*E*
// --  res = {CE, CF, DE, DF}
```

4.2 Convex hull 3D in $O(n^2)$

```
struct Plane {
| pt O, v;
| vector<int> id;
};

vector<Plane> convexHull3(vector<pt> p) {
| vector<Plane> res;
| int n = p.size();
| for (int i = 0; i < n; i++)
| | p[i].id = i;
| for (int i = 0; i < 4; i++) {
| | vector<pt> tmp;
| | for (int j = 0; j < 4; j++)
| | | if (i != j)
| | | | tmp.pb(p[j]);
| | res.pb({tmp[0],
| | | | | | (tmp[1] - tmp[0]) * (tmp[2] -
↪ tmp[0]),
| | | | | | {tmp[0].id, tmp[1].id, tmp[2].id}});
| | if ((p[i] - res.back().O) % res.back().v > 0)
↪ {
| | | res.back().v = res.back().v * -1;
| | | swap(res.back().id[0], res.back().id[1]);
| | }
| }
| vector<vector<int>> use(n, vector<int>(n, 0));
| int tmr = 0;
| for (int i = 4; i < n; i++) {
| | int cur = 0;
| | tmr++;
| | vector<pair<int, int>> curEdge;
| | for (int j = 0; j < sz(res); j++) {
| | | if ((p[i] - res[j].O) % res[j].v > 0) {
| | | | for (int t = 0; t < 3; t++) {
| | | | | int v = res[j].id[t];
| | | | | int u = res[j].id[(t + 1) % 3];
| | | | | use[v][u] = tmr;
| | | | | curEdge.pb({v, u});
| | | | }
| | | } else {
| | | | res[cur++] = res[j];
| | | }
| | }
| | res.resize(cur);
| | for (auto x : curEdge) {
| | | if (use[x.S][x.F] == tmr)
| | | | continue;
| | | res.pb({p[i], (p[x.F] - p[i]) * (p[x.S] -
↪ p[i]), {x.F, x.S, i}});
| | }
| }
| return res;
}

// plane in 3d
// (A, v) * (B, u) -> (O, n)

pt n = v * u;
pt m = v * n;
double t = (B - A) % u / (u % m);
pt O = A - m * t;
```

4.3 Minimal covering disk

```
pair<pt, dbl> minDisc(vector<pt> p) {
| int n = p.size();
| pt O = pt(0, 0);
| dbl R = 0;
| random_shuffle(all(p));
| for (int i = 0; i < n; i++) {
| | if (ls(R, (O - p[i]).len())) {
| | | O = p[i];
| | | R = 0;
| | | for (int j = 0; j < i; j++) {
| | | | if (ls(R, (O - p[j]).len())) {
| | | | | O = (p[i] + p[j]) / 2;
| | | | | R = (p[i] - p[j]).len() / 2;
| | | | | for (int k = 0; k < j; k++) {
| | | | | | if (ls(R, (O - p[k]).len())) {
| | | | | | | Line l1((p[i] + p[j]) / 2,
| | | | | | | | | | (p[i] + p[j]) / 2 + (p[i] -
↪ p[j]).rotate());
| | | | | | | Line l2((p[k] + p[j]) / 2,
| | | | | | | | | | (p[k] + p[j]) / 2 + (p[k] -
↪ p[j]).rotate());
| | | | | | | O = l1 * l2;
| | | | | | | R = (p[i] - O).len();
| | | | | | }
| | | | | }
| | | | }
| | | }
| | }
| }
| return {O, R};
}
```

4.4 Polygon tangent

```
pt tangent(vector<pt>& p, pt O, int cof) {
| int step = 1;
| for (; step < (int)p.size(); step *= 2);
| int pos = 0;
| int n = p.size();
| for (; step > 0; step /= 2) {
| | int best = pos;
| | for (int dx = -1; dx <= 1; dx += 2) {
| | | int id = ((pos + step * dx) % n + n) % n;
| | | if ((p[id] - O) * (p[best] - O) * cof > 0)
| | | | best = id;
| | }
| | pos = best;
| }
| return p[pos];
}
```

4.5 Rotate 3D

```
// Rotate 3d point along axis on angle
/*
 * 2D
 * x' = x cos a - y sin a
 * y' = x sin a + y cos a
*/
struct quater {
    double w, x, y, z; // w + xi + yj + zk
    quater(double tw, const pt3 &v) : w(tw),
        x(v.x), y(v.y), z(v.z) { }
    quater(double tw, double tx, double ty, double
        tz) : w(tw), x(tx), y(ty), z(tz) { }
    pt3 vector() const {
        return {x, y, z};
    }
    quater conjugate() const {
        return {w, -x, -y, -z};
    }
    quater operator*(const quater &q2) {
        return {w * q2.w - x * q2.x - y * q2.y - z *
            q2.z, w * q2.x + x * q2.w + y * q2.z - z *
            q2.y, w * q2.y - x * q2.z + y * q2.w + z *
            q2.x, w * q2.z + x * q2.y - y * q2.x + z *
            q2.w};
    }
};

pt3 rotate(pt3 axis, pt3 p, double angle) {
    quater q = quater(cos(angle / 2), axis *
        sin(angle / 2));
    return (q * quater(0, p) *
        q.conjugate()).vector();
}
```

4.6 Rotation matrix 2D

Rotation of point (x, y) through an angle α in counterclockwise direction in 2D.

$$\begin{pmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{pmatrix} \cdot \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} x' \\ y' \end{pmatrix}$$

4.7 Sphere distance

```
double sphericalDistance(double f1, double t1,
        double f2, double t2, double radius) {
    double dx = sin(t2)*cos(f2) - sin(t1)*cos(f1);
    double dy = sin(t2)*sin(f2) - sin(t1)*sin(f1);
    double dz = cos(t2) - cos(t1);
    double d = sqrt(dx*dx + dy*dy + dz*dz);
    return radius*2*asin(d/2);
}
```

4.8 Draw svg pictures

```
struct SVG {
    FILE *out;
    double sc = 50;
    void open() {
        out = fopen("image.svg", "w");
        fprintf(out, "<svg
            xmlns='http://www.w3.org/2000/svg'
            viewBox='-1000 -1000 2000 2000'>\n");
    }
    void line(point a, point b) {
        a = a * sc, b = b * sc;
        fprintf(out, "<line x1='%f' y1='%f' x2='%f'
            y2='%f' stroke='black'/>\n", a.x, -a.y, b.x,
            -b.y);
    }
    void circle(point a, double r = -1, string col
        = "red") {
        r = sc * (r == -1 ? 0.3 : r);
        a = a * sc;
        fprintf(out, "<circle cx='%f' cy='%f' r='%f'
            fill='%s'/>\n", a.x, -a.y, r, col.c_str());
    }
    void text(point a, string s) {
        a = a * sc;
        fprintf(out, "<text x='%f' y='%f'
            font-size='100px'>%s</text>\n", a.x, -a.y,
            s.c_str());
    }
    void close() {
        fprintf(out, "</svg>\n");
        fclose(out);
        out = 0;
    }
    ~SVG() {
        if (out) {
            close();
        }
    }
} svg;
```

5 Graphs

5.1 HLD

```
const int N = 1e5 + 7;

int n;
vector<int> g[N];

int sz[N];
int nxt[N];
int par[N];
int par_hld[N];

void dfs(int v, int last) {
    sz[v] = 1;
    nxt[v] = -1;
    par[v] = last;
    par_hld[v] = -1;
    for (int u : g[v]) {
        if (u != last) {
            dfs(u, v);
            sz[v] += sz[u];
            if (nxt[v] == -1 || sz[nxt[v]] <
    sz[u]) {
                nxt[v] = u;
            }
        }
    }
    if (nxt[v] != -1) {
        par_hld[nxt[v]] = v;
```

```
    }
}

vector<vector<int>> hld;
int ind_tree[N];
int ind[N];

void build_hld() {
    dfs(0, 0);
    for (int v = 0; v < n; ++v) {
        if (par_hld[v] == -1) {
            int u = v;
            hld.push_back({});
            while (u != -1) {
                hld.back().push_back(u);
                u = nxt[u];
            }
            reverse(hld.back().begin(),
↪   hld.back().end());
            for (int i = 0; i <
↪   hld.back().size(); ++i) {
                int u = hld.back()[i];
                ind_tree[u] = hld.size() - 1;
                ind[u] = i;
            }
        }
    }
}
```

5.2 Dinic's algorithm

```
const int N = 500;
const int M = 10000;

int n, m;
struct Edge {
    int v, u, c, f = 0;
    Edge() {}
    Edge(int v, int u, int c) : v(v), u(u), c(c)
↪   {}
    int cf() { return c - f; }
} e[2 * M];
vector<int> g[N];
int s, t;

void add_edge(int v, int u, int c) {
    static int i = 0;
    e[i] = {v, u, c};
    g[v].push_back(i);
    e[i ^ 1] = {u, v, 0};
    g[u].push_back(i ^ 1);
    i += 2;
}

int df;

int dist[N];

bool bfs() {
    queue<int> q;
    q.push(s);
    fill(dist, dist + n, -1);
    dist[s] = 0;
    while (!q.empty()) {
        int v = q.front();
        q.pop();
        for (int i : g[v]) {
            if (e[i].cf() >= df && dist[e[i].u]
↪   == -1) {
                dist[e[i].u] = dist[v] + 1;
                q.push(e[i].u);
            }
        }
    }
    return dist[t] != -1;
}

int ptr[N];

bool dfs(int v) {
    if (v == t) {
        return true;
    }
    for (; ptr[v] < g[v].size(); ++ptr[v]) {
        int i = g[v][ptr[v]];
        if (dist[v] + 1 == dist[e[i].u] &&
↪   e[i].cf() >= df && dfs(e[i].u)) {
            e[i].f += df;
            e[i ^ 1].f -= df;
            return true;
        }
    }
    return false;
}

void dinic() {
    const int K = 30;
    for (df = 1 << K; df >= 1; df >>= 1) {
        while (bfs()) {
            fill(ptr, ptr + N, 0);
            while (dfs(s)) {
            }
        }
    }
}
```

5.3 General matching

```
// COPYPASTED FROM E-MAXX
namespace general_matching {
const int MAXN = 256;
int n;
vector<int> g[MAXN];
int match[MAXN], p[MAXN], base[MAXN], q[MAXN];
bool used[MAXN], blossom[MAXN];

int lca(int a, int b) {
| bool used[MAXN] = {0};
| for (;;) {
| | a = base[a];
| | used[a] = true;
| | if (match[a] == -1)
| | | break;
| | a = p[match[a]];
```

```
    }
    for (;;) {
        b = base[b];
        if (used[b])
            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 (size_t i = 0; i < g[v].size(); ++i) {
            int to = g[v][i];
            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);
                mark_path(to, curbase, v);
                for (int i = 0; i < n; ++i)
                    if (blossom[base[i]]) {
                        base[i] = curbase;
                        if (!used[i]) {
                            used[i] = true;
                            q[qt++] = i;
                        }
                    }
            } else if (p[to] == -1) {
                p[to] = v;
                if (match[to] == -1)
                    return to;
                to = match[to];
                used[to] = true;
                q[qt++] = to;
            }
        }
    }
    return -1;
}
```

```
vector<pair<int, int>> solve(int _n,
↪ vector<pair<int, int>> edges) {
    n = _n;
    for (int i = 0; i < n; i++)
        g[i].clear();
    for (auto o : edges) {
        g[o.first].push_back(o.second);
        g[o.second].push_back(o.first);
    }
    memset(match, -1, sizeof match);
    for (int i = 0; i < n; ++i) {
        if (match[i] == -1) {
            int v = find_path(i);
            while (v != -1) {
                int pv = p[v], ppv = match[pv];
                match[v] = pv, match[pv] = v;
                v = ppv;
            }
        }
    }
    vector<pair<int, int>> ans;
    for (int i = 0; i < n; i++) {
        if (match[i] > i) {
            ans.push_back(make_pair(i, match[i]));
        }
    }
    return ans;
}
} // namespace general_matching
```

5.4 Hungarian algorithm

```
// maximum bipartite matching problem for a
↪ weighted graph.

namespace hungary {
const int N = 210;

int a[N][N];
int ans[N];

int calc(int n, int m) {
    ++n, ++m;
    vector<int> u(n), v(m), p(m), prev(m);
    for (int i = 1; i < n; ++i) {
        p[0] = i;
        int x = 0;
        vector<int> mn(m, INF);
        vector<int> was(m, 0);
        while (p[x]) {
            was[x] = 1;
            int ii = p[x], dd = INF, y = 0;
            for (int j = 1; j < m; ++j)
                if (!was[j]) {
                    int cur = a[ii][j] - u[ii] - v[j];
                    if (cur < mn[j])
                        mn[j] = cur, prev[j] = x;
                    if (mn[j] < dd)
                        dd = mn[j], y = j;
                }
            for (int j = 0; j < m; ++j) {
                if (was[j])
```

```
|   |   |   |   |   u[p[j]] += dd, v[j] -= dd;
|   |   |   |   else
|   |   |   |   |   mn[j] -= dd;
|   |   |   }
|   |   |   x = y;
|   |   }
|   |   while (x) {
|   |   |   int y = prev[x];
|   |   |   p[x] = p[y];
|   |   |   x = y;
|   |   }
|   }
|   for (int j = 1; j < m; ++j) {
|   |   ans[p[j]] = j;
|   }
|   return -v[0];
}
// How to use:
// * Set values to a[1..n][1..m] (n <= m)
// * Run calc(n, m) to find minimum
// * Optimal edges are (i, ans[i]) for i = 1..n
// * Everything works on negative numbers
//
// !!! I don't understand this code, it's
↪   copypasted from e-maxx
} // namespace hungary
```

5.5 Kuhn and Min Vertex Covering

```
// Left: 0..n-1, Right: 0..m-1
int n, m;
vector<vector<int>> gl, gr;

bool dfs_kuhn(int u, vector<int> &pairR,
↪   vector<int> &color, int c) {
    color[u] = c;
    for (auto v : gl[u]) {
        if (pairR[v] == -1) {
            pairR[v] = u;
            return true;
        }
    }
    for (auto v : gl[u]) {
        if (color[pairR[v]] != c &&
↪   dfs_kuhn(pairR[v], pairR, color, c)) {
            pairR[v] = u;
            return true;
        }
    }
    return false;
}

vector<int> kuhn_matching() {
    vector<int> pairR(m, -1);
    vector<int> color(n, 0);
    for (int u = 0; u < n; ++u) {
        dfs_kuhn(u, pairR, color, u + 1);
    }
    return pairR;
}
void dfs_min_vertex_covering(int u, vector<int>
↪   &pairR, vector<bool> &visited) {
    visited[u] = true;
    for (auto v : gl[u]) {
        if (!visited[pairR[v]]) {
            dfs_min_vertex_covering(pairR[v],
↪   pairR, visited);
        }
    }
}

pair<vector<int>, vector<int>>
↪   min_vertex_covering() {
    vector<int> pairR = kuhn_matching();
    vector<bool> is_paired_left(n, false);
    for (auto &u : pairR) {
        if (u != -1) {
            is_paired_left[u] = true;
        }
    }
    vector<bool> visited(n, false);
    for (int u = 0; u < n; ++u) {
        if (!is_paired_left[u] && !visited[u]) {
            dfs_min_vertex_covering(u, pairR,
↪   visited);
        }
    }
    vector<int> res_left, res_right;
    for (int u = 0; u < n; ++u) {
        if (!visited[u]) {
            res_left.emplace_back(u);
        }
    }
    for (int v = 0; v < m; ++v) {
        if (pairR[v] != -1 && visited[pairR[v]])
↪   {
            res_right.emplace_back(v);
        }
    }
    return {res_left, res_right};
}
```

5.6 Mincost

```
const int N = 107;
const int M = 1007;
const int INF = 1e9 + 7;

struct Edge {
    int v, u, c, f, w;

    int cf() { return c - f; }

    Edge() {}
    Edge(int v, int u, int c, int f, int w) :
↪   v(v), u(u), c(c), f(f), w(w) {}
};

int n, m;
Edge edges[2 * M];
vector<int> g[N];
```

```cpp
void add_edge(int v, int u, int c, int w) {
    static int i = 0;
    g[v].push_back(i);
    g[u].push_back(i + 1);
    edges[i] = {v, u, c, 0, w};
    edges[i + 1] = {u, v, 0, 0, -w};
    i += 2;
}

int dist[N];

bool in_q[N];

void spfa(int start) {
    fill(dist, dist + n, INF);
    fill(in_q, in_q + n, false);
    dist[start] = 0;
    queue<int> q({start});
    in_q[start] = true;
    while (!q.empty()) {
        int v = q.front();
        q.pop();
        in_q[v] = false;
        for (int i : g[v]) {
            Edge e = edges[i];
            if (e.cf() && dist[e.u] > max(-INF, dist[e.v] + e.w)) {
                dist[e.u] = max(-INF, dist[e.v] + e.w);
                if (!in_q[e.u]) {
                    q.push(e.u);
                    in_q[e.u] = true;
                }
            }
        }
    }
}

bool used[N];

int cnt_dfs = 0;

int dfs(int v, int t, int f) {
    ++cnt_dfs;
    if (v == t) {
        return f;
    }
    used[v] = true;
    for (int i : g[v]) {
        Edge e = edges[i];
        if (e.cf() && !used[e.u] && dist[v] + e.w == dist[e.u]) {
            int nxt_f = dfs(e.u, t, min(f, e.cf()));
            if (nxt_f) {
                edges[i].f += nxt_f;
                edges[i ^ 1].f -= nxt_f;
                return nxt_f;
            }
        }
    }
    return 0;
}

int phi[N];

void johnson(int start) {
    fill(dist, dist + n, INF);
    priority_queue<pair<int, int>, vector<pair<int, int>>,
                        greater<pair<int, int>>>
        q;
    dist[start] = 0;
    q.push({dist[start], start});
    while (!q.empty()) {
        int v = q.top().second;
        int d = q.top().first;
        q.pop();
        if (d == dist[v]) {
            for (int i : g[v]) {
                Edge e = edges[i];
                if (e.cf() && dist[e.u] > dist[e.v] + e.w + phi[v] - phi[e.u]) {
                    assert(e.w + phi[v] - phi[e.u] >= 0);
                    dist[e.u] = dist[e.v] + e.w + phi[v] - phi[e.u];
                    q.push({dist[e.u], e.u});
                }
            }
        }
    }
}

void mincost(int s, int t) {
    // TODO: delete all negative cycles
    spfa(s);
    copy(dist, dist + n, phi);
    while (dfs(s, t, INF)) {
        johnson(s);
        for (int v = 0; v < n; ++v) {
            dist[v] += phi[v];
            phi[v] = dist[v];
        }
        fill(used, used + n, false);
    }
}
```

6 Numeric

6.1 Burnside's lemma

$$|X/G| = \frac{1}{|G|}\sum_{g\in G}|St(g)|$$

$St(g)$ denote the set of elements in X that are fixed by g, i.e. $St(g) = \{x \in X | gx = x\}$.

6.2 Chinese remainder theorem

```cpp
int CRT(int a1, int m1, int a2, int m2) {
    return (a1 - a2 % m1 + m1) * (ll)rev(m2, m1) % m1 * m2 + a2;
}
```

6.3 AND/OR/XOR convolution

```
// Transform to a basis with fast convolutions of
↪ the form c[z] = sum_{z=x⊕y} a[x] · b[y],
// where ⊕ is one of AND, OR, XOR.
// The size of a must be a power of two.

void FST(vector<int> &a, bool inv) {
    int n = szof(a);
    for (int step = 1; step < n; step *= 2) {
        for (int i = 0; i < n; i += 2 * step) {
            for (j = i; j < i + step; ++j) {
                int &u = a[j], &v = a[j + step];
                tie(u, v) =
                    inv ? pii(v - u, u) : pii(v, u + v); //
↪ AND
                    inv ? pii(v, u - v) : pii(u + v, u); //
↪ OR
                    pii(u + v, u - v); // XOR
            }
        }
    }
    if (inv)
        for (int &x : a)
            x /= sz(a); // XOR only
}

vector<int> conv(vector<int> a, vector<int> b) {
    FST(a, 0);
    FST(b, 0);

    for (int i = 0; i < szof(a); ++i) {
        a[i] *= b[i];
    }

    FST(a, 1);
    return a;
}
```

6.4 Counting size of the maximum general matching

In order to find a size of the maximum matching:

1. Build Tutte matrix. (x_{ij} are random numbers)

$$A_{ij} = \begin{cases} x_{ij} & \text{if edge } (i,j) \text{ exists and } i < j \\ -x_{ij} & \text{if edge } (i,j) \text{ exists and } i > j \\ 0 & \textit{otherwise} \end{cases}$$

2. The size of the maximum matching equals to the size of the maximum independent set divided by 2.

3. $(A^{-1})_{ji} \neq 0$ iff edge (i,j) belongs to some complete matching.

6.5 Counting number of spanning trees

In order to count number of spanning trees:

1. Build the Laplacian matrix. That is difference between the degree matrix and the adjacency matrix.

2. Delete any row and any column of this matrix.

3. Calculate it's determinant.

6.6 Some formulas

- $\sum\limits_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$
- $\sum\limits_{i=1}^{n} i^3 = \frac{n^2(n+1)^2}{4}$
- $\sum\limits_{i=1}^{n} i^4 = \frac{n(n+1)(2n+1)(3n^2+3n-1)}{30}$
- $\sum\limits_{k=0}^{n} k\binom{n}{k} = n2^{n-1}$
- $\sum\limits_{k=0}^{n} \binom{n}{k}^2 = \binom{2n}{n}$

6.7 Miller–Rabin primality test

```
// assume p > 1
bool isprime(ll p) {
    const int a[] = {2, 3, 5, 7, 11, 13, 17, 19,
↪   23, 0};
    ll d = p - 1;
    int cnt = 0;
    while (!(d & 1)) {
        d >>= 1;
        cnt++;
    }
    for (int i = 0; a[i]; i++) {
        if (p == a[i]) {
            return true;
        }
        if (!(p % a[i])) {
            return false;
        }
    }
    for (int i = 0; a[i]; i++) {
        ll cur = mpow(a[i], d, p); // a[i] ^ d (mod
↪ p)
        if (cur == 1) {
            continue;
        }
        bool good = false;
        for (int j = 0; j < cnt; j++) {
            if (cur == p - 1) {
                good = true;
                break;
            }
            cur = mult(cur, cur);
        }
        if (!good) {
            return false;
        }
    }
    return true;
}
```

6.8 Taking by modullo (Inline assembler)

```
inline void fasterLLDivMod(ull x, uint y, uint
↪ &out_d, uint &out_m) {
    uint xh = (uint)(x >> 32), xl = (uint)x, d, m;
#ifdef __GNUC__
    asm(
```

```
    "divl %4; \n\t"
    : "=a" (d), "=d" (m)
    : "d" (xh), "a" (xl), "r" (y)
  );
#else
  __asm {
    mov edx, dword ptr[xh];
    mov eax, dword ptr[xl];
    div dword ptr[y];
    mov dword ptr[d], eax;
    mov dword ptr[m], edx;
  };
#endif
  out_d = d; out_m = m;
}
```

6.9 First solution of $(p + step \cdot x) \bmod mod < l$

```
// returns value of (p + step * x), i.e. number
↪ of steps x = (ans - p) / step (mod mod)
int smart_calc(int mod, int step, int l, int p) {
  if (p < l) {
    return p;
  }
  int d = (mod - p + step - 1) / step;
  int np = (p + d * step) % mod;
  if (np < l) {
    return np;
  }
  int res = smart_calc(step, mod % step, l, l +
↪ step - 1 - np);
  return l - 1 - res;
}
```

6.10 Multiplication by modulo in `long double`

```
ll mul(ll a, ll b, ll m) { // works for MOD 8e18
  ll k = (ll)((long double)a * b / m);
  ll r = a * b - m * k;
  if (r < 0)
    r += m;
  if (r >= m)
    r -= m;
  return r;
}
```

6.11 Numerical integration

```
function<dbl(dbl, dbl, function<dbl(dbl)>)> f =
↪ [&](dbl L, dbl R, function<dbl(dbl)> g) {
  const int ITERS = 1000000;
  dbl ans = 0;
  dbl step = (R - L) * 1.0 / ITERS;
  for (int it = 0; it < ITERS; it++) {
    double xl = L + step * it;
    double xr = L + step * (it + 1);
    dbl x1 = (xl + xr) / 2;
    dbl x0 = x1 - (x1 - xl) * sqrt(3.0 / 5);
    dbl x2 = x1 + (x1 - xl) * sqrt(3.0 / 5);
    ans += (5 * g(x0) + 8 * g(x1) + 5 * g(x2)) /
↪ 18 * step;
  }
  return ans;
};
```

6.12 Pollard's rho algorithm

```
namespace pollard {
using math::p;

vector<pair<ll, int>> getFactors(ll N) {
  vector<ll> primes;

  const int MX = 1e5;
  const ll MX2 = MX * (ll)MX;

  assert(MX <= math::maxP && math::pc > 0);

  function<void(ll)> go = [&go, &primes](ll n) {
    for (ll x : primes)
      while (n % x == 0)
        n /= x;
    if (n == 1)
      return;
    if (n > MX2) {
      auto F = [&](ll x) {
        ll k = ((long double)x * x) / n;
        ll r = (x * x - k * n + 3) % n;
        return r < 0 ? r + n : r;
      };
      ll x = mt19937_64()() % n, y = x;
      const int C = 3 * pow(n, 0.25);

      ll val = 1;
      forn(it, C) {
        x = F(x), y = F(F(y));
        if (x == y)
          continue;
        ll delta = abs(x - y);
        ll k = ((long double)val * delta) / n;
        val = (val * delta - k * n) % n;
        if (val < 0)
          val += n;
        if (val == 0) {
          ll g = __gcd(delta, n);
          go(g), go(n / g);
          return;
        }
        if ((it & 255) == 0) {
          ll g = __gcd(val, n);
          if (g != 1) {
            go(g), go(n / g);
            return;
          }
        }
      }
    }
    primes.pb(n);
  };

  ll n = N;

  for (int i = 0; i < math::pc && p[i] < MX; ++i)
    if (n % p[i] == 0) {
```

```
      primes.pb(p[i]);
      while (n % p[i] == 0)
        n /= p[i];
    }

  go(n);

  sort(primes.begin(), primes.end());

  vector<pair<ll, int>> res;
  for (ll x : primes) {
    int cnt = 0;
    while (N % x == 0) {
      cnt++;
      N /= x;
    }
    res.push_back({x, cnt});
  }
  return res;
}
} // namespace pollard
```

6.13 Polynom division and inversion

```
poly inv(poly A, int n) // returns A^-1 mod x^n
{
  assert(sz(A) && A[0] != 0);
  A.cut(n);

  auto cutPoly = [](poly &from, int l, int r) {
    poly R;
    R.v.resize(r - l);
    for (int i = l; i < r; ++i) {
      if (i < sz(from))
        R[i - l] = from[i];
    }
    return R;
  };

  function<int(int, int)> rev = [&rev](int x, int
    m) -> int {
    if (x == 1)
      return 1;
    return (1 - rev(m % x, x) * (ll)m) / x + m;
  };

  poly R({rev(A[0], mod)});
  for (int k = 1; k < n; k <<= 1) {
    poly A0 = cutPoly(A, 0, k);
    poly A1 = cutPoly(A, k, 2 * k);
    poly H = A0 * R;
    H = cutPoly(H, k, 2 * k);
    poly R1 = (((A1 * R).cut(k) + H) * (poly({0})
    - R)).cut(k);
    R.v.resize(2 * k);
    forn(i, k) R[i + k] = R1[i];
  }
  return R.cut(n).norm();
}

pair<poly, poly> divide(poly A, poly B) {
  if (sz(A) < sz(B))
    return {poly({0}), A};

  auto rev = [](poly f) {
    reverse(all(f.v));
    return f;
  };

  poly q =
      rev((inv(rev(B), sz(A) - sz(B) + 1) *
    rev(A)).cut(sz(A) - sz(B) + 1));
  poly r = A - B * q;

  return {q, r};
}
```

6.14 Polynom roots

```
const double EPS = 1e-9;
double cal(const vector<double> &coef, double x)
  {
  double e = 1, s = 0;
  for (double i : coef) s += i * e, e *= x;
  return s;
}

int dblcmp(double x) {
  if (x < -EPS) return -1;
  if (x > EPS) return 1;
  return 0;
}

double find(const vector<double> &coef, double l,
    double r) {
  int sl = dblcmp(cal(coef, l)), sr =
    dblcmp(cal(coef, r));
  if (sl == 0) return l;
  if (sr == 0) return r;
  for (int tt = 0; tt < 100 && r - l > EPS; ++tt)
    {
    double mid = (l + r) / 2;
    int smid = dblcmp(cal(coef, mid));
    if (smid == 0) return mid;
    if (sl * smid < 0) r = mid;
    else l = mid;
  }
  return (l + r) / 2;
}

vector<double> rec(const vector<double> &coef,
    int n) {
  vector<double> ret; //
    c[0]+c[1]*x+c[2]*x^2+...+c[n]*x^n, c[n]==1
  if (n == 1) {
    ret.push_back(-coef[0]);
    return ret;
  }
  vector<double> dcoef(n);
  for (int i = 0; i < n; ++i) dcoef[i] = coef[i +
    1] * (i + 1) / n;
  double b = 2; // fujiwara bound
  for (int i = 0; i <= n; ++i) b = max(b, 2 *
    pow(fabs(coef[i]), 1.0 / (n - i)));
```

```
    vector<double> droot = rec(dcoef, n - 1);
    droot.insert(droot.begin(), -b);
    droot.push_back(b);
    for (int i = 0; i + 1 < droot.size(); ++i) {
        int sl = dblcmp(cal(coef, droot[i])), sr = dblcmp(cal(coef, droot[i + 1]));
        if (sl * sr > 0) continue;
        ret.push_back(find(coef, droot[i], droot[i + 1]));
    }
    return ret;
}

vector<double> solve(vector<double> coef) {
    int n = coef.size() - 1;
    while (coef.back() == 0) coef.pop_back(), --n;
    for (int i = 0; i <= n; ++i) coef[i] /= coef[n];
    return rec(coef, n);
}
```

6.15 Simplex method

```
struct simplex_t {
    vector<vector<double>> mat;
    int EQ, VARS, p_row;

    vector<int> column;

    void row_subtract(int what, int from, double x) {
        for (int i = 0; i <= VARS; ++i)
            mat[from][i] -= mat[what][i] * x;
    }

    void row_scale(int what, double x) {
        for (int i = 0; i <= VARS; ++i)
            mat[what][i] *= x;
    }

    void pivot(int var, int eq) {
        row_scale(eq, 1. / mat[eq][var]);

        for (int p = 0; p <= EQ; ++p)
            if (p != eq)
                row_subtract(eq, p, mat[p][var]);

        column[eq] = var;
    }

    void iterate() {
        while (true) {
            int j = 0;
            for (; j != VARS and mat[EQ][j] < eps; ++j) {}

            if (j == VARS)
                break;

            double lim = 1e100;
            int arg_min = -1;

            for (int p = 0; p != EQ; ++p) {
                if (mat[p][j] < eps)
                    continue;

                double newlim = mat[p][VARS] / mat[p][j];
                if (newlim < lim)
                    lim = newlim, arg_min = p;
            }

            if (arg_min == -1)
                throw "unbounded";

            pivot(j, arg_min);
        }
    }

    simplex_t(const vector<vector<double>>& mat_): mat(mat_) {
        for (int i = 0; i < SZ(mat); ++i) // fictuous variable
            mat[i].insert(mat[i].begin() + SZ(mat[i]) - 1, double(0));

        EQ = SZ(mat), VARS = SZ(mat[0]) - 1;
        column.resize(EQ, -1);
        p_row = 0;

        for (int i = 0; i < VARS; ++i) {
            int p;
            for (p = p_row; p < EQ and abs(mat[p][i]) < eps; ++p) {}

            if (p == EQ)
                continue;

            swap(mat[p], mat[p_row]);
            column[p_row] = i;
            row_scale(p_row, 1. / mat[p_row][i]);

            for (p = 0; p != EQ; ++p)
                if (p != p_row)
                    row_subtract(p_row, p, mat[p][i]);

            p_row += 1;
        }

        for (int p = p_row; p < EQ; ++p)
            if (abs(mat[p][VARS]) > eps)
                throw "unsolvable (bad equalities)";

        if (p_row) {
            int minr = 0;
            for (int i = 0; i < p_row; ++i)
                if (mat[i][VARS] < mat[minr][VARS])
                    minr = i;

            if (mat[minr][VARS] < -eps) {
                mat.push_back(vector<double>(VARS + 1));

                mat[EQ][VARS - 1] = -1;
                for (int i = 0; i != p_row; ++i)
                    mat[i][VARS - 1] = -1;
```

```
        pivot(VARS - 1, minr);
        iterate();

        if (abs(mat[EQ][VARS]) > eps)
          throw "unsolvable";

        for (int c = 0; c != EQ; ++c)
          if (column[c] == VARS - 1) {
            int p = 0;
            while (p != VARS - 1 and abs(mat[c][p]) < eps)
              ++p;

            assert(p != VARS - 1);
            pivot(p, c);
            break;
          }

        for (int p = 0; p != EQ; ++p)
          mat[p][VARS - 1] = 0;

        mat.pop_back();
      }
    }
  }

  double solve(vector<double> coeff, vector<double>& pans) {
    auto mat_orig = mat;
    auto col_orig = column;

    coeff.resize(VARS + 1);
    mat.push_back(coeff);

    for (int i = 0; i != p_row; ++i)
      row_subtract(i, EQ, mat[EQ][column[i]]);

    iterate();

    auto ans = -mat[EQ][VARS];
    if (not pans.empty()) {
      for (int i = 0; i < EQ; ++i) {
        assert(column[i] < VARS);
        pans[column[i]] = mat[i][VARS];
      }
    }

    mat = std::move(mat_orig);
    column = std::move(col_orig);
    return ans;
  }

  double solve_min(vector<double> coeff, vector<double>& pans) {
    for (double& elem: coeff)
      elem = -elem;

    return -solve(coeff, pans);
  }
};
```

6.16 Some integer sequences

Bell numbers:

n	B_n	n	B_n
0	1	10	115 975
1	1	11	678 570
2	2	12	4 213 597
3	5	13	27 644 437
4	15	14	190 899 322
5	52	15	1 382 958 545
6	203	16	10 480 142 147
7	877	17	82 864 869 804
8	4 140	18	682 076 806 159
9	21 147	19	5 832 742 205 057

Numbers with many divisors:

$x \leq$	x	$d(x)$
20	12	6
50	48	10
100	60	12
1000	840	32
10 000	9 240	64
100 000	83 160	128
10^6	720 720	240
10^7	8 648 640	448
10^8	91 891 800	768
10^9	931 170 240	1 344
10^{11}	97 772 875 200	4 032
10^{12}	963 761 198 400	6 720
10^{15}	866 421 317 361 600	26 880
10^{18}	897 612 484 786 617 600	103 680

Partitions of n into unordered summands

n	$a(n)$	n	$a(n)$	n	$a(n)$
0	1	20	627	40	37 338
1	1	21	792	41	44 583
2	2	22	1 002	42	53 174
3	3	23	1 255	43	63 261
4	5	24	1 575	44	75 175
5	7	25	1 958	45	89 134
6	11	26	2 436	46	105 558
7	15	27	3 010	47	124 754
8	22	28	3 718	48	147 273
9	30	29	4 565	49	173 525
10	42	30	5 604	50	204 226
11	56	31	6 842	51	239 943
12	77	32	8 349	52	281 589
13	101	33	10 143	53	329 931
14	135	34	12 310	54	386 155
15	176	35	14 883	55	451 276
16	231	36	17 977	56	526 823
17	297	37	21 637	57	614 154
18	385	38	26 015	58	715 220
19	490	39	31 185	59	831 820
100	190 569 292				

7 Strings

7.1 Aho-Corasick

```
const int N = 1e6 + 7;
const int A = 26;
```

```
struct Node {
    int nxt[A];
    int term;
    int par;
    int par_c;
    int go[A];
    int suf;
    int sup;

    Node() {
        fill(nxt, nxt + A, -1);
        term = 0;
        par = -1;
        par_c = -1;
        fill(go, go + A, -1);
        suf = -1;
        sup = -1;
    }
} trie[N];
int root = 0, sz_t = 1;

void add(string s) {
    int v = root;
    for (int i = 0; i < (int)s.size(); ++i) {
        s[i] -= 'a';
        if (trie[v].nxt[s[i]] == -1) {
            trie[v].nxt[s[i]] = sz_t;
            trie[sz_t] = Node();
            trie[sz_t].par = v;
            trie[sz_t].par_c = s[i];
            ++sz_t;
        }
        v = trie[v].nxt[s[i]];
    }
    ++trie[v].term;
}

vector<int> order;

void build() {
    order.clear();
    order.push_back(root);
    queue<int> q;
    for (int i = 0; i < A; ++i) {
        int u = trie[root].nxt[i];
        if (u != -1) {
            trie[root].go[i] = u;
            q.push(u);
        } else {
            trie[root].go[i] = root;
        }
    }
    while (!q.empty()) {
        int v = q.front();
        q.pop();
        order.push_back(v);
        if (trie[v].par == root) {
            trie[v].suf = root;
        } else {
            trie[v].suf =
    trie[trie[trie[v].par].suf].go[trie[v].par_c];
        }
        trie[v].sup =
            (trie[trie[v].suf].term ? trie[v].suf
    : trie[trie[v].suf].sup);
        for (int i = 0; i < A; ++i) {
            int u = trie[v].nxt[i];
            if (u != -1) {
                trie[v].go[i] = u;
                q.push(u);
            } else {
                trie[v].go[i] =
    trie[trie[v].suf].go[i];
            }
        }
    }
}

int go(string s) {
    int v = root;
    for (int i = 0; i < (int)s.size(); ++i) {
        v = trie[v].go[s[i] - 'a'];
    }
    return v;
}
```

7.2 Manacher's algorithm

```
// returns vector ret of length (|s| * 2 - 1),
//   ret[i * 2] -- maximal length of palindrome
    with center in i-th symbol
//   ret[i * 2 + 1] -- maximal length of
    palindrome with center between i-th and (i +
    1)-th symbols
vector<int> find_palindromes(string const& s) {
    string tmp;
    for (char c : s) {
        tmp += c;
        tmp += '!';
    }
    tmp.pop_back();

    int c = 0, r = 1;
    vector<int> rad(szof(tmp));
    rad[0] = 1;
    for (int i = 1; i < szof(tmp); ++i) {
        if (i < c + r) {
            rad[i] = min(c + r - i, rad[2 * c - i]);
        }
        while (i - rad[i] >= 0 && i + rad[i] <
    szof(tmp) && tmp[i - rad[i]] == tmp[i +
    rad[i]]) {
            ++rad[i];
        }
        if (i + rad[i] > c + r) {
            c = i;
            r = rad[i];
        }
    }

    for (int i = 0; i < szof(tmp); ++i) {
        if (i % 2 == 0) {
            rad[i] = (rad[i] + 1) / 2 * 2 - 1;
        } else {
```

```
| | | rad[i] = rad[i] / 2 * 2;
| | }
| }

| return rad;
}
```

7.3 Min Cyclic Shift O(n)

```
string min_cyclic_shift(string s) {
| s += s;
| int n = s.size();
| int i = 0, ans = 0;
| while (i < n / 2) {
| | ans = i;
| | int j = i + 1, k = i;
| | while (j < n && s[k] <= s[j]) {
| | | if (s[k] < s[j])
| | | | k = i;
| | | else
| | | | ++k;
| | | ++j;
| | }
| | while (i <= k)  i += j - k;
| }
| return s.substr(ans, n / 2);
}
```

7.4 Suffix array + LCP

```
vector<int> build_suffarr(string s) {
| int n = szof(s);
| auto norm = [&](int num) {
| | if (num >= n) {
| | | return num - n;
| | }
| | return num;
| };
| vector<int> classes(s.begin(), s.end()),
↪ n_classes(n);
| vector<int> order(n), n_order(n);
| iota(order.begin(), order.end(), 0);
| vector<int> cnt(max(szof(s), 128));
| for (int num : classes) {
| | cnt[num + 1]++;
| }
| for (int i = 1; i < szof(cnt); ++i) {
| | cnt[i] += cnt[i - 1];
| }

| for (int i = 0; i < n; i = i == 0 ? 1 : i * 2)
↪ {
| | for (int pos : order) {
| | | int pp = norm(pos - i + n);
| | | n_order[cnt[classes[pp]]++] = pp;
| | }
| | int q = -1;
| | pii prev = {-1, -1};
| | for (int j = 0; j < n; ++j) {
| | | pii cur = {classes[n_order[j]],
↪ classes[norm(n_order[j] + i)]};
| | | if (cur != prev) {
| | | | prev = cur;
| | | | ++q;
| | | | cnt[q] = j;
| | | }
| | | n_classes[n_order[j]] = q;
| | }
| | swap(n_classes, classes);
| | swap(n_order, order);
| }
| return order;
}

void solve() {
| string s;
| cin >> s;
| s += "$";
| auto suffarr = build_suffarr(s);

| vector<int> where(szof(s));
| for (int i = 0; i < szof(s); ++i) {
| | where[suffarr[i]] = i;
| }

| vector<int> lcp(szof(s));
| int cnt = 0;
| for (int i = 0; i < szof(s); ++i) {
| | if (where[i] == szof(s) - 1) {
| | | cnt = 0;
| | | continue;
| | }
| | cnt = max(cnt - 1, 0);
| | int next = suffarr[where[i] + 1];
| | while (i + cnt < szof(s) && next + cnt <
↪ szof(s) && s[i + cnt] == s[next + cnt]) {
| | | ++cnt;
| | }
| | lcp[where[i]] = cnt;
| }
}
```